

# Systems for Usable Machine Learning

by

Alexandra Zytek

B.S., Rensselaer Polytechnic Institute (2018)

S.M., Massachusetts Institute of Technology (2021)

Submitted to the Department of Electrical Engineering and Computer Science  
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2025

© 2025 Alexandra Zytek. All rights reserved.

The author hereby grants to MIT a nonexclusive, worldwide, irrevocable, royalty-free license to exercise any and all rights under copyright, including to reproduce, preserve, distribute and publicly display copies of the thesis, or release the thesis under an open-access license.

Authored by: Alexandra Zytek  
Department of Electrical Engineering and Computer Science  
January 18, 2025

Certified by: Kalyan Veeramachaneni  
Principal Research Scientist, Laboratory for Information and Decision Systems  
Thesis Supervisor

Accepted by: Leslie Kolodziejcki  
Professor of Electrical Engineering and Computer Science  
Chair, Department Committee on Graduate Students



# Systems for Usable Machine Learning

by

Alexandra Zytek

Submitted to the Department of Electrical Engineering and Computer Science  
on January 18, 2025 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

## ABSTRACT

Many real-world decision problems are complex, with outcomes difficult to measure and evaluate. The impact of decisions made in these domains is nuanced and takes a long time to be fully realized. Individual mistakes can lead to significant costs, and computational tools such as ML models must be integrated alongside existing, well-established human workflows. These properties of such decision problems means that ML solutions must be *usable* in order to be effective — in other words, developed and deployed in such a way as to be used by humans in decision-making and improve outcomes. In order improve ML usability, developers create ML tools, or diverse kinds of interfaces that allow users to understand ML models and their predictions.

In this thesis, we use real-world case studies to synthesize generalizable lessons for applying usable ML tools to complex, real-world decision problems. Based on experience developing ML tools for child welfare screening, we propose a formal taxonomy of feature properties related to usability and interpretability. We then discuss the design and development of a system to make generating ML explanations that use such interpretable features more effective. Pyreal is a framework and Python library implementation that uses updated data transformers to generate explanations of ML models and predictions using interpretable features.

Motivated by the development and customization effort required to develop ML tools for new applications, we then discuss the development of Sibyl, a configurable and comprehensive system for generating usable ML interfaces for a wide range of applications. We then discuss our case study in applying Sibyl to the decision problem of wind turbine monitoring.

We then discuss Explingo, our system for transforming traditional ML explanations into natural language narratives to further improve the usability of ML outputs.

We finish by discussing the practical lessons this work demonstrates related to the need for usable ML, the challenges specific to these complex applications, ethical questions, and future directions.

Thesis supervisor: Kalyan Veeramachaneni

Title: Principal Research Scientist, Laboratory for Information and Decision Systems



# Acknowledgments

A PhD is never truly completed alone, and a lot of mentorship, guidance, and support went into the creation of this one.

I would like to start by thanking my thesis supervisor Kalyan Veeramachaneni for all the guidance throughout this PhD process. You always kept me focused on the practical and impactful, and I believe my work is better for it. I would also like to thank the rest of my thesis committee, Sara Beery and Manish Raghaven, for their invaluable insights and discussions.

I was blessed with a large number of wonderful collaborators when working on the research in this thesis. Thank you to Dongyu Liu for his constant feedback and guidance on the ML tools and evaluations conducted in this work. Thank you to Sara Pido and Sarah Alnegheimish for all the support and inspiration on the Explingo project, and being wonderful collaborators and friends. I feel incredibly lucky to have had the opportunity to work with Wei-En (Warren) Wang for more than four years; your constant ideas and initiative contributed significantly to the Sibyl and Pyreal systems, as well as my own inspiration towards research.

I would like to offer the deepest thanks to everyone involved in the child welfare screening project, which inspired most of the work in this thesis. Thank you especially to Rhema Vaithianathan - I have never worked with someone with such a nuanced and thoughtful approach to research. Thank you as well to the rest of the CSDA team and other collaborators on the project including Larissa Lorimer, Diana Benavides Prado, Marie-Pascale Grimon, Chriss Mills, and Megh Mayur. I would also like to extend a deep thank you to the Larimer Country child welfare screening team for all the feedback and discussion.

Thank you to the Iberdrola team, including Sofia Koukoura, Rob Jones, and Liam Faulkner for their continuous feedback, guidance, and constant willingness to support this research. It was so wonderful to get the chance to work with such engaged and brilliant collaborators.

Thank you to all the anonymous user study participants for all the rich feedback — both the positive and the negative — without whom no human-centered research would be possible.

One cannot underestimate the value of good labmates; I had great ones, who made the research rewarding and offered inspiration and learning. Thank you to my fellow PhD students Micah Smith, Lei Xu, and Alicia Sun. And an additional thank you to all the wonderful MEngs and visiting students throughout the years including Frances, Grace, Linh, Michael, Nassim, Alex, Andy, and Romain.

A thesis also involves a lot of extra work behind the scenes. Thank you to Cara Giaimo,

for not just contributing greatly to the writing of this thesis, but for all the deeper insights. Thank you to Arash Akhgari for his work on the figures in this thesis, without which it would certainly be a less appealing and clear document. Thank you to Michaela Henry, not only for her incredible ability to organize the chaos of an academic lab, but also for her invaluable insights. Thank you to Katie O'Reilly all the organization and logistic help. Thank you to Leslie Kolodziejski all the support.

A PhD is difficult in more ways than just academic ones, and nothing is more valuable than good friends and family. Thank you to my fellow members of the climbing team - especially Isaac, Jack, Katie, and Nora - and the European club; these groups were such important sources of joy (and exercise, and free food). Thank you to all the Stephens throughout the years for teaching me what it means to grow and thrive (for the most part). I cannot offer enough thank you to all my fellow PhD students. Thank you to Isaac, Kevin, Camille, Stella, Kru, Anish, Serena, Bazyli, Peter, Josh, James, Graeme, Sarah, John, Jonas, and all the other lovely MIT grad students I have been so lucky to get to spend my days with. I'd like to give an extra special thank you to Alice and Luke for being my oldest friends, constantly there for me since before I considered a PhD a possibility.

Thank you to Asher for being the best dev-ops manager I could imagine. You are genuinely the only reason working on my systems was such a smooth, pleasant, and productive process. And of course, thank you also for being such an endless source of support, adventure, and mushroom photos.

Finally, enormous thank you to my sister Alice, for showing me that I could be so much more than I once thought, to my sister Adi for being my eternal best friend, and of course to my parents for their endless support, love, and encouragement, and without whom I most literally could not have completed this PhD.

# Contents

<b>Title page</b>	<b>1</b>
<b>Abstract</b>	<b>3</b>
<b>Acknowledgments</b>	<b>5</b>
<b>List of Figures</b>	<b>11</b>
<b>List of Tables</b>	<b>13</b>
<b>1 Introduction</b>	<b>15</b>
1.1 Usable ML . . . . .	16
1.2 Considerations for Usable ML: People . . . . .	18
1.2.1 Decision-makers . . . . .	20
1.2.2 Bridges . . . . .	21
1.2.3 ML Developers . . . . .	22
1.2.4 ML Contributors . . . . .	22
1.3 Considerations for Usable ML: Processes . . . . .	22
1.4 Technical Challenges in Usable ML . . . . .	24
1.5 Problem Statement and Contributions . . . . .	25
<b>2 Background</b>	<b>27</b>
2.1 Human-Centered ML . . . . .	27
2.2 Explainable ML . . . . .	28
2.3 Explainable ML for Decision Making . . . . .	30
2.4 ML Systems for Building ML Tools . . . . .	31
<b>3 Application 1: Usable ML for Child Welfare Screening</b>	<b>33</b>
3.1 Understanding Context and Decision-maker Needs . . . . .	34
3.1.1 Observing Decision-Making Without ML to Understanding the People and Processes . . . . .	34
3.1.2 Observing Screening with ML to Understand ML Usability Challenges	36
3.2 Developing and Iterating on ML Tool Interface Design . . . . .	38
3.3 Evaluating ML Tool Interfaces with Simulated Screening Session . . . . .	38
3.3.1 Study Results . . . . .	40
3.4 Discussion . . . . .	44

3.4.1	Accurate explanations may be more useful than faithful ones . . . . .	44
3.4.2	Interpretable features are essential for understanding . . . . .	44
3.4.3	The case study highlights several challenges related to systems for usable ML . . . . .	45
<b>4</b>	<b>Interpretable Features</b>	<b>47</b>
4.1	Case Studies in ML Applications . . . . .	48
4.1.1	Education: Abstracting Features . . . . .	48
4.1.2	Cybersecurity: Tracking Features . . . . .	49
4.1.3	Healthcare: Disaggregating Features . . . . .	50
4.1.4	Satellite Monitoring: Marking Real Features . . . . .	50
4.2	Key Lessons . . . . .	51
4.3	Formal Literature Review . . . . .	52
4.4	Taxonomy of Machine Learning Features . . . . .	53
4.4.1	Model-Focused Users: The Model-Ready Feature Space . . . . .	54
4.4.2	Decision Problem-Focused Users: The Interpretable Feature Space . . . . .	55
4.4.3	Feature Properties: All Users . . . . .	56
4.5	Feature Transforms . . . . .	57
4.5.1	Model-ready transforms . . . . .	58
4.5.2	Interpretable transforms . . . . .	58
4.6	Discussion . . . . .	60
4.6.1	Using feature properties . . . . .	60
4.6.2	Risks . . . . .	60
4.6.3	Shortcomings from existing systems . . . . .	61
<b>5</b>	<b>Pyreal: A System for Interpretable Explanations</b>	<b>63</b>
5.1	An Updated Transformer . . . . .	65
5.1.1	Transforming for Interpretability . . . . .	66
5.2	Defining Feature Spaces . . . . .	67
5.3	The Interpretable Explanation Algorithm . . . . .	68
5.3.1	Formal Equation for the Pyreal Algorithm . . . . .	70
5.4	RealApps: The Interpretable Explanation Algorithm Encapsulated . . . . .	71
5.4.1	Implementation . . . . .	72
5.5	Evaluation . . . . .	76
5.5.1	Evaluating the Interpretability of Pyreal Explanations . . . . .	77
5.5.2	Directly comparing the interpretability of Pyreal and no-Pyreal Explanations . . . . .	81
5.5.3	Evaluating the Usability of the Pyreal library . . . . .	82
5.5.4	Evaluating the Runtime Performance of Explanation Generation . . . . .	87
5.6	Discussion . . . . .	88
<b>6</b>	<b>Sibyl: A System for Usable ML</b>	<b>91</b>
6.1	Requirements of a System for Usable ML . . . . .	92
6.1.1	Inputs . . . . .	93
6.1.2	Outputs . . . . .	93



6.1.3	Interactions . . . . .	94
6.1.4	Configurations . . . . .	96
6.2	Sibyl System . . . . .	98
6.2.1	Sibyl-API . . . . .	98
6.2.2	Sibylapp . . . . .	101
6.3	Case Studies . . . . .	102
6.4	Discussion . . . . .	107
<b>7</b>	<b>Application 2: Usable ML for Wind Turbine Monitoring</b>	<b>111</b>
7.1	Data and Modelling . . . . .	112
7.2	Understanding Context and End-User Needs . . . . .	114
7.2.1	Process . . . . .	115
7.3	Developing Usable ML Interfaces . . . . .	116
7.3.1	User Feedback . . . . .	116
7.3.2	Interfaces . . . . .	116
7.4	Evaluation . . . . .	120
7.4.1	Method: Simulated Case Study . . . . .	120
7.5	Discussion . . . . .	125
<b>8</b>	<b>Explingo: System for Generating and Validating Natural-Language AI Explanations</b>	<b>129</b>
8.1	Background . . . . .	131
8.2	The EXPLINGO System: NARRATOR and GRADER . . . . .	131
8.2.1	NARRATOR: Transforming ML Explanations to Narratives . . . . .	131
8.2.2	GRADER: Assessing the Quality of Narratives . . . . .	132
8.2.3	Using EXPLINGO for a New Application . . . . .	134
8.3	Exemplar and Evaluation Datasets . . . . .	135
8.4	Automated Grading of Narratives . . . . .	136
8.4.1	Accuracy . . . . .	137
8.4.2	Completeness . . . . .	138
8.4.3	Fluency . . . . .	139
8.4.4	Conciseness . . . . .	141
8.5	Optimizing the Narrator . . . . .	141
8.6	Results . . . . .	142
8.7	Discussion and Future Work . . . . .	145
8.7.1	System Integration and Open-Source Implementation . . . . .	145
8.7.2	Lessons from Adjusting the GRADER . . . . .	146
8.7.3	Future Work . . . . .	146
<b>9</b>	<b>Discussion and Future Work</b>	<b>149</b>
9.1	Lesson 1: Identify the decision-makers and users of ML first . . . . .	150
9.2	Lesson 2: Many properties of real-world decision problems affect how ML is used . . . . .	151
9.3	Lesson 3: Research in real-world, long-term applications uncovers rich insights	153

9.4	Lesson 4: Progress in usable ML is led by building systems to address known challenges — deeper challenges are discovered after . . . . .	153
9.5	Lesson 5: Consider usability, not just explainability . . . . .	154
9.6	Lesson 6: The parameters of Type 2 decision problems come with unique challenges . . . . .	155
9.7	Ethical considerations posed by this work . . . . .	156
9.8	Future Extensions . . . . .	157
9.8.1	Formal Deployment Evaluation . . . . .	157
9.8.2	Adding Data Modalities . . . . .	158
<b>10</b>	<b>Conclusion</b>	<b>159</b>
<b>A</b>	<b>Glossary</b>	<b>161</b>
A.1	People . . . . .	161
A.2	Processes . . . . .	161
A.3	Systems and Artifacts . . . . .	162
A.4	Adjectives . . . . .	162
<b>B</b>	<b>Pyreal Additional Contents</b>	<b>165</b>
B.1	Code for Generating Interpretable Explanation without Pyreal . . . . .	165
<b>C</b>	<b>Sibyl Additional Contents</b>	<b>167</b>
C.1	Screenshots of Sibylapp Interfaces . . . . .	167
	<b>References</b>	<b>175</b>

# List of Figures

1.1	Summary of the properties of complex decision problems we focus on in this thesis. . . . .	16
1.2	The usable ML process. . . . .	18
3.1	The process we followed for the child welfare screening case study. . . . .	34
3.2	The child welfare screening process. . . . .	37
3.3	The final version of the "Details" Sibyl interface for child welfare screening. . . . .	39
4.1	Summary of our feature taxonomy . . . . .	48
5.1	A hypothetical example to illustrate the benefits of interpretable explanations. . . . .	64
5.2	The interpretable explanation transformer pipeline. . . . .	69
5.3	Decision-maker workflow with Pyreal. . . . .	72
5.4	ML Developer workflow with Pyreal. . . . .	73
5.5	Pyreal class structure. . . . .	76
5.6	Sample figures shown to users in the explanation-interpretability user study. . . . .	78
5.7	Summary of Pyreal explanation average usefulness scores compared to non-Pyreal alternatives. . . . .	79
5.8	Tutorial code given to participants in Pyreal library usability study. . . . .	83
5.9	Distribution of times to generate explanations in Pyreal developer study. . . . .	85
5.10	Result of retrospective questions asked to participants in our developer user study. . . . .	86
5.11	Average percent change in explanation generation time for generating an interpretable explanation with Pyreal versus without Pyreal . . . . .	87
6.1	Sibyl system diagram. . . . .	98
6.2	Entity relational diagram for the Sibyl database. . . . .	101
6.3	Configuring Sibyl for child welfare. . . . .	104
6.4	Screenshot from Sibyl applied to child welfare. . . . .	104
6.5	Snippet of setup wizard for house pricing application. . . . .	105
6.6	Sample snippet of an interface from Sibylapp for the housing sample application. . . . .	106
6.7	Feature-level plot explaining how the ML model uses the <b>Average Precipitation per Year (cm)</b> feature for the emissions sample application. . . . .	108
6.8	Configuration file approach to preparing the emissions Sibyl application. . . . .	109

7.1	Visualization of the interactions between the cut-off time, the target time, and the lead time over two possible prediction scenarios. . . . .	112
7.2	Summary of the wind turbine monitoring process. . . . .	116
7.3	Wind turbine application context configuration. . . . .	117
7.4	Sibylapp prediction summary interface for brakepad failure prediction. . . . .	118
7.5	Sibylapp Explore a Prediction interface for wind turbine monitoring. . . . .	119
7.6	Summary of reported usefulness of each Sibyl interface in the simulated case study. The average score considers “did not use” as a score of 0. . . . .	123
7.7	Summary of distribution of time spent on each interface during the simulated case study. Overall, participants spent the most time exploring individual predictions and prediction summaries. . . . .	124
8.1	Sample Explingo NARRATOR inputs and outputs. . . . .	130
8.2	Prompt passed to the Explingo GRADER to compute the accuracy metric. . . . .	136
8.3	Prompt passed to the Explingo GRADER to compute the completeness metric. . . . .	137
8.4	Prompt passed to the Explingo GRADER to compute the fluency metric. . . . .	139
8.5	Evaluation of the Explingo GRADER on fluency. . . . .	140
8.6	Correlation between mean metric grades over all datasets . . . . .	144
8.7	Open-source library implementation of narrative generation process. . . . .	146
B.1	Equivalent code required to generate feature contribution explanation without Pyreal . . . . .	166
C.1	Overall structure of a Sibylapp tool, including the following components. . . . .	167
C.2	The main explanation interface contents for the Explore a Prediction interface. . . . .	168
C.3	The main explanation interface contents for the Similar Entities (Houses) interface. . . . .	169
C.4	The main explanation interface contents for the Compare Entities interface. . . . .	170
C.5	The main explanation interface contents for the Experiment with Changes interface. . . . .	171
C.6	The main explanation interface contents for the Understand the Model – Feature Importance interface. . . . .	172
C.7	The main explanation interface contents for the Understand the Model – Global Contributions interface. . . . .	173
C.8	The main explanation interface contents for the Understand the Model – Explore a Feature interface. . . . .	174

# List of Tables

1.1	Applications we applied usable ML to in this thesis. . . . .	17
1.2	Considerations for usable ML, as applied to housing pricing application. . . .	19
1.3	Roles and people collaborated with. . . . .	23
3.1	List of usability challenges that impact the usability of ML models. . . . .	35
3.2	The usable ML considerations for child welfare screening. . . . .	36
3.3	The proposed ML tool interfaces for child welfare screening, and the challenges they were theorized to address. . . . .	40
3.4	Summary of factors causing increased or decreased trust in the model for child welfare. . . . .	43
3.5	Key findings about specific challenges to ML usability that we seek to address in the rest of this thesis. The chapter column gives the chapter in this thesis where we aim to address each challenge. . . . .	45
4.1	Summary of example applications referenced in developing our taxonomy for interpretable features. . . . .	49
4.2	Summarized results of the methodological literature review on interpretable features. . . . .	52
4.3	Summary of the properties including in our formal feature taxonomy. . . . .	54
4.4	Example of feature transforms using the Forest Cover dataset . . . . .	62
5.1	Summary of properties of Pyreal transformers . . . . .	65
5.2	Sample data and inverse explanation transforms applied to two categories of explanations. . . . .	66
5.3	List of Pyreal Explainer classes currently supported. . . . .	74
5.4	List of Pyreal Transformer classes currently supported. . . . .	74
5.5	List of Pyreal Explanation Types currently supported. . . . .	75
5.6	Summary of participant groups in the evaluation of explanation interpretability. . . . .	77
5.7	Summary of participants in each condition of our library usability study. . . . .	82
5.8	Summary of mistakes made by participants in the no-Pyreal condition of our developer study. . . . .	84
6.1	List of interactions users have with ML. . . . .	95
6.2	Configurations to be considered when developing ML tools for new domains. . . . .	97
6.3	List of Sibyl-API routes. . . . .	99
6.4	List of Sibyl-API routes. . . . .	102

7.1	Considerations for usable ML for wind turbine monitoring. . . . .	113
7.2	Summary of Sibyl evaluation metrics for brake pad failure prediction simulated study. . . . .	121
7.3	List of interactions and configurations to add, based on wind turbine monitoring case study. . . . .	126
8.1	Summary of Explingo exemplar datasets. . . . .	133
8.2	Summary of metric validation datasets for the accuracy and completeness metrics. . . . .	135
8.3	Agreement between GRADER and human-labeled validation set for Accuracy and Completeness. . . . .	138
8.4	Narrative quality scores for narratives generated by our NARRATOR. . . . .	143
8.5	Narrative quality scores generated by our NARRATOR in our two highest-scoring conditions, averaged by dataset. . . . .	145

# Chapter 1

## Introduction

There is a large category of decision problems that are too complex and important for simple, automated ML solutions to contribute. Improving outcomes in these complex decision problems requires more thoughtfully developed decision support systems that, rather than behaving completely autonomously, instead augment human decision processes.

These complex decision problems contrast many of the problems that ML has been successfully applied to, like short-time-horizon stock trading and video recommendation systems, which we label Type 1 decision problems. They have simple outcomes that can be observed in a short timeframe and are straightforward to evaluate. Individual decisions associated with Type 1 problems tend to be low risk. Because of these properties, it is relatively simple to apply computational approaches, such as machine learning (ML), to these problems.

The category of decision problems that we focus on in this thesis — which we dub Type 2 decision problems — is characterized by several key properties, summarized in Figure 1.1. They have complex outcomes that are difficult to measure, and take time to be fully realized. Single mistakes can have significant costs. ML models developed for these decision problems usually have their outputs deployed alongside human decision-makers, who incorporate ML outputs alongside rich external information and ultimately make decisions themselves. Evaluating the impact of these ML outputs is difficult and time-consuming, and requires going far beyond traditional ML performance metrics. Examples of such Type 2 decision problems are broad, and include making medical diagnoses, deciding when to offer bank loans, making university application decisions, making legal decisions, and deciding when to investigate potential cases of child abuse.

Because Type 2 decision problems involve the active use of ML outputs in human decision processes, these outputs often need to be integrated into larger decision support systems. These systems include the ML outputs as well as other **ML augmentations** such as explanations, data visualizations and uncertainty quantification metrics. The objective of these systems is to provide an essential property of ML — what we call **usability**, or the ability of ML to improve human decision-making. These augmentations are combined in interactive interfaces we call **usable ML tools** (or simply *ML tools*)<sup>1</sup>.

This thesis seeks to thoroughly develop the concept of usable ML by 1) formally defin-

---

<sup>1</sup>Throughout this thesis, we use the term *ML tool* to specifically refer to ML interfaces that display ML outputs and other helpful augmentations. This is in contrast to other uses of the term ML tools which include libraries such as scikit-learn that are used to develop ML models.

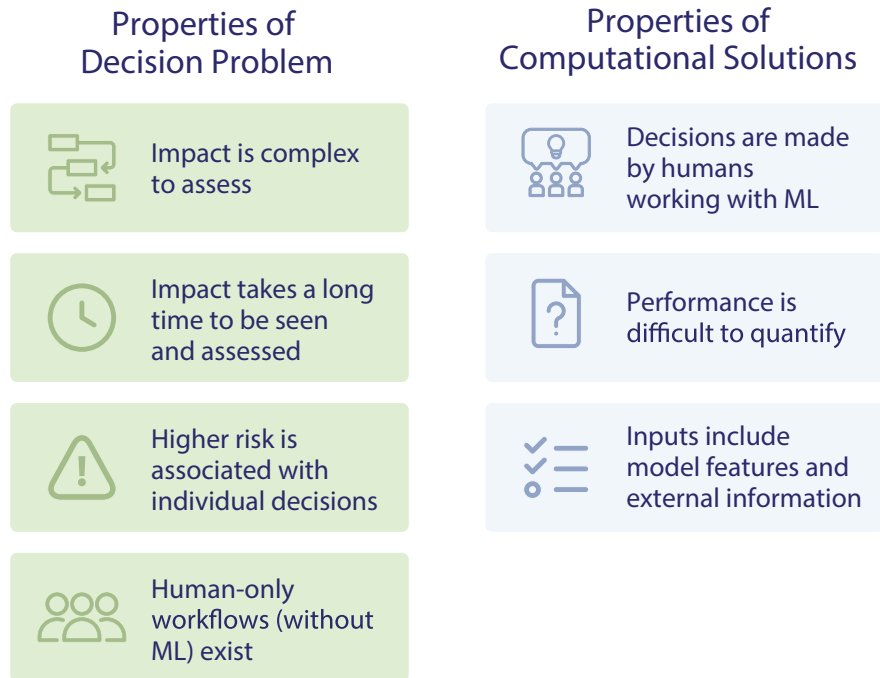


Figure 1.1: Summary of the properties of Type 2 decision problems that is the focus of this thesis.

ing usable ML and motivating its importance, 2) defining the types of decision problems that require usable ML, and 3) identifying and taking steps to address the key technical challenges involved in developing usable ML tools. In particular, we motivate the need for considering the usability of ML tools through multiple case studies in real-world applications, introduce novel systems for developing ML tools, and evaluate these tools in more real-world applications.

## 1.1 Usable ML

Our formal definition of ML usability is as follows: ML models, outputs, and tools are usable if they 1) are incorporated in human decision making in order to address a problem and 2) have a measurable, positive impact on the outcome of the problem.

To better understand this definition, consider the following example of a decision problem where usable ML is required. A real estate agent regularly uses a ML tool developed by an external company. This system includes outputs from an ML model, called the Ames Housing model, which predicts home sale prices in Ames, Iowa based on factors such as the house’s size, location, and utilities. The real estate agent works with prospective homebuyers to better understand their needs and desires and help them bid on houses for sale. This situation meets our parameters for a Type 2 problem: A bad decision can have significant negative impacts for the homebuyer, who may miss out on a good home by underbidding or overpay by overbidding. Evaluating how good the final bidding decision was is complicated, and depends in part on long-term and nuanced opinions of homebuyers. The real estate agent,



Table 1.1: Applications we applied usable ML to in this thesis.

<b>Application</b>	<b>Decision Problem</b>	<b>ML Prediction Problem</b>	<b>Chapters</b>
House Pricing	Given factors a house such as size, location, and utilities, decide on an appropriate bid price for a prospective homebuyer.	Predict the current value of house in dollars	1
Child Welfare Screening	Given a report of potential child abuse, including a description of the case given by a concerned party as well as background information about the child and other relevant individuals, decide whether to screen-in the case for further investigation.	Predict a 1-20 risk score, representing the probability that the child will be removed from their home in the next two years if screened in and investigated	3
Wind Turbine Monitoring	Given recent signal information about a wind turbine, decide whether any manual inspections or maintenance of the brake pad are required.	Predict the probability that a turbine brake pad will experience failure within various time frames.	7

working with homebuyers, makes the final decision about how much to bid — a complex consideration that includes not only the information about the house that the model takes into account, but also homebuyer sentiment and preferences.

Now, consider a product manager at the company that makes the ML tool the real estate agent uses. Their job is to ensure that the real estate agents using their system are able to do so effectively, so it can improve their decision-making. In other words, the product manager’s goal is to make their ML tool usable. To do so, they need to consider requirements such as:

1. The ML model must have useful and sufficiently accurate outputs (in this case, of house sale price).
2. The ML tool must include any information the decision-makers (real estate agents and homebuyers) expect and need, presented in a way that they can easily understand.
3. The real estate agents and homebuyers should be able to get this information they expect and need from the ML tool without making significant changes to their existing decision process

Considering requirements such as these is a key component of ensuring ML tools are usable. This thesis discusses several case studies that more thoroughly reveal the challenges and requirements of usable ML tools, as well as introducing systems that make meeting these requirements easier. In particular, meeting these requirements requires carefully considering certain properties of the application.

Developing usable ML is complicated, as people working in different decision problems may have wildly different expertise and requirements. For example, our real estate agent may get confused by complex ensemble-model visualisations, while an ML developer using a model for their own work may find such a visualization useful and illuminating. Similarly, a real estate agent working on a timeline of a few days to place a bid on a house may find

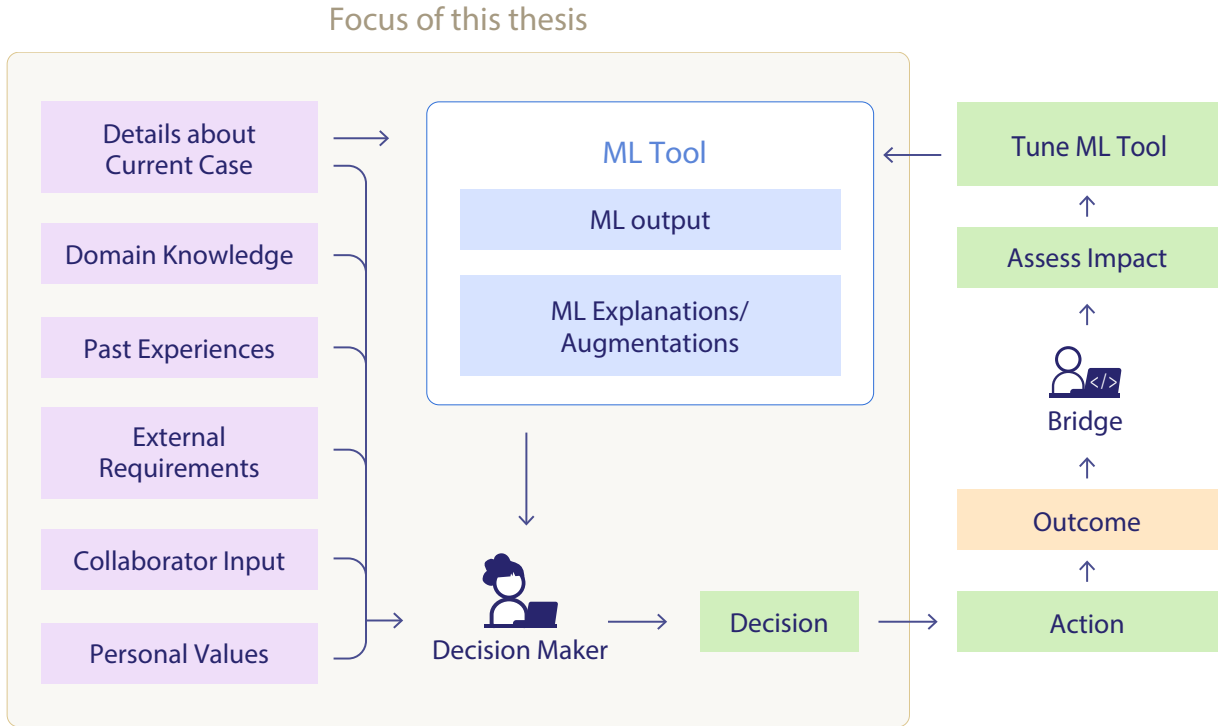


Figure 1.2: The usable ML process. Decision-makers take in a wide variety of information, along with the ML outputs and information in the ML tool to make decisions.

detailed information more useful than a doctor who needs to act within a few minutes. In every case, usable ML tools need to seamlessly integrate into existing decision processes if they are to improve decision outcomes. Throughout this thesis, we delve into case studies in three applications, summarized in Table 1.1.

When developing usable ML, we must consider 1) the people involved, including their areas of expertise and the roles they play, and 2) the processes currently used in the decision problem. Table 1.2 summarizes the considerations we discuss for house pricing application. This table will be revisited for all case studies throughout this thesis.

## 1.2 Considerations for Usable ML: People

Developing usable ML first requires carefully understanding who will be using them, as well as their areas of expertise and the language they use.

There are five key types of people that need to be considered when developing usable ML. In this chapter, we introduce these roles, as well as the specific individuals we interacted with while completing the work in the thesis that held these roles across diverse applications.

1. **Decision-makers** use ML tools, including ML outputs and ML augmentations, in order to make decisions.
2. **Bridges** are people who bridge the gap between ML developers and decision-makers.

Table 1.2: Considerations for usable ML, as applied to housing pricing application.

Processes	
<b>Decision problem</b>	Should the homebuyer bid on this house, and if so, how much?
<b>Action space</b>	Whether or not to place an offer, what price to offer
<b>Assessed outcome</b>	Real estate agent commission, homebuyer ratings of real estate agent
<b>Time to make a decision</b>	Unbounded
<b>Associated risk</b>	Medium
<b>ML Integration</b>	Proxy (Predicted house value)
People	
<b>Decision-maker(s)</b>	Real estate agent and prospective homebuyer
- <i>ML/data expertise</i>	None
- <i>Domain expertise</i>	Some/Expert
<b>Bridges</b>	Product manager at real-estate ML modeling company
<b>ML Developers</b>	ML developer at real-estate ML modeling company
<b>ML Contributors</b>	Developers of usable ML systems (authors)
<b>Affected Individuals</b>	Homeseller, other homebuyers

They combine their experience in ML with their understanding of the domain at hand to help integrate ML into new decision problems.

3. **ML Developers** are people who prepare data and ML models for use in real-world applications — in other words, they are experts in applied ML.
4. **ML Contributors** are researchers, research engineers, and other experts who specialize in fields such as ML, AI, explainable AI, and human-computer interaction, and develop and contribute to broader systems for developing usable ML. The author of this thesis would fall under this role.
5. **Affected Individuals** include anyone else who is affected by decisions made using ML. Because Type 2 decision problems often have long-reaching impacts, the affected group may be quite large.

To better illustrate how these roles manifest and interact, consider the housing pricing application described above. In this scenario, the **decision-makers** include both the real estate agent and the prospective homebuyer, who together use the ML model to decide whether to bid and for how much. Neither decision-maker in this case is expected to have any ML expertise, and while the real estate agent will have domain expertise, the homebuyer may not. The **bridge** is the customer-facing product manager at the company that developed the real estate ML tool. The **ML developers** are the people at the real estate modeling company who trained the ML model and developed the ML tool. The **ML contributors** are researchers like the author of this thesis who worked on general ML systems that could be used to develop ML tools for this application. The **affected individuals** include the homesellers and other homebuyers on the market, who may be affected by the eventual bid.

We now go through these roles in more detail, including the specific individuals we have interacted with — summarized in Table 1.3

### 1.2.1 Decision-makers

Decision-makers incorporate ML outputs alongside their own expertise, experience, and external information to make decisions and complete their work. They tend to be interested in what the ML model and its outputs can tell them about the world and the decision problem at hand. They want information about the ML model to be curated to include only what is relevant for their decision-making.

Generally decision-makers do not have any data science or ML expertise, and have much expertise in the domain of the decision problem. However, this is not guaranteed, and in fact decision-makers could be experts in ML (such as in a decision problem about the ML model itself), or could have no specific domain expertise (such as when ML outputs are used by the general populace). Therefore, putting extra consideration to the expertise of decision-makers is essential.

Decision-makers are a very diverse group, and therefore require most nuanced and thorough consideration. Because of the challenges involved in fully understanding complex applications, we enlist help from bridges, discussed in the next section, to enabled smoother collaborations. Here, we discuss the examples decision-makers involved in the work done in this thesis.

**Child welfare screeners** make decisions about whether or not to investigate a case of potential child abuse that has been reported to a child protective services (CPS) hotline as a potential victim of child abuse or neglect.

The child welfare screeners' decision problem that we focused on is to decide whether to screen in a child for further investigation. If they decide to screen-in, CPS will conduct additional interviews and investigations to gather more information, before deciding whether further intervention is required. If they screen-out, no investigation will be conducted unless the child is referred again in the future.

The child welfare screeners we worked with did not have any formal experience with ML or data science, but did have extensive expertise in the application domain of child welfare. Their primary interest in the ML outputs was as an extra check to ensure they were not missing or misinterpreting the large amounts of data that they need to process in a short timeframe. For example, a hotline narrative that does not on its own illicit much concern may be more notable if associated with a child who has been referred many times by many different people — a ML tool with properly selected ML explanations can flag information like this.

**Child welfare supervisors** are also involved in the child welfare screening process as decision-makers, but with different goals and requirements. Child welfare supervisors oversee the screening process, ensuring that decisions are made effectively, correctly, and ethically. While they are still interested in understanding individual ML outputs when partaking in screening sessions, they are also interested in understanding broader patterns in screening decisions — and relatedly, the broader logic used by the ML model. Therefore, they are interested in ML augmentations like feature importance and summaries ML outputs on past cases. This group includes experts in child welfare, with even more years of experience than child welfare screeners; however, like the screeners, they are not expected to have any existing ML or data science expertise.

We interfaced with both child welfare groups in person and virtually, through a collabo-

ration with a county in Colorado.

**Wind turbine engineers** keep wind turbines running effectively and seek to minimize downtime and maximize energy output. Their decision problem is to decide how to maintain wind turbines, including when to send contractors up the wind turbines. Sending contractors up turbines to inspect and fix them in person is an expensive and potentially dangerous task, so it is essential to make these decisions carefully and appropriately. Like the child welfare decision-makers, this group has expertise in the application domain (wind turbine engineering), though not necessarily ML or data science expertise.

Unlike other decision-maker groups, this group interacts with ML outputs less directly, instead getting this information through the filter of the **wind turbine analysts**, who fill both the decision-maker and bridge role. The ML outputs the wind turbine engineers look at have been selected and aggregated by these wind turbine analysts, who use their combined turbine engineering and ML expertise to help parse out the most relevant information.

We interfaced with the wind turbine groups virtually through a collaboration with the electric utility company Iberdrola.

Our **real estate practitioner** decision-maker group consists of a broad range of individuals with professional experience in the real estate industry (such as agents, mortgage brokers, or appraisers), found through Prolific<sup>2</sup>. All of these individuals are experienced with the real estate industry, but their data science and ML expertise varied greatly from none to a lot. In this thesis, we did not specifically consider real estate practitioners who had real-world experience working with ML outputs, though ML models are becoming increasingly popular in the industry. These ML models include house price prediction/appraisal models, house recommender systems, and predictive investment systems.

We also interfaced with the **general populace**, or people selected from a random pool with no specific filters expertise, again through Prolific. As consumer-level ML tools become increasingly common, a larger number of people will interact with ML for their day-to-day decision problems. Therefore, it is worth considering the needs and expectations of a decision-maker audience of the general populace.

## 1.2.2 Bridges

The bridge role is becoming increasingly common as ML tools are developed for a wider range of applications. These individuals combine experience with ML with understanding of the needs of specific decision problems to help develop, tune, and deploy ML tools. They connect ML ML developers to decision-makers to enable smooth collaborations.

A classic variety of bridge is the **product manager**, a role that has existed for a long time in software development. These individuals traditionally bridge the gap between software developers and software users — analogous in our terminology to ML developers and decision-makers. They understand the needs and restrictions of both sides[1], and therefore perfectly fit the description of a bridge in applications where the product is a ML tool.

For our child welfare screening application, **social scientists** with the Centre for Social Data Analytics at Auckland University of Technology filled the role of bridge, and performed tasks not unlike those done by product managers. These scientists combine understanding of

---

<sup>2</sup><https://www.prolific.com/>

ML with experience in social welfare domains like child welfare screening to train ML models and develop ML tools. They validate ML models, and ensure they meet the practical, ethical, and legal requirements of child welfare screening. Our interactions with this group were in-person and virtually through a research collaboration.

For our wind turbine application, **wind turbine analysts** fill the bridge role. This team interfaces between turbine engineers and ML developers to develop, tune, and evaluate ML models and ML tools for wind turbine monitoring. Unlike in other domains, the data analysts in this case also partially fill the role of decision-maker, as discussed above, as they additionally help parse, filter, and aggregate ML outputs for the turbine engineers. This group has expertise in ML and data science, as well as an understanding of the wind turbine domain.

### 1.2.3 ML Developers

ML developers prepare data for an train ML models and develop ML tools for use in various applications. They have ML and data science expertise. While ML developers often aim to gain understanding of specific application domains to help with their work, they tend to not be experts, and interface with bridges to aid with their understanding. ML developers may work within specific applications as **applied ML developers**. Alternatively, they may work for companies on more general ML tools.

We worked with many ML developers ranging from those with less than a year of ML or Python development experience (recruited through Prolific) to those with 10+ years of specialized ML experience (recruited through platforms including Upwork and MIT email lists).

### 1.2.4 ML Contributors

ML Contributors are ML experts who create and contribute to generalized systems for developing usable ML. This group includes a wide range of **ML theorists**, **researchers**, and **research engineers** in areas such as ML, AI, explainable AI, and human-computer interaction. The author fills this role.

## 1.3 Considerations for Usable ML: Processes

Having identified and understood the people in an application, the next step is to understand the broader decision process including the decisions that can be made, the actions that can be taken, and the outcomes to be tracked. The Type 2 decision process is summarized in Figure 1.2, including the incorporation of an ML tool. The key questions that elucidate this process are:

1. What is the **decision problem**, or the decision that needs to be made?
2. What is the **action space**, or the possible actions the decision-maker can take based on the decision they make?

Table 1.3: Roles and types of people involved in the research in this thesis, and their corresponding expertise.

Role	Position	Application	Domain expertise	ML expertise
<b>Decision-makers</b>	Child welfare screeners	Child welfare screening	High	Low
	Child welfare supervisors	Child welfare	high screening	Low
	Turbine engineers	Wind turbine monitoring	High	Med
	Turbine data analysts	Wind turbine monitoring	Med-High	Med-High
	Real estate practitioners	House pricing	High	Varies
	General populace	House pricing	Low	Low
<b>Bridges</b>	Product managers	All	Med-High	Med-High
	Social scientists	Child welfare screening	Med-High	Med-High
	Data analysts	Wind turbine monitoring	Med-High	Med-High
<b>ML Developers</b>	Applied ML engineers	All	Low	High
<b>ML Contributors</b>	Authors	All	Low	High
	ML research community	All	Low	High

3. What is the **assessed outcome**, or the metric(s) to be optimized by the decision problem? In Type 2 decision problems, the real outcome of interest is often complex and difficult to quantify, so a related question is, what reasonable, measurable proxies exist for the outcome?

In addition to these three components of the process, a few more considerations are important:

1. How much **time** do the decision-makers have to make a decision? Does the decision have to be made in a matter of seconds, minutes, or hours? Or is time not relevant?
2. What is the **risk** associated with the decision — in particular with individual mistakes? Most decision problems will have negative consequences from repeated bad performance, but costs of individual errors vary greatly from low (such as a briefly irritated customer) to very high (such as death).
3. How are **ML outputs integrated** into the decision workflow? Will the ML model run completely autonomously, autonomously with human oversight, make direct recommendations of action that a decision-maker either follows or ignores, or give proxy outputs do not directly recommend action but are more broadly relevant to the problem? The last option is most likely for Type 2 problems.
4. What are the details of the **decision process** — what information do decision-makers look at, and how do they integrate that information in their work?

For the house pricing application, the decision being made (question 1) is how much (if anything) the homebuyer should bid on the house. The actions that can be taken (question 2) based on this decision are to bid a specific amount, or to not bid at all. The outcomes to be assessed (question 3) include homebuyer satisfaction, which could be proxied by metrics such as homebuyer feedback ratings, and real estate agent commission. The time available to make the decision will depend on the current housing market, but will usually be at least

several days. Risk is subjective; we label it here as medium, because while there are not significant risks to health or safety, a home is a very significant purchase. Finally, in this context the ML model offers proxy information — the predicted sale price of the house — which correlates with but does not directly correspond to the bid price.

## 1.4 Technical Challenges in Usable ML

These considerations for usable ML help guide the development of *ML systems* that can enable the creation of ML tools for diverse domains. They also highlight several technical challenges that this thesis seeks to address.

As we have seen, many people with many different types of expertise are involved in the process of developing and using ML tools. Decision-makers, who often have no ML or data science knowledge at all, need to be able to understand and use ML outputs. Bridges, who are familiar with ML but likely not experts at ML development, need to actively contribute to the development, tuning, and evaluation of ML tools. ML developers need to be able to create ML tools that include diverse and usable ML augmentations such as ML explanations without having to become experts in explainable AI research. Even affected individuals not directly involved in the decision problem may find themselves needing to understand how ML outputs have impacted them — for instance, as part of the right to explanation promised by the European Union’s General Data Protection Regulation (GDPR) [2].

Additionally, the diversity of Type 2 decision problems means ML tools will need to look very different for different applications. In some applications, decision-makers need to understand the problem and make a decision in minutes. In some, a tiny oversight could have catastrophic results. In some, the ML output itself is far less important than the corresponding ML explanations. Currently, ML tools need to be custom built and configured for every situation. Not only does this take tremendous time and resources, it also forces ML developers to be involved extensively and takes power away from bridges and decision-makers.

This is where a systems approach in computer science offers an approach — through the development of a highly configurable, generalizable system for usable ML, including a consistent and easy-to-use high-level API, usable ML can become faster and easier to apply to diverse domains.

In particular, this system needs to address three key challenges:

1. ML augmentations such as explanations, dataset visualizations, and performance metrics need to be understandable to the decision-makers who need to use them. Currently, many explainable ML algorithms create explanations that use features in the format — or *feature space* — required by the model. These explanations provide information in terms of heavily engineered features and are therefore difficult to understand.
2. The system needs to support easily creating ML tools that support the full range of interactions that decision-makers need, and be highly configurable to the diverse requirements of different applications. Creating new ML tools should not take months for each new application.



3. These ML tools need to be evaluated within the bounds of long-term, real-world projects. Lab studies cannot capture the intricacies of real-world applications.

In this thesis, we introduce a system for developing ML tools that are usable to diverse decision-makers. We also apply this system to multiple real-world domains, and evaluate their usability.

## 1.5 Problem Statement and Contributions

This thesis offers several primary contributions, which seek to address the practical and technical challenges involved in developing usable ML for Type 2 decision problems.

1. We introduce (above), and apply to multiple real-world applications, a formal set of considerations that act as a road map for developing usable ML tools.
2. We propose a formal taxonomy for interpretable features, motivated by work in multiple real-world applications (Chapter 4). We then develop a system called Pyreal for creating ML augmentations using interpretable features (Chapter 5).
3. We formalize the scope of interactions and configurations that must be considered when developing usable ML for diverse decision problems, and discuss our system called Sibyl that supports developing usable ML for this full scope (Chapter 6).
4. We evaluate our system and the ML tools developed with it through a longstanding project within the real-world application of wind turbine monitoring (Chapter 7).

We begin by giving some background on applied and explainable ML. We then formally introduce the many kinds of people we collaborated with in the work for this thesis. We then go through a motivating application in child welfare screening. This application demonstrates the technical challenges involved in developing usable ML tools. Then, we go through the technical work of the thesis, including our formal taxonomy for interpretable features, our system for creating ML augmentations with interpretable features, and our system for developing usable ML. Finally, we go through our evaluation in wind turbine monitoring.



# Chapter 2

## Background

In this chapter, we go through existing work on applied and explainable ML.

### 2.1 Human-Centered ML

In the past, some ML researchers have advocated for the field to adopt a *human-centered* perspective [3] — one that considers ML models acting alongside humans as part of collaborative teams. This perspective considers how humans use, interact with, adapt to, and evaluate ML tools and ML outputs [3]. A truly human-centered ML approach is comprehensive end-to-end, beginning with human-in-the-loop training and ending with evaluations based on the metrics decision-makers are most interested in [3]. This thesis focuses on the decision-making part of this pipeline.

Nyre-Yu et. al. [4] found that just explaining ML outputs is not always helpful; other kinds of augmentations, such as details about historic cases and data visualizations, may be preferable. Additionally, Cheng et. al. [5] found that interactive explanations work better for real applications than static ones. Throughout this paper, we prefer to use the term “usability” rather than “explainability,” to highlight that we consider many different kinds of augmentations, rather than only explanations.

Riedl [6] suggests that taking steps to create ML tools that help humans understand ML models better is an important part of human-centered ML research. A significant benefit of this work is that it allows humans to understand and move forward in the event that an ML output is incorrect [6]. In particular, Riedl emphasizes the importance of considering the increasingly large pool of decision-makers without ML expertise using ML tools.

Another important goal of human-centered ML is to lower the barrier to entry for building ML tools [7]. People without specific ML expertise offer many valuable insights about the decision process that are essential to leverage. Our system for developing usable ML tools, Sibyl, takes steps to bridge this gap and bring these people in.

Much literature has emphasized the importance of considering diverse audiences of decision-makers when developing ML tools for use in decision making, and suggested different taxonomies of roles to consider. Preece et. al. [8] proposed four main categories of individuals. In their proposal, developers, like our ML developers, build ML applications. Theorists seek to understand and contribute to ML theory — we tie this role in with ML contributors. Ethi-

cists are concerned with the fairness and accountability of ML systems — we have found that people across many roles take on this consideration, and do not consider it to be a specific role. Finally, users, like our decision-makers, use ML tools and ML outputs.

Barredo-Arrieta et. al. [9] emphasize the importance of considering and defining the target audience for explainable ML applications. They discuss applications where the audience includes domain experts, managers, and regulatory entities (all of which fall under our decision-maker umbrella), data scientists (which can be ML developers or ML contributors), product owners (bridges), and affected individuals. Each of these groups needs different things from explainable ML. For example, domain experts using ML models need to calibrate trust in the model, understand causality, and gain the additional information they need to make decisions, while regulatory entities may be more interested in the fairness and privacy of explainable ML.

Ray Hong et. al. [10] similarly investigated how people with different types of expertise and experience interact with explainable ML differently. They consider three categories of people. Model builders develop ML applications — this position is encompassed by our ML developer and ML contributor roles. Model breakers have the domain knowledge necessary to verify that ML models work effectively and meet application goals. Our bridge role includes this task, as well as other related tasks. Model consumers, like our decision-makers, use ML outputs to make decisions.

Ehsan et. al. [11] experimented with explainable ML by showing people with different levels of ML expertise robots that explained their actions in different ways. They determined that people, especially those with ML experience, tend to trust numbers to potentially unwarranted degrees, including valuing “explanations” that consisted only of unintelligible numbers. They further emphasized the importance of considering the expertise of the people who will use ML outputs and DSSs when developing them.

## 2.2 Explainable ML

The field of explainable ML (also called explainable AI or XAI) seeks to “open the black box” of ML models to help explain ML outputs to decision-makers. Many usability concerns in ML — such as lack of trust, difficulty reconciling disagreements between decision-makers and ML outputs, and difficulty connecting ML outputs to the broader real-world picture — stem at least in part from the black-box nature of ML models [12].

Doshi-Velez and Kim proposed that the need for explainable ML stems from an “incompleteness in the problem formulation” [13], which prevents the ML model from being thoroughly evaluated or trusted. This incompleteness prevents rigorous reasoning about the problem, which requires a more nuanced investigation by humans, which in turns requires explanation [13].

ML explanations can take many forms; broadly, they seek to provide some logic or justification for why a model acts the way it does, or deeper insight into how an ML model may act in the future. ML explanations can be *global*, meaning they explain the ML model’s logic in general, or *local*, meaning they explain a specific ML output [13]. For example, a global explanation may take the form of “the ML model values houses based on their size”, while a local explanation takes the form of “this house will sell for more than average because it is

larger than average”.

The explainable ML literature has proposed a wide range of algorithms for explaining ML models. These algorithms can be *model-agnostic*, meaning they work on any type of ML model that follows some basic structure (such as taking in tabular data and returning an output prediction) or *model-specific*, meaning they are developed to work with specific properties of one type of ML model architecture [13]. Some ML models are furthermore considered to be *inherently interpretable* (also called white-box models) [13], meaning they require few enough operations and have simple enough structures that their logic can be directly understood without external explanation algorithms. For example, models like linear/logistic regression, decision trees, and decision rules are considered to be white-box models. However, my past work [12] has found that even white-box models can be difficult for decision-makers without ML expertise to fully understand.

Due to the wide variety of ML models architectures used throughout real-world applications, the systems discussed in this thesis focus primarily on model-agnostic explainable ML algorithms. Examples of these include the following:

1. Global Surrogate Modeling [14] is an approach where one aims to train a white-box ML model that roughly approximates the logic or decision boundary of the black-box ML model being explained. In this approach, a white-box model is trained to predict the black-box model’s outputs.
2. Local Interpretable Model-Agnostic Explanations (LIME) [15] is a local explainable ML algorithm built around the concept of local surrogates. Unlike global surrogates, local surrogates seek to match the black-box model only for a small set of inputs. Rather than training a surrogate model on a full training dataset, in LIME the surrogate is trained only on a dataset of input/output pairs that are in the local region of the input from the input/output pair of interest.
3. SHapley Additive exPlanations (SHAP) [16] is a local explainable ML algorithm that assigns a positive or negative contribution to every feature used by the ML model, indicating how much that feature increased or decreased the ML output on the specific input. The A in SHAP stands for *additive*, meaning the contributions of each feature together will add up to the final ML output. SHAP’s feature contributions are instances of Shapely values, a concept from game theory, that state how to fairly distribute a payout (the ML output) across all players (features).
4. Permutation Feature Importance [17] produces global explanations through permuting features individually in the training dataset, and observing how much the model performance drops based on this permutation.
5. Counterfactuals [18] seek to find the smallest change in an input required to change the ML output to a different class (for classification models), or change the output by some threshold (for regression models). For example, a counterfactual explanation may look like “your loan would not have been rejected if your salary was \$10,000 higher”.
6. Anchors [19] can be thought of as the opposite of counterfactual explanations — they seek to find the minimum set of features such that, if they are unchanged, the ML

output will also not change, even if all other features are changed by some threshold amount.

Many examples throughout this paper use SHAP-based explanations, as this is a popular, intuitive approach.

## 2.3 Explainable ML for Decision Making

Developing explainable ML for use in decision-making is a highly context-dependent task. Explainable ML tools tend to be developed for an audience with ML expertise. Past work has suggested it is beneficial to include decision-makers more directly in the explainable ML development process. Hong et. al. [10] found through interviews that ML explanations developed with decision-makers involved were better attuned to the domain-specific needs of the decision problem, and suggested that these explanations can be misused if they are made without considering the target audience (for example, doctors could misinterpret explanations as having causal meaning if they're presented incorrectly). Bhatt et. al. [20] found that most ML tools featuring explanations are intended for use by ML developers rather than by decision-makers, leading to usability challenges. Jiang and Senge [21] found that decision-makers without ML expertise have different goals for explainable ML than ML developers, such as a need to vet the model per their own legal and ethical requirements, and for explanations be accurate but understandable given their expertise. The authors found that collaborating with these teams is essential for generating clear explanations. Arrieta et. al. [9] found that different groups use explainable ML for very different purposes, ranging from model debugging for ML developers to knowledge discovery for decision-makers.

Past attempts to use explainable ML tools for real-world decision problems has demonstrated the importance of tuning explanations to the intricacies of each domain, while also demonstrating the difficulties involved in doing so.

Tonekaboni et. al. [22] surveyed clinicians in an intensive care unit and an emergency department to characterise what kind of explanations are effective for this group of decision-makers. Participants were asked to imagine a hypothetical tool that would predict cardiac arrest. Key factors that clinicians reported wanting to see included 1) aid in justifying their own medical decisions, 2) details about the ML model's context, including situations in which it is likely to make errors, and 3) an understanding of the data the ML model uses to calibrate trust.

Lundberg et. al. developed PRESCIENCE, a ML tool focused on preventing hypoxaemia during surgeries [23]. This system predicts the risk of hypoxaemia in the next five minutes using a gradient-boosting algorithm trained on time series data. It also includes several visualizations to explain its predictions, including SHAP feature contribution explanations.

Kwon et. al. [24] developed RETAINVIS, a ML tool for explaining predictions made by recurrent neural networks (RNNs) applied to electronic medical records (EMRs). The system was developed with active feedback from decision-makers (medical practioners). RETAINVIS includes five different visualizations for looking into RNNs: 1) an overview of patient information including multiple plot types, 2) a summary of patient data over time, 3) an interface that enables comparisons across different patients, 4) details about individual pa-

tients, and 5) a what-if analysis system that allows users to experiment with hypothetical cases by editing patient values.

## 2.4 ML Systems for Building ML Tools

Several systems and frameworks for creating ML tools for diverse decision problems already exist.

Wang et. al. [25] developed a human-driven conceptual framework for developing explainable ML tools. They found that decision-makers seek explanations to justify unexpected occurrences, monitor for important events, or facilitate learning. They created a taxonomy of AI techniques based on how they support human reasoning and represent information.

Many existing systems have ML developers as the target audience, for use in ML model debugging and tuning. What-If Tool [26] is a system that creates interactive ML tools focused on counterfactuals and probing model behavior designed for debugging models. It enables ML developers to better understand how ML models perform on different subsets of input data and under varying hypothetical parameters. InterpretML [27] is an explanation system for debugging and auditing ML models. It offers a unified, extensible API that enables users to generate explanations in a standardized way regardless of model type. InterpretML is meant for audiences of both ML developers and other individuals interested in the model, though its primary focus is on aiding understanding of the ML model rather than helping with active decision-making. Manifold [28] is an ML explanation system specifically made for debugging models and improving performance.

Other systems create ML tools for an audience of decision-makers without ML expertise. H2O Driverless AI [29] is an autoML system that includes an explanation component. This system includes several explanation methods, including decision tree surrogates, K-LIME, and multiple feature importance metrics. Our work focuses more closely on the usability and explainability component of the usable ML workflow, aiming to increase application-specific configurability.

AI Explainability 360 [30] and AI Fairness 360 [31] are systems for building decision-maker-focused systems that offer explainability and fairness metrics, respectively, for applied ML. Both offer extensible and flexible Python-based APIs that bring together different methods for providing additional information about ML models and their outputs, with the goal of making explainability and fairness metrics accessible to a wider audience. Sibyl extends this work by offering additional kinds of augmentations on top of explainability, as well as additional layers that enable this information to be accessed without Python code.





## Chapter 3

# Application 1: Usable ML for Child Welfare Screening

The work in this thesis was motivated by a year-long investigation into developing usable ML tools for child welfare screening. Prior work on ML tools tended to focus on applications used by ML developers [28], [32], [33] or decision-makers in more technical or data-driven fields such as medicine [23], [24]. We aimed to expand the scope of ML ML tools by undertaking a thorough investigation into the challenges involved when child welfare screeners, decision-makers who lack ML or data science expertise, need to use and understand ML outputs.

ML outputs have been deployed in Allegheny County, PA and elsewhere, with the goal of enabling more efficient and consistent decision-making and improving the overall health and safety of children and families [34]. For our case study, we worked with an ML model introduced to our collaborating county in Colorado by Vaithianathan et. al. [34]. This was a LASSO regression ML model trained on 461 features, including information such as the child and parents' ages, past referrals and their outcomes, and past court involvements. The model predicts the likelihood a child will be removed from their home in the next two years if they are investigated, translated to a 1 through 20 risk score, where higher scores were shown to correlate with higher risk of abuse [34]. The purpose of this model was to help child welfare screeners decide which cases to screen in for further investigation, by highlighting situations where the data may paint a different picture than their intuition.

Due to the high-stakes nature of this application, and the decision-makers' lack of ML expertise, we expected to find some usability challenges with ML outputs. Through our previous literature review [35], we identified seven key challenges in using ML outputs for decision-making described in the literature, codified in Table 3.1. We investigated which, if any, of these applied to child welfare screening.

The work in this chapter seeks to address four research questions:

- RQ1** What are the answers to the considerations described in Chapter 1 for the field of child welfare screening? Who are the people and what are the processes involved?
- RQ2** What challenges do child welfare screeners encounter when trying to use ML outputs for decision-making?
- RQ3** Past work has relied on simple visualizations of the ML output — what additional

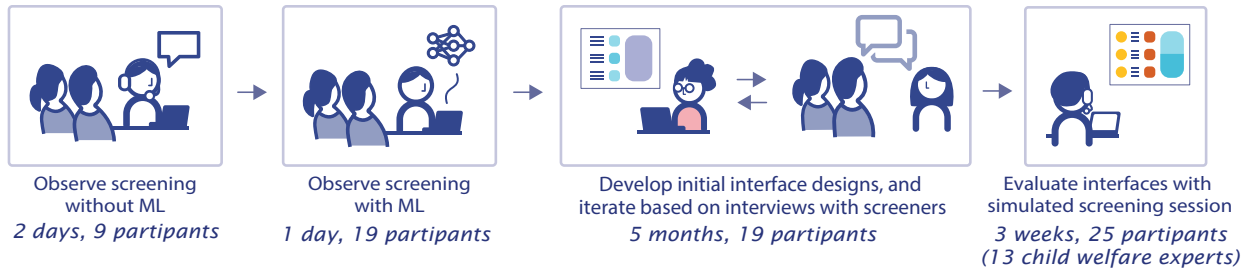


Figure 3.1: The process we followed for the child welfare screening case study.

ML augmentations such as explanations can be helpful in mitigating these usability challenges?

**RQ4** What design choices must be made when building ML tools with these augmentations to optimize them for use by child welfare screeners and other similar decision-makers?

The work in this chapter was previously described in [12] and further elaborated on in [36]. In this thesis, we highlight methods and findings from this work relevant to the need for systems for usable ML that help motivate the rest of the thesis.

**Case Study Participants.** Over the course of a year, we collaborated with a pool of 19 child welfare screeners and child welfare supervisors working for the child protective services (CPS) department in a collaborating county in Colorado. This group made up our participant pool for the feedback and development portion of the case study. All participants regularly participate in the county’s child welfare screening decision-making process. Our collaborations began in December 2019, with two days of in-person field observations. Following this, we observed a simulated case review session via video conferencing and conducted several interviews, also via video conferencing. Finally, the child welfare screeners and supervisors participated in our user study digitally.

Our study involved four steps, as summarized in figure 3.1.

## 3.1 Understanding Context and Decision-maker Needs

To address our first two research questions — understanding the people and processes involved in child welfare screening, and the challenges that arise when child welfare screeners try to use ML outputs — we performed a series of field observations and interviews [12]. In this section, we discuss our goals and the findings we made during these steps.

### 3.1.1 Observing Decision-Making Without ML to Understanding the People and Processes

To better understand the people involved in child welfare screening and the existing child welfare screening process (**RQ1**), we travelled to our collaborating county in Colorado to observe how child welfare screeners make screening decisions *without* using ML outputs. Our observations led to the following findings:

Table 3.1: List of usability challenges that impact the usability of ML models.

Usability Challenge	Code	Mitigating Tools
Lack of <b>Trust</b>	<b>TR</b>	Global explanations, local explanations, performance metrics, historical predictions and results
Difficulty Reconciling human-ML <b>Disagreements</b>	<b>DIS</b>	Local explanations
Unclear <b>Consequences</b> of actions	<b>CON</b>	Cost-benefit analysis, historical predictions and results
Lack of <b>Accountability</b> or protections from accountability	<b>ACC</b>	Local explanations, performance metrics
<b>Ethical Concerns</b> (ex. possible bias, concerns about oversimplification)	<b>ETH</b>	Global explanations, local explanations, ML fairness metrics, historical predictions and results
<b>Confusing</b> or unclear prediction <b>Target</b> (ie. the measure predicted by the model has an unclear meaning or significance)	<b>CT</b>	Cost-benefit analysis, further analysis of prediction target impact
<b>Unhelpful</b> prediction <b>Target</b> (ie. the measure predicted by the model is not relevant to the required decision)	<b>UT</b>	Retrain model with new prediction target

1. The primary people involved include the child welfare screeners and their supervisors (decision-makers), as well as the social scientists developing the ML model (bridges).
2. Figure 3.2 shows the decision process used by our collaborating county. In cases of high concern, a referral may be screened in immediately after it is received by CPS (this decision is made by a child welfare supervisor). In most cases, however, the decision of whether to screen-in or screen-out a referral is made by a team of child welfare screeners. It is this team that receives the risk score computed by the ML model.
3. This team spends five to ten minutes on each case. Most of this time is spent going over the details of the case, followed by one to two minutes of discussion, after which the screening decision is made.
4. A large amount of this five to ten minutes of screening time is dedicated to determining factors in the case that are associated with higher and lower likelihoods of abuse — referred to as risk and protective factors, respectively — and weighing these against each other. The factors considered will vary based on the details of the case, but may include information such as child’s age (very young children are more vulnerable), criminal records of adults involved, whether there are any trusted adults active in the

Table 3.2: The usable ML considerations for child welfare screening, including people and processes.

Process	
<b>Decision problem</b>	Whether to investigate a potential case of child abuse
<b>Action space</b>	Investigate immediately or within a set number of days; provide additional services; log with no action
<b>Measured outcome</b>	Number of re-referrals with substantiated abuse; number of investigations without abuse
<b>Time to make a decision</b>	Minutes
<b>Associated risk</b>	High
<b>ML Integration</b>	Proxy (Risk score)
People	
<b>Decision-maker(s)</b>	Child welfare screeners and supervisors
- <i>ML/data expertise</i>	None
- <i>Domain expertise</i>	Expert
<b>Bridges</b>	Social scientists
<b>ML Developers</b>	Social scientists, Authors
<b>ML Contributors</b>	Authors
<b>Affected Individuals</b>	Referred children and families; communities of referred families

child’s life, and actions and statements made by adults involved (for example, a mother taking action to separate an aggressive partner from her children).

Through our interviews, we identified the answers to the application considerations shown in Table 3.2.

### 3.1.2 Observing Screening with ML to Understand ML Usability Challenges

Next, we sought to identify the usability challenges faced by decision-makers when using ML outputs. To identify the particular ML usability challenges relevant to child welfare screening (**RQ2**), we observed a simulated case review session where decision-makers used ML outputs. In this session, our 19 participating child welfare screeners and supervisors were split into three teams, which we will refer to as **T1**, **T2**, and **T3**. They were asked to make screening decisions about real referrals that the county had handled in the past. There was no filtering done on these cases, so some cases may have been ones the screeners reviewed in the past. For each case, they had access to the same information that child welfare screeners typically have during screenings — including the age of all involved parties, and a written description of the potential abuse as given by the referring party — as well as a 1 through 20 risk score outputted by the ML model. After receiving this information, the participants went ahead with their usual decision process: discussing the case as a team for five to ten minutes and then making a screening decision. The teams were then interviewed and asked to reflect upon how they used each ML output, whether the outputs aligned with their expectations, and how they impacted their screening decisions. Each team made decisions on seven to nine cases.

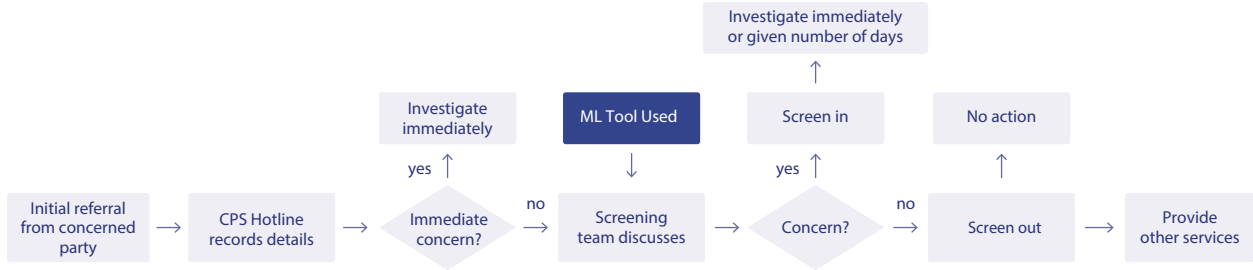


Figure 3.2: The child welfare screening process used by the CPS department of our collaborating county [12]. First, the CPS hotline receives a referral, which is then sent to a child welfare supervisor. In a minority of cases, the supervisor will deem the child or children involved to be at immediate risk, and will screen-in the case for immediate further investigation. In most cases, however, the case will be reviewed by a team of child welfare screeners the next day. This team will be given the ML output (risk score, dark blue box). If this team decides to screen-in the case, it will be investigated further through home visits, interviews, or other means. Otherwise, the case will be recorded, but not investigated unless re-referred. In the case of a screen-out, the screeners may elect to provide the family with other family services, such as access to counseling or food benefits.

During interview sessions, all three teams expressed reservations about using the ML outputs. Based on their responses to our interview questions, we identified four key usability challenges:

1. **Lack of Trust** **TR** Participants expressed a lack of trust when making screening decisions using the ML outputs, as evidenced by their tendency to not consider the outputs at all when they disagreed with their intuition. For example, when asked whether an output score caused them to reconsider one of their decisions, **T3** responded

*“No, [we were] surprised it is that low.”*

2. **Difficulty reconciling human-ML disagreements** **DIS** Participants did not have a clear path forward when they disagreed with the ML outputs, sometimes electing to ignore it entirely (see previous item) and sometimes trying to justify it based on how they thought the ML model worked. For example, **T2** reported in one case that the score made them

*“think a little deeper about why the score is so high [and caused us to] take another look at [the history]”*

3. **Unclear prediction target** **CT** Because the ML model provides proxy information (a 1-20 risk score based on the likelihood of removal from home within 2 years) rather than a direct decision suggestion, there was some confusion about how to use the ML outputs. For example, when asked how the model affected their decision process, **T3** responded

*“[we did not know] enough of what the score means to know how to accurately use it.”*

**T3** also said they

*“wish we knew how we got to the score.”*

4. **Concerns about Ethics** **ETH** As expected for such a sensitive domain, participants were concerned about the ethics of using the ML outputs. There was concern that the model could prevent critical thinking. **T3** commented

*“[The model] could be dangerous for people just looking at the number, need to take everything into account. Makes you stop and think and ask yourself are you critically thinking...”*

## 3.2 Developing and Iterating on ML Tool Interface Design

To address our second two research questions – regarding which kinds of ML augmentations help address usability challenges and what design decisions need to be made when developing them – we created a prototype ML tools including five interfaces [12]. We began by designing high-fidelity mock-ups for each of the interfaces. Table 3.3 summarizes the motivation behind each interface in terms of its theorized effect on addressing the usability challenges we identified.

Early in the design process, we learned that the word “factor” is more familiar to child welfare screeners and supervisors than “feature” when referring to pieces of information used when making decisions. For the purposes of consistency, we use the word "factor" throughout this chapter when referring to data inputs used by the ML model.

We then iterated on our interfaces based on feedback from screeners and supervisors, as described in detail in [12]. The final version of one of our interfaces is shown and described in Figure 3.3.

## 3.3 Evaluating ML Tool Interfaces with Simulated Screening Session

Having gone through and addressed initial feedback, we implemented an experimental ML tool based on our mock-up designs. We then evaluated this system with two formal user studies, in the form of simulated screening sessions.

Our first study had 12 participants, including a mix of ML developers and social scientists. We chose to run a study with this group first in order to iterate on the basic UI/UX elements of the DSS without taking too much of the child welfare screeners’ limited time. Although our ML developer participants had no prior experience in child welfare screening, it was possible for them to understand the process intuitively enough to use the ML tool for rudimentary decision-making. We refer to this group as the domain *non-expert participants*, as they are not experts in child welfare screening.

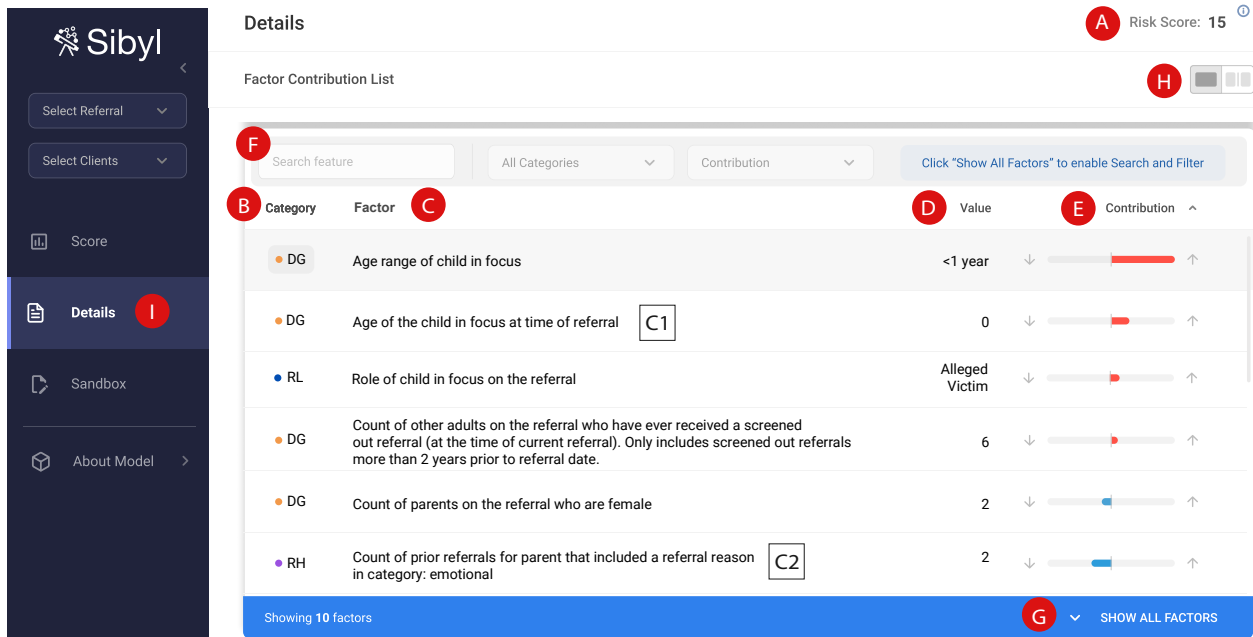


Figure 3.3: The final version of the "Details" interface after factoring in feedback from our final user study evaluation. This interface shows how particular factors contribute to the child welfare screening ML outputs. Labeled elements are as follows: (A) The ML output (risk score) for the case (1-20). (B) Categories for each factor, such as demographics (DG) or referral history. (C) A short description of each factor. (D) The value of numeric or categorical factors. (E) The contribution of each factor (the table can be sorted in ascending or descending order of contribution). (F) UI components for searching by factor name or filtering by category, enabled when the full factor list is shown. (G) A button for switching between a view that shows only the top 10 most contributing factors and one that shows all factors. (H) A button for switching between a single-table view and a side-by-side view, which splits factors that increase and decrease risk. (I) A sidebar for switching between different interfaces.

For our second study, we engaged 13 of the collaborating child welfare screeners from our case study participant pool, 2 of whom completed the task while video conferencing and screen-sharing with us. We refer to this group at the domain *expert participants*, as they are experts in child welfare screening. We discuss the results of both user studies in this section. **Data used:** For privacy reasons, we used only synthetically generated and deidentified data for this study. Data for different factors was generated using the CTGAN synthetic data generation algorithm [37], in which a generative model is learned from the real data and samples that resemble the real data are drawn from this model.

Case descriptions were real paragraph-form narratives provided by concerned parties during past referrals. Names were changed by a representative of the county for de-identification. For example:

*“Caller (teacher) says Abby (age 5) came into school with bruise on arm. Caller says Abby often comes in bruised. Abby told teacher she fell off bike. Teacher asked Abby’s mother about this...”*

Participants were first shown a short video explaining how to use our ML tool. Next,

Table 3.3: The proposed ML tool interfaces (left column), the challenges they were theorized to address (middle column, using the codes from Table 3.1), and the reasons we expected these interfaces to address the given challenges (right column). **TR**: Lack of trust in the model. **DIS**: Difficulty reconciling disagreements. **CT**: Confusing prediction target. **ETH**: Ethical concerns.

Interface	Challenges Addressed	How does it address the challenge?
	<b>TR</b>	
Case-Specific Details	<b>TR</b> <b>DIS</b> <b>CT</b> <b>ETH</b>	Reveals relevancy of considered factors Highlights factors that may have been missed or mis-used Translates score to a concrete factor list Allows for critical thinking about factors and score
Sandbox	<b>DIS</b> <b>ETH</b>	Allows users to test theorized justifications Allows for thinking through what-if scenarios
Similar Cases	<b>TR</b> <b>ETH</b>	Provides information on past performance Provides a deeper look into the nuance of cases
Global Factor Importance	<b>TR</b> <b>CT</b>	Reveals how the model generally makes predictions Translates the score to a concrete factor list
Factor Distributions	<b>TR</b> <b>CT</b>	Shows how well the model performed on past cases Shows the relationship between the risk score and removals from home

they were shown 7 case descriptions, accompanied by ML outputs (risk scores) and the ML tool populated with simulated data.

Participants were then asked to make a screen-in/screen-out decision, and to answer some reflection questions. These questions included 1) five-point Likert-scale style questions about how much participants trusted the model and how confident they felt in their decisions, 2) multiple-choice questions about which interfaces in the ML tool were helpful, and 3) free response questions about trust in the model and general feedback.

In total, the child welfare expert participants completed 73 individual case analyses, and the non-experts completed 75.

### 3.3.1 Study Results

#### The Details interface was rated the most helpful by screeners

To address **RQ2**, we analyzed the self-reported helpfulness of each interface in the ML tool.

The Details interface was considered the most helpful interface by a large margin, by both experts and non-experts. It was labelled as being helpful by experts in 91.8% of case analyses, and by non-experts in 90.7% of case analyses. This was significantly higher than Sandbox (experts: 16.4%, non-experts: 22.6%) and Factor Distributions (experts: 20.5%, non-experts: 8.0%). Factor Importance was never listed as helpful by either group.



## Participants with child welfare screening expertise rely on the ML tool less than those without this expertise

Unsurprisingly, domain non-experts were more likely to report listening to the ML output without considering the additional augmentations provided by our ML tool. One non-expert participant commented,

*“No idea what is going on in this case description – so completely defer to the model here.”*

Another non-expert participant commented

*“I found the score useful - and used it as a justification for screening out without exploring in detail all the factors.”*

Additionally, non-experts reported that they used the model “a lot” or “a great deal” on 46.4% of cases, while experts chose these options in 15.6% of cases.

## The ML Tool can both increase and decrease trust for different reasons

The ML tool was reported to both increase and decrease domain expert participants’ trust in the model, for different reasons. To analyse why this might occur, we thematically analyzed the responses to the open question “What made you trust the model more or less?” We gathered 56 responses to this question on individual cases and divided the answers into two categories, based on the corresponding answer to the 5-point Likert scale question “How much did you trust the model’s prediction for this case?” 43 free-responses corresponded to trusting the model “a great deal,” “a lot,” or “a moderate amount,” while 13 corresponded to trusting the ML output “a little” or “not at all.” For all but three of these answers, the response corresponded with the degree of trust listed (ie., participants who reported trusting the ML output provided reasons why they trusted it *more*). We do not include the three exceptions in our analysis.

To conduct the thematic analysis, we categorized responses according to which (if any) specific interfaces were referenced, and further identified what specific elements of the interfaces were referenced. For answers that did not reference a specific interface, we identified themes in the responses, such as agreement with the ML output or references to general information provided by the ML tool. With these two categorization mechanisms, we sorted the responses into 7 themes that corresponded with increasing trust, and 5 themes that corresponded with decreasing trust.

Table 3.4 lists these themes. We see that agreeing with the ML output increases trust in the model the most. Beyond this, the Details interface was frequently cited as increasing trust, either due to specific factors listed or more general elements of the interface, such as the number of factors. Trust was reduced when there was confusion or inconsistency in the presented information, or when the interfaces suggested that the ML model did not consider important factors that participants knew about.

## The way information is presented in ML tools can cause confusion

To address **RQ3**, we categorize and summarize the comments made by users regarding the ML tool design choices, as well as the steps we took to address them.

1. **Too many factors shown.** The ML model was originally trained on over 400 factors, all of which were presented in the ML tool, but many of these factors have zero or near-zero weight. For example, one expert participant commented:

*“Too many factors listed. I only want to see the material risk and protective factors.”*

Our updated version of the ML tool only shows 10 factors by default, with an option to show more.

2. **Confusion caused by correlated factors.** The ML model uses some engineered factors, resulting in factors that have deterministic relationships. For example, there is a numeric factor called `AGE OF CHILD`, and then a set of binary factors referring to each age group: i.e. `CHILD IS LESS THAN 1 YEAR OLD`, `CHILD IS BETWEEN THE AGES OF 1 AND 3`, etc. These factors may cause confusion when shown directly to participants. In addition to increasing the cognitive load without providing additional useful information, interfaces that display these factors may reveal seemingly contradictory or unusual relationships. For example, the categorical age factor may contribute greatly, while the numeric age factor does not.

Additionally, having these correlated factors causes confusion on the sandbox page, as it is possible to change one factor without changing all of the other deterministically-correlated factors in its set. One non-expert participant commented,

*“I’m not sure, in the sandbox, if I change one feature, other features will be changed automatically.”*

To solve these problems, we combined the correlated factors in the information we displayed, forming categorical factors out of binary one-hot encoded factors, and summing the additive contributions.

3. **Confusion caused by Boolean terminology.** One source of confusion was the method of displaying Boolean factors. In our original design, we displayed the description of the factor, with a value of `True` or `False`. This is the most accurate way of representing the model’s logic, but it is not the most intuitive way for the child welfare screeners. One expert participant said,

*“The ‘true’ and ‘false’ is hard to interpret... Would rather have a positive statement (e.g., no perpetrator named)”*

Therefore, our final version of the ML tool instead states only true statements about the child — including by negating descriptions of false factors. For example, the factor `CHILD HAS SIBLINGS` with a value of `False` will be displayed as `CHILD DOES NOT HAVE SIBLINGS`.

Table 3.4: Summary of answers to the question “what made you trust the model more/less” by child welfare expert participants. The top section lists reasons for having “a great deal”, “a lot” or “a moderate amount” of trust in the model. The bottom section lists reasons for having “a little” or “not at all” trust in the model. The first column lists the general themes we found in the answers. The second column lists the number of answers that fell within each theme. The final column lists selected answers for each theme.

Themes that increased trust		
Category	# of answers	Sample answers
The screener liked seeing what factors were considered risk or protective (Details)	8	<i>“The details and the risk and protective factors and the contribution they have” “Info in the details and risk factors”</i>
The screener agreed with a specific factor being listed as risk or protective (Details)	8	<i>“The past number of child welfare involvements (listed in the features listed)”, “The risk factors involved, especially prior placements, benefits, and current CPS involvement”</i>
The number of risk or protective factors aligned with the score (Details)	5	<i>“Very few risk factors”, “The lack of protective factors”</i>
The score agreed with screener intuition	10	<i>“...not residing with the alleged perpetrators which I would assume would reduce the risk score”, “Model prediction makes sense”</i>
The explanation agreed with screener intuition	3	<i>“Risk factors made sense for the model prediction number”</i>
The screener liked seeing how the score would vary under different conditions (Sandbox)	1	<i>“Details under sandbox of why the risk level was so high”</i>
The screener felt getting an explanation allowed for more understanding in general	6	<i>“Allowed for more understanding”, “History clarification”</i>
Themes that decreased trust		
Category	# of answers	Sample answers
A specific, key factor was not considered by the model	3	<i>“This is a case for law enforcement, not CPS”, “...it may have been handled during the open case”</i>
The importance weighting of factors was off (Details page)	2	<i>“Young, vulnerable children being left alone is still cause for concern, despite past involvement”</i>
The score disagreed with screener intuition	3	<i>“Seems high, no health history, no real proof of any drug use, no proof child is at any risk of abuse”</i>
The way information was presented was confusing	2	<i>“There were discrepancies between the info in the referral and the info provided by the tool”</i>
The screener wanted more information	2	<i>“I would want to see other referrals for the family”</i>

## 3.4 Discussion

Here, we describe our key takeaways for usable ML in general from this case study.

### 3.4.1 Accurate explanations may be more useful than faithful ones

Robnik-Sikonja and Bohanec [38] define the *accuracy* of an ML explanation as how well it generalizes to other unseen examples (i.e., how accurately these rules predict what happens in the real world), and *fidelity* as how well an explanation describes the ML model itself.

The decision-makers in our case study (child welfare screeners and supervisors) were mostly interested in getting *accurate* explanations that provided information about *the case at hand*. As evidenced by all three findings about the interface design (Section 3.3.1), screeners wanted to receive information about the ML model in a language and format that mirrored their own, not the format used by the model itself. This is also evidenced by design requests like using the terms *risk and protective factors*, rather than the more ML-centric terms *negative and positive features*.

### 3.4.2 Interpretable features are essential for understanding

Simple ML model architectures, such as regression, are often cited as being inherently interpretable [39] — in other words, humans are able to understand how they work. However, this work suggested that even simple ML models may cause confusion in decision-maker groups such as child welfare screeners, and might lead to challenges when attempting to explain ML outputs to these groups.

Instead, our work found that, for the purposes of making ML outputs usable for decision-makers, the interpretability of the factors used (in ML terms, features) may be most important. In our study, the child welfare screeners were often confused when explanations used features that did not have clear implications for risk. For example, in our user study, one child welfare screener said

“... 2 parents have missing date-of-birth is shown as a significant blue bar which I can't imagine is protective.”

Additionally, one-hot-encoded features were not interpretable, and many of the reasons screeners trusted the model more or less (Table 3.4) related to the specific features.

At the time of this study, the phrase *interpretable features* still lacked a formal and thorough definition. In our case study, all the features we were working with were interpretable by the strictest definition — the screeners could understand exactly which real-world concepts they represented, and their associations with these concepts. This level of interpretability was not sufficient to prevent confusion.

We determined from this work that the ML literature would benefit from a more precise definition of *interpretable features*; one that factors in the context and audience. In the next chapter, we discuss our proposed taxonomy for interpretable features to address this issue.

Table 3.5: Key findings about specific challenges to ML usability that we seek to address in the rest of this thesis. The chapter column gives the chapter in this thesis where we aim to address each challenge.

Challenge	Example	Chapter
Features shown to decision-makers must be interpretable	Screeners were confused by one-hot encoded features and features that seemed irrelevant	4, 5
Terms used in ML tools must match the language decision-makers are used to	Screeners preferred the term "protective/risk factors" to "positive/negative features"	6
Not all ML augmentations are appropriate for all applications	Screeners felt that the similar cases page would lead to biased decision making.	6
ML tools should be appropriate for the existing decision process.	Screeners wanted a side-by-side view of protective/risk factors to mimic what they used normally, and did not want to see more information than they could parse in their limited timeframe	6, 8
Evaluations within real or realistic settings reveal challenges that are not identified through interviews alone	Only the simulated screening session revealed certain aspects of how screeners interacted with factors, such as being confused by uninterpretable features.	7

### 3.4.3 The case study highlights several challenges related to systems for usable ML

Table 3.5 summarizes the key findings related to usable ML revealed by this work. The rest of the work in this thesis was motivated by these findings. Chapter 4 discusses our in-depth analysis into the need for interpretable features. Chapter 5 discusses our system for generating explanations that use such interpretable features. Chapter 6 discusses our system for more easily adapting ML tools for diverse applications, without requiring customization. Chapter 7 discusses another investigation into the real-world use of ML tools. Finally, Chapter 8 seeks to further adapt ML tools to existing workflows through generating natural language sentences.



# Chapter 4

## Interpretable Features

Our experience developing and evaluating ML tools for child welfare screening demonstrated the importance of interpretable features to usable ML. We investigated several other real-world applications [40]–[43] that confirmed that ML models are only as interpretable as their features. ML explanations and other augmentations are generally presented using the language of the features the model uses; whether through presenting feature importances, meaningful example inputs, decision boundary visualizations, or other explanation methods. We conducted a literature review (Section 4.3) that confirmed that the literature has acknowledged the importance of interpretable features (IFs) and suggested a wide variety of methods to generate them, but lacks a nuanced and context-aware analysis of what it means for a feature to be “interpretable” for a given purpose. In particular, we find that an “interpretable feature” is often defined simply as one that was human-generated or worded using human-readable language. However, our extensive experience in five diverse applications have suggested that this definition is incomplete. Indeed, for features to be useful as part of ML explanations and other augmentations, they may require a variety of properties.

In this chapter, we aim to better define the concept of feature interpretability, focusing on applications that use tabular data or featurized time series data.

The lack of formal investigation into what makes features interpretable is partly because literature tends to focus only on a subset of people who use ML tools — mostly ML developers and ML contributors, who have expertise in ML and data science and are interested in training, improving, and validating models. For these groups, the goal of using these tools is to engineer (or discover, such as through deep learning techniques) features that maximize model performance. In reality, though, many other people also use these tools, or may in the future.

Bridges, decision-makers, and affected individuals are especially likely to need explanations that use features with a deeper level of interpretability. In this chapter, we discuss our experiences working with these groups, and the feature interpretability properties that they find necessary.

Figure 4.1 summarizes the contributions of this chapter, and highlights differences between the needs of users focused on the model (ML developers, ML contributors, and bridges) versus users focused on the decision problem (bridges, decision-makers, and affected individuals).

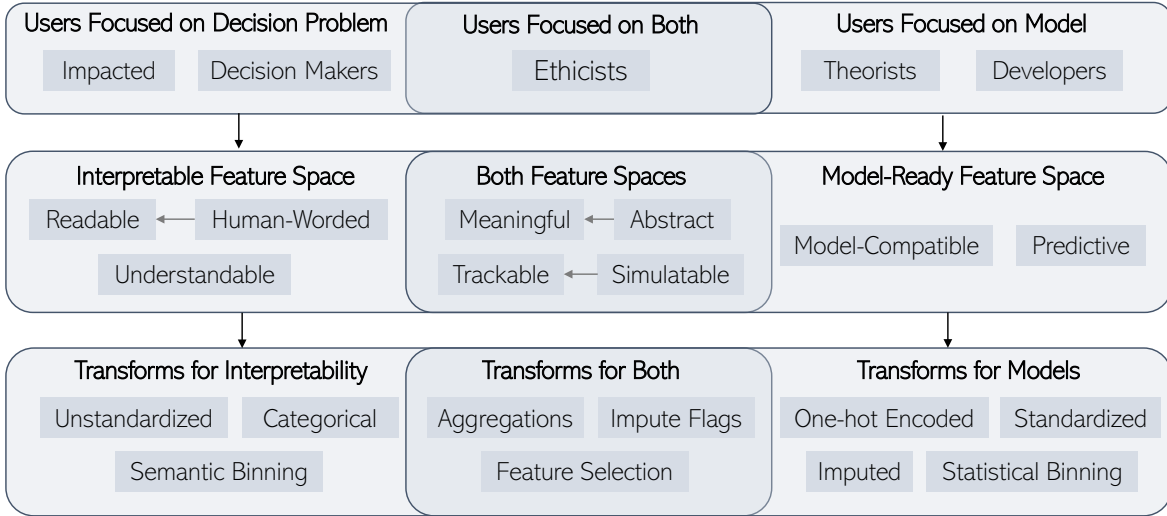


Figure 4.1: Summary of our feature taxonomy. This table shows the different users (top layer), the feature properties required for their decision processes (middle layer), and the transforms that yield the required feature properties (bottom layer). The lines between properties indicate that, for a feature to have the property at the head-end of the arrow, it must also have the property at the tail-end of the arrow. An example of a path through this figure: Decision-makers usually require the interpretable feature space, which is characterized by features that are **understandable**, **human-worded**, and **readable**. To achieve these characteristics, model explanations should rely on transform states such as unstandardized, categorical, and semantic binning, and should avoid model-ready transformations such as unflagged imputations or one-hot encoding

## 4.1 Case Studies in ML Applications

In addition to child welfare screening, we investigated a number of other real-world applications for ML tools in order to better understand the importance and challenges of interpretable features. We will introduce them below. For each application, we worked with decision-makers (who did not have ML experience) who were using ML outputs and ML tools to help with their decision-making. In all cases, we found that a major roadblock to the usefulness of ML outputs stemmed from confusion related to the features. We will provide examples of these roadblocks throughout the paper as we introduce our feature property taxonomy. These applications are summarized in Table 4.1.

### 4.1.1 Education: Abstracting Features

In the domain of online education, we worked on feature engineering and ML modeling tasks using data related to massive open online classes (MOOCs) [43]. This research applies to a diverse set of people, including ML developers working on MOOC analysis systems, course developers looking to learn about and improve online course quality (decision-makers), and



Table 4.1: Summary of example applications referenced in developing our taxonomy for interpretable features.

Domain	Stakeholder Type	Description
Child Welfare Screening [12]	<b>Decision-makers:</b> Social workers screening cases <b>Bridges:</b> Social scientists reviewing the model	Collaborated with child welfare screeners to develop an ML tool. The tool provides detailed insights about feature contributions and distributions.
Education [43]	<b>Decision-makers:</b> Instructors conducting courses Instructors designing courses <b>ML Developers:</b> Developers working on analysis systems	Ran a series of different experiments involving feature engineering and analyzing outcomes from MOOCs
Cybersecurity [40]	<b>Decision-makers:</b> Domain experts monitoring for attacks	Developed, evaluated, and deployed machine learning for cybersecurity. Extended the detection capability of security operations centers with machine learning models trained to identify attacks in log data.
Electronic Health Records [41]	<b>Decision-makers:</b> Doctors conducting surgery	Collaborated with doctors to develop an ML tool to explain predictions on EHR data. The tool displays feature contributions, and shows the original raw signal data that features are generated from.
Satellite Monitoring [42]	<b>Decision-makers:</b> Domain experts monitoring signals	Collaborated with satellite experts to develop an ML tool to visualize ML-predicted anomalies in satellite signal data.

instructors who want to better understand their students (decision-makers). Goals for ML in this application include classifying student engagement level and type, and identifying course resource usage. The features used include data such as problem set grades, number of hours of course video watched, or number of discussion posts written.

We worked through the process of engineering MOOC features to better inform users about students and courses. Our experiences, and the literature on this topic, have shown that the most useful features take the form of abstract concepts that have meaning to the decision-makers (such as summarizing features relating to completing work and interacting with course materials into a feature called **participation**). We also found it important to ensure that decision-makers fully understand the source of these concepts.

### 4.1.2 Cybersecurity: Tracking Features

Our next application is in the domain of cybersecurity — specifically, the detection of Domain Generation Algorithms (DGAs) [40], a well-documented command and control mechanism [44]. However, the lessons learned are applicable to a wide range of threat detection

use cases. DGAs are designed by attackers to generate dynamic domains that are then used as meeting points for compromised machines and remote attackers. The key idea is that the remote attacker and the compromised machine share the generation logic, and will therefore generate the same set of domains, each on its own side. The remote attacker then registers one of the many domains generated by the algorithm, while the malware in the compromised machines attempts to connect to each and every one of the generated domains until a communication channel is established with the remote attacker [40]. In this case, our decision-makers were security analysts, and the decision problem was how to respond to a potential attack. The ML model was trained to detect and flag potential DGA attacks based on aggregated information (features) about the activity of internal IPs captured in DNS logs (such as the number of failed DNS requests, or the randomness of queried domains).

We found that the raw data — in this case, the relevant sections of DNS logs — was more useful to the decision-makers than the generated features, as the logs held more familiar and actionable information. For example, the security analysts need to investigate the specific domains queried by a suspected IP to determine the presence or absence of a DGA. In this case, the challenge for ML explainability was how to select these relevant logs, in order to accurately track feature values back to the raw data they were generated from.

### 4.1.3 Healthcare: Disaggregating Features

In our healthcare application, we built ML models using Electronic Health Record (EHR) data to improve the quality of clinical care [41]. In particular, we worked closely with six clinicians (decision-makers) with an average of 17.5 years of medical work experience. Their decision problem was predicting complications after cardiac surgery. The clinicians were interested in using ML models to predict the chance of a patient developing five types of complications (lung, cardiac, arrhythmia, infectious, and others) after surgery. To this end, we extracted various types of ML features from the medical records of target patients, mainly including three types of static features (demographics, surgery information, and diagnosis results) and three types of time-varying features (lab tests, routine vital signs, and surgery vital signs). We then built five individual binary classifiers, with each predicting one of the five complication types.

We adopted SHAP [16] to calculate the feature contributions. As in the cybersecurity application, we found that features engineered through aggregations on the raw data are not as interpretable as the original raw signals. Offering a way for clinicians to see and get explanations on the raw signals was required.

### 4.1.4 Satellite Monitoring: Marking Real Features

In the domain of satellite monitoring, we developed a ML tool to visualize the results of time-series anomaly detection models [42]. While this application used only time-series data, rather than tabular data, we learned lessons about tracking imputations that extend to the tabular use case. The ML tool allowed domain experts to analyze and discuss the relevance and validity of anomalies predicted by the ML model. We actively collaborated with members of our intended decision-maker group — six satellite monitors. The ML model we employed for the ML tool makes use of time-series sensor data to detect anomalous periods.

We developed the ML tool with active feedback from our decision-makers. We then ran two user studies to evaluate the ML tool, first with the six satellite monitors, and then with a group of 25 members of the general public, using stock price data. This process revealed feature-relevant considerations – such as the need to clarify which feature values were the result of an imputation process, and therefore not real values.

## 4.2 Key Lessons

From our experiences in the domains described above, we have found three key lessons:

**Lesson 1: We Need an Interpretable Feature Space.** More focus is needed on what we refer to as the *interpretable feature space*. The term *feature space* refers to data in a specific format. The interpretable feature space, then, is this data transformed to a human-understandable format that enables usable ML. Much attention in the literature has been given to collecting, selecting, and engineering features [45], [46], usually with the goal of maximizing model test-set and real-world performance. The result is a heavy focus on the transformations that convert from the *original feature space* to the *model-ready feature space*. However, when an ML model needs to interface with decision-makers who lack ML expertise, the interpretable feature space may be needed. This feature space is engineered to directly represent concepts related to the decision problem and human cognition, rather than ML and statistical concepts. More work is needed on how to define, generate, and evaluate this feature space; this chapter focuses on defining the space.

**Lesson 2: Features Require Certain Properties in Order to Be Interpretable.** What properties are required for features to be interpretable heavily depends on audience and application. As suggested in Lesson 1, if the goal is only to maximize ML model performance, important properties may include that features are in a model-compatible format and are statistically correlated with the target variable. However, if the goal is provide useful explanations to decision-makers, features that understandably relate to real-world application and abstract concepts may be better. We propose a formal taxonomy of these feature properties in Section 4.4.

**Lesson 3: We Need to Transform the Other Way.** Feature properties are achieved through a combination of feature selection and engineering. Transforms that prepare data to be input to an ML model may increase or decrease feature interpretability; in the latter case, these transforms (which we dub *model-ready transforms*) may need to be undone when used in ML explanations and other augmentations. In addition, transforms that are not useful for getting outputs from the ML model, but do improve feature interpretability, can make ML tools more usable. However, many ML libraries do not include functionality for transforms that improve interpretability or undo model-ready transforms. Section 4.5 provides a partial list of such transforms.

### 4.3 Formal Literature Review

Table 4.2: Summarized results of the methodological literature review on interpretable features. Papers were counted towards a category if it was explicitly referred to in the text.

Comments on Feature Interpretability	Number of Papers	Sample Papers
<b>No Elaboration:</b> Mentions interpretable features without any elaboration or citations regarding meaning or purpose	47	---
<b>Generating IFs:</b> Proposes an ML architecture or feature engineering/selection approach that results in interpretable features	49	[47] [48]
<b>Importance of IFs:</b> Discusses or claims the general importance of interpretable features	47	[20] [49] [10]
<b>Defines IFs:</b> Provides some context-specific or general definition or metric for interpretable features.	30	[50] [51] [52]
<b>Interpreting Features:</b> Proposes an algorithm for interpreting the meaning of latent neural network features	23	[53] [54] [55]
<b>Relies on IFs:</b> States that the methods in the paper rely on interpretable features to be effective	18	[56] [57]
<b>Falls short of IFs:</b> Explicitly states that the methods in the paper did not achieve interpretable features	5	[58] [59] [60]
<b>Concerns about IFs:</b> Describes potential drawbacks of interpretable features	2	[61] [62]

To further understand existing perspectives on interpretable features in the literature, we conducted a methodological literature review. We reviewed papers from 19 top conferences and journals in the fields of machine learning, human-computer interaction, computer vision, and data visualization. Within each of these venues, we searched for papers that included (in title or text body) the keywords “machine learning” or “ML”, as well as the phrase “interpretable feature(s)” or any of its semantic equivalents (readable/understandable/explainable feature(s), interpretability of (the) feature(s)). We limited our search to papers that were published during or after 2015, as this can be considered the start of the newest wave of interest in explaining black-box ML [63]. In total, our literature review included 106 papers.<sup>1</sup>

For each paper, we noted the domain, the data type, and the goal of the paper, as well as the reason for mentioning interpretable features, which we then codified into 8 categories, seen in Table 4.2. We also took note of the papers that provided any kind of definition, either general or context-specific, for the concept of “interpretable features” (hereafter IFs).

<sup>1</sup>For comparison, we ran a larger search using the same procedure for all papers published in the same years and venues described above that mentioned “machine learning” or “ML” and the word “features” in any context. This search revealed a total of 13,104 papers that talked about features and ML.

Our findings from this literature review can be summarized as follows.

**Attention is given to the importance of IFs.** The need for IFs is well documented in the literature. 47 papers mentioned the importance of IFs, either in general or to the methods proposed in the paper. 49 papers claimed to generate interpretable features through their proposed methods, though only 18 offered any kind of concrete definition of an interpretable feature. Additionally, five papers explicitly stated that their algorithm or model generated uninterpretable features.

**Little work on IFs addresses tabular data.** Of the 106 papers in our literature review, 47 mentioned interpretable features only briefly, with no further elaboration as to how or why interpretable features were chosen or used. 80 focused on image, text, or time series data, and/or discussed interpretable features in the context of interpreting the latent features discovered by neural networks. 26 papers provided concrete insights on interpretable input features for tabular or arbitrary data types.

**There is a lack of concrete definitions for IFs.** Just 30 out of 106 papers offered at least a brief general or context-specific definition or quantification method for interpretable features. Of these, seven defined interpretable features as being those selected or engineered by humans, or similar to those selected by humans. Four defined interpretable features by contrasting them to the uninterpretable latent features used by neural networks or PCA. Four papers considered features to be more interpretable when they were related to abstract concepts. Only one paper [64] directly quantified the interpretability of features through a user study.

From our formal literature review, we conclude that despite widespread agreement that interpretable features are important for various reasons<sup>2</sup>, there is little work formalizing what makes a feature interpretable, and almost no work quantifying the interpretability of features. Most of the concrete work on interpretable features has focused on complex data domains such as image and text, and aims to add interpretability to deep neural networks. Based on our experiences in the five domains described in our introduction, we claim that a deeper, human-centered understanding of interpretable features for tabular features is necessary.

We have also found that the term *interpretable feature* does not have a clearly agreed-upon meaning, but usually refers to features that are human-generated or can be understood by humans at a surface level (i.e., humans can understand what a feature refers to). In this paper, we seek to go beyond the term *interpretable* to introduce a complete, descriptive taxonomy of feature properties.

## 4.4 Taxonomy of Machine Learning Features

In this section, we formalize and build on the current literature on ML features to develop a thorough and nuanced taxonomy of feature properties. Through this process, we aim to

---

<sup>2</sup>It is worth noting that two papers in our literature review did find that in contexts where privacy is important, uninterpretable features may be preferable - see the last row of Table 4.2.

Table 4.3: Summary of the properties including in our formal feature taxonomy. For the examples, we use the following hypothetical scenario: a regression model trained on normalized data to predict the maximum speed of the car. **Quality** is a composite feature computed based on other features, and **x12** is an arbitrary predictive engineered feature.

	Quality (numeric)	Horsepower (numeric)	Color (categorical)	Norm. Horsepower (numeric)	Color is blue (Boolean)	x12 (numeric)
Readable	✓	✓		✓	✓	
Human-Worded	✓	✓	✓			
Understandable	✓	✓	✓			
Meaningful	✓		✓	✓		
Abstract Concept	✓					
Predictive	✓	✓		✓		✓
Model-Compatible	✓	✓		✓		✓
Model-Ready	✓			✓		✓

develop guidelines that ML developers can reference when selecting, engineering, and presenting ML features for specific applications. We motivate our proposed taxonomy through experiences with the real-world applications discussed above, and our findings from our formal literature review. Table 4.3 summarizes our taxonomy, and provides examples from a hypothetical application of predicting the maximum speed of a car.

We do not claim this to be an exhaustive list of feature properties — we have selected a set based directly on lessons learned while working on the five described applications. For example, we do not include properties such as causal, actionable, inherent [65], or extrinsic features [65]. While such properties are essential to understand for developing ML augmentations and ML tools, they warrant a separate discussion.

Users across different applications will expect to see features with different feature properties. Of particular importance is whether the objective of these users is to understand the ML model itself or to use the model to better understand the context of the decision problem. Therefore, we split our discussion of properties into three sections, one for each of these user groups (model-focused and decision problem-focused) and one for properties that apply to both groups.

#### 4.4.1 Model-Focused Users: The Model-Ready Feature Space

In this section, we list properties that ML models require from features. These therefore are the properties that people interested in the ML model (ML developers, ML contributors, and some bridges) will expect to see in features presented in ML augmentations. These are the defining properties of what we call the *model-ready feature space*.

- **Predictive** features correlate with the prediction target, or ML output value. Features that are not predictive are not expected to improve a model’s performance. Feature selection and engineering methodologies aim to produce a set of predictive features.

Features may be predictive via a directly causal link — for example, referencing Figure 4.3, it is likely that horsepower is a predictive feature when predicting the maximum speed of a car. Predictiveness may also result from indirect correlations; for example, we assume car color is not predictive for maximum speed, but it could be if, for whatever reason, faster cars come in similar colors.

- **Model-Compatible** features are in formats that are supported by the model’s architecture. Put another way, model-compatible features can be fed into the model to get an output without causing errors. For example, many models, such as linear regression, cannot accept categorical features by default — these features must be one-hot-encoded or converted to numerical codes. Additionally, many models cannot handle missing data, so imputation is required.
- **Model-Ready** features are in the final format expected by the model, as used during model training and intended by model developers. This includes both compatibility and performance-based feature engineering. Model-ready features are always **model-compatible**. Put another way, model-ready features can be put into the model without error and will result in the most accurate prediction possible for the model. This category will include the required transforms for model-compatibility, as well as transforms such as standardization or normalization. Many of these transformations will reduce the interpretability of features while improving the performance of the model.

#### 4.4.2 Decision Problem-Focused Users: The Interpretable Feature Space

In this section, we list properties that users who are focused on the real-world decision problem (decision-makers, bridges, and affected individuals) expect to see in features. These are the defining properties of what we call the *interpretable feature space*.

- **Readable** features are written in language that allows users to understand what is being referred to, and do not use any codes except those that are readily understood by the users. Concretely, for a feature to be readable it must take from a common, known vocabulary. Across the literature, this is generally considered a bare minimum for interpretability. Examples of unreadable features include those using unfamiliar codes such as `x12`, unfamiliar acronyms such as `M_miss_DOB` instead of `Mother is missing date of birth`, or features such as those generated by PCA (unless they are accompanied by an explanation of which input features they came from). Readable features include natural-language words, phrases, or commonly understood codes (`Age`) and straightforward descriptions of feature engineering operations (`log(humidity)` [66]).
- **Human-worded** features are described in the way that is most natural for users. Human-worded features are always **readable**.

As described in the previous chapter, we found this property to be essential in the child welfare domain. For example, child welfare screeners were confused by the one-hot encoding language used by the model, so our visualizations displayed categorical features (`gender -> MALE` instead of `gender is male -> TRUE`). The one-hot encoded versions are **readable**, in that they do not use unclear codes, but are still not natural and take additional time to parse. Additionally, child welfare screeners preferred Boolean features presented as negative and positive statements (`the child has no siblings` rather than `the child has siblings -> FALSE`).

- **Understandable** features refer to real-world metrics that users can reason about. The extent to which a feature is understandable will depend on the context and users' expertise, but generally this category will not include engineered features that are the results of mathematical operations unless they are commonly referred to concepts. For example, `age` is generally understandable, but `log(humidity)` [66] is generally not.

### 4.4.3 Feature Properties: All Users

These feature properties are relevant to the ML model's logic while also potentially improving interpretability, and therefore may be useful to all types of users.

- **Meaningful** features are those that users have reason to believe actually correlate with or are related to the ML output. This property may be essential for usable, trustworthy ML that does not confuse users. While this property relates closely to **predictive**, some features may be predictive as a result of spurious correlations and therefore may not be meaningful to users. Conversely, some features may seem to users like they should correlate with the prediction target, but may not actually have any predictive power. ML models that use only meaningful features may generalize better [67], so this property may also be of interest to ML developers and other model-focused users.

As an example, in our case study in child welfare screening, one screener said

*"... 2 parents have missing date-of-birth is shown as [significantly decreasing risk] which I can't imagine is protective."*

In this case, the Boolean feature `parents missing date of birth` is understandable, but still causes confusion because it is unclear how and why this should correlate with risk.

- **Abstract Concepts** are features that may not be directly measurable, but are calculated through some combination of the original dataset features. These features take the form of abstract concepts that are relevant to users. While they can improve the interpretability of ML models, they may also improve performance. For example, when evaluating massive open online courses (MOOCs) in the domain of education, abstract concepts such as `participation` and `achievement` [68] are used, which in turn are hand-crafted from metrics such as hours spent watching videos and homework attempts (for the former), and test and homework grades (for the latter).



- **Trackable** features come with a clear data lineage — they can be associated accurately with the raw data they were calculated from. For example, in the EHR domain, we found that features like `mean(heart rate)` may be more useful if presented alongside the original time series heart rate data that the mean was calculated from [41]. Similarly, in the cybersecurity domain, information about potential DGA attacks was more useful when provided along with the relevant DNS logs [40].

Another form of trackability is knowing when a feature value is the result of imputation (for example, the value represents a mean or synthetic value, rather than an actual collected data point), and therefore does not link to any real data. This property was requested by the satellite monitors [42].

- **Simulatable** features are those where the user has enough information and knowledge to accurately recompute the feature from raw data — or *simulate* the feature computation process. **Simulatable** features are always **trackable**.

For example, in the MOOC domain, a feature described as `test grade over time` is **trackable** (as it is clearly computed from raw test grades) but **not simulatable**, as it is unclear what the exact formula used to calculate this information is. It could refer to the slope of the trend line of grades, or the difference between the most recent grade and the average grade.

## 4.5 Feature Transforms

Having introduced a set of feature properties valued by different user groups, we will now discuss how to transform data to create features that hold these properties.

Some properties must be considered during the feature selection phase. For example, **predictiveness** and **meaningfulness** are attributes of the feature meaning itself, and can only be provided by carefully selecting such features. **Understandability** will also come in part from careful feature selection.

However, many other properties can be instilled in features through feature engineering or providing additional information. **Readability**, **trackability**, and **simulatability** can often be instilled through hand-writing feature definitions or providing relevant visualizations. **Model-compatibility** and **model-readiness** are instilled through traditional feature engineering. **Human-wordedness** and to some extent **understandability** can also be instilled by feature engineering, though with a different set of transforms than those traditionally used.

In this section, we consider two categories of ML feature transforms. *Model-ready transforms*, or those that take features from the original feature space to the model-ready feature space, have received extensive coverage in the literature [45] [46]. In this section, we suggest a subset of possible *interpretable transforms*, or those that take features from the original feature space to the interpretable feature space.

Table 4.4 provides examples of these transforms from the forest covertype dataset [69]. This dataset includes 12 input features and one target variable. Each observation in this dataset refers to a 30 by 30 meter patch in Roosevelt National Forest, and includes information such as elevation, sunlight levels, and soil type. The intended output value is the type of forest cover in that area.

### 4.5.1 Model-ready transforms

Traditionally, most work on feature engineering has focused on the model-ready feature space. These transforms generally have the purpose of 1) converting the data to a form that is compatible with the model, such as one-hot encoding categorical data or imputing missing data; 2) improving performance by factoring in an understanding of the model’s methodology, such as normalizing features for models that are sensitive to scale or using dimensionality reduction to avoid overfitting; or 3) improving performance by factoring in an understanding of the application domain.

Model-ready transforms will often reduce the interpretability of features. For example, in the child welfare domain, we learned that one-hot encoding reduces the [human-wordedness](#) of features, and binning numeric features may reduce the [understandability](#) of features if the bins are not semantically meaningful [12]. Additionally, in our experience with satellite signal monitoring, we found that imputing data can reduce [trackability](#) if users do not know which data is real.

### 4.5.2 Interpretable transforms

Some model-ready transforms may naturally lead to interpretable features, especially those that factor in domain knowledge to improve model performance. Other times, feature interpretability is improved by either undoing model-ready transforms or introducing new transforms.

Here, we introduce transforms that may improve feature interpretability, along with examples from our own experiences with real-world applications.

- **Converting to Categorical.** One-hot encoded features are not [human-worded](#). For example, `gender is female -- TRUE` should be replaced with `gender -- FEMALE`. This transform becomes especially important as the cardinality of the categorical feature increases.
- **Semantic Binning.** Binning numeric features based on meaningful, real-world distinctions can improve both model interpretability and performance. For example, in the child welfare domain, users frequently referenced and relied on age categories, which were binned based on child development stages (infant, toddler, teenager, etc.).

This kind of binning is often more interpretable than binning based on statistical metrics (such as ensuring equal bin widths or sizes), as is often done to improve model performance. In the forest covertype example in Table 4.4, consider the difference between binning elevation based on ecologically significant elevation zones such as `foothills` vs `alpine`, compared to generating bins of equal widths, resulting in bins such as `2525m - 3192m`.

- **Flagged Imputations.** Explanations that display imputed (synthetic) data can confuse users. In our experience with satellite monitoring, the raw data signals included missing data that was then imputed. When our visualizations did not distinguish between real data and imputed data, users were confused.

- **Aggregated Numeric Features.** Numeric features may be more interpretable in an aggregated format, especially when the model takes several very closely related metrics. In the forest covertype example, we combine the horizontal and vertical distance metrics to get total distance to hydrology. In the child welfare example, the overload of data that screeners experienced could be remedied in part by summing together measures, such as physical, sexual, and emotional abuse referrals, into a total referral count metric.
- **Categorical Feature Granularity.** Categorical features can often be represented with varying levels of granularity, through hierarchical relationships with other features. Both model performance and model interpretability can be improved by selecting the right granularity for a given domain. In our forest covertype example, all soil types present in the dataset belong to exactly one of eight geologic soil zones. Therefore, we could either present the individual soil type or the soil zone, depending on which is more useful to users.
- **Combining into Abstract Concepts.** An extension of the previous two transforms: abstract concepts that have been carefully crafted from data using an understanding of the application domain can be much faster and easier to parse than simply including large numbers of individual features. This is generally done by combining a set of features with a hand-crafted formula. In our forest covertype example, an abstract, human-intuitive feature may be `light level`, computed by aggregating the three `hillshade` features that quantify the illumination of a point at different times of day.
- **Reversing Scaling.** Standardized or normalized features are almost never interpretable. Features should be displayed in an unstandardized format in explanations, unless the relative position of a value is meaningful.
- **Reversing Feature Engineering.** Some transforms that have been done to improve model performance may need to be undone to improve interpretability. For example, a feature represented by `sqrt(value)` may need to be squared to get `value`.
- **Connecting to Raw Data.** Sometimes, raw data (for example, the original signal data) is more meaningful to users than engineered features. For example, in the EHR domain, the patient pulse signal itself is often more useful than the feature `MEAN(pulse)`. Therefore, explanations may be more useful if they display the signal itself, linked to the model features. This transform is an extension of reversing feature engineering, by explicitly displaying how the engineered feature has been calculated from raw data.

## 4.6 Discussion

### 4.6.1 Using feature properties

We will now revisit a few of the applications introduced in Section 4.1, to give concrete examples of how the proposed taxonomy in this paper could be used to improve the usability of ML tools.

In the child welfare screening application, we identified — through interviews and field observations — that the decision-makers were child welfare screeners who were not expected to have any prior understanding of data science or ML. Their primary interest was in the screening decision problem, and they wanted explanations of ML outputs to aid with this task. They trust their existing decision process, and mainly want a ML tool that highlights pertinent information they may have missed. For these reasons, child welfare screening requires features that are **meaningful**, **understandable**, and **human-worded**. We can achieve these properties in part through feature selection and semantic binning, while ensuring features we present are categorical and unstandardized.

In the electronic health record application, we identified that our decision-makers were doctors looking to improve outcomes during surgery. They were used to making decisions with full access to patient data – in particular time series for metrics such as vital signs. Therefore, features needed to be **simulatable**, which is provided by connecting the features to data and including impute flags.

Similar to these two examples, we encourage ML developers developing ML tools for real-world applications to collaborate with decision-makers and/or bridges to understand which feature properties will be necessary for usability.

### 4.6.2 Risks

While interpretable features may be necessary to make a ML tool useful, there are always risks involved in transforming features to the interpretable feature space. Providing explanations and other augmentations using anything other than the model-ready feature space risks lowering the fidelity of the explanations. Some of the transformations also have the potential to greatly bias the explanations – a fact that could be used maliciously. For instance, selecting formulas to generate abstract concepts and binning numerical data can hide certain patterns that are being used by the model. Consider the example of an ML model that outputs crime recidivism predictions and includes information about race, among other factors. A developer with bad intentions could maliciously tie the race feature into a broader concept called **socioeconomic factors** that hides the fact that race is being used by the model in an unfair way.

Additionally, forcing features to be more interpretable may reduce model performance. In particular, limiting a feature set to only those features seen as meaningful to humans will prevent the model from discovering potentially unknown or unexpected patterns in the data. Some of these patterns may relate to real-world correlations or causal paths that would have been very valuable to increasing knowledge in that domain. It is essential for ML developers to understand the trade-offs between different feature spaces, and make context-aware decisions when selecting features.

Finally, as noted by two papers from our literature review [61] [62], using interpretable features may reduce data privacy in cases where the features describe sensitive information. In such cases, it may be necessary to avoid interpretable properties such as readability in the interest of preserving privacy.

### 4.6.3 Shortcomings from existing systems

As we found in our literature review, although many sources discuss the importance of interpretable features, developing ML tools that use interpretable features requires significant effort. The primary challenge is that many types of powerful ML models require some model-ready transformations in order to perform well. Many ML model architectures cannot take in categorical data, or are not robust to features that exist on different numeric scales.

On top of this, as we discuss in further detail in the next chapter, many algorithms for generating ML explanations and other augmentations present data in the format used by the model. While many libraries that implement data transformers, such as `sklearn`, include functionality for "undoing" transformations on data, similar systems do not exist for transforming these other augmentations. Therefore, there is currently a performance-interpretability trade-off that must be made when training ML models. In the next chapter, we introduce our approach to creating explanations and other augmentations in a different feature space than the one expected by the model, allowing for both high-performing models and interpretable and usable ML tools.

Table 4.4: Example of feature transforms using the Forest Cover dataset

Feature Transform	Feature Name	Feature Description	Feature Type	Sample Values			
Original	Elevation	Elevation in meters	Numeric	3179	3123	2157	
	Horizontal Distance To Hydrology	Horizontal distance to nearest surface water features	Numeric	450	218	85	
	Vertical Distance To Hydrology	Vertical distance to nearest surface water features	Numeric	56	21	10	
	Hillshade 9am	Hillshade index at 9am, summer solstice	Numeric	156	228	210	
	Hillshade Noon	Hillshade index at noon, summer solstice	Numeric	231	229	131	
	Hillshade 3pm	Hillshade index at 3pm, summer solstice	Numeric	211	187	103	
	Wilderness area	Wilderness area designation	Categorical	Comanche Peak	Comanche Peak	Rawah	
	<b>Model-Ready Transforms</b>						
	Statistical Binning	Elevation Range	Uniform-width bins for Elevation	Categorical	Medium (2525m-3192m)	Medium (2525m-3192m)	Low (1859m-2525m)
		One-Hot Encoding	Area Rawah	Wilderness area is Rawah	Boolean	FALSE	FALSE
Area Neota			Wilderness area is Neota	Boolean	FALSE	FALSE	FALSE
Standardization	Area Comanche Peak	Wilderness area is Comanche Peak	Boolean	TRUE	TRUE	FALSE	
	Area Cache la Poudre	Wilderness area is Cache la Poudre	Boolean	FALSE	FALSE	FALSE	
	Elevation Standardized	Standardized Elevation	Numeric	0.784	0.584	-2.87	
PCA	PCA 1	Feature 1 from PCA	Numeric	-1.965	2.278	-0.851	
	PCA 2	Feature 2 from PCA	Numeric	2.265	1.547	3.095	
<b>Interpretable Transforms</b>							
Semantic Binning	Elevation Zone	Colorado Life Zone corresponding to elevation	Categorical	Subalpine	Subalpine	Foothills	
Flagged Imputed	Elevation Flag	Flags if elevation value was imputed	Boolean	FALSE	FALSE	FALSE	
Aggregated Numerical	Distance from Hydrology	Total distance from hydrology, computed from horizontal and vertical distances	Numeric	453	219	85	
Categorical Granularity	Soil Geologic Zone	Geologic zone of soil type (8 total)	Categorical	igneous and metamorphic	glacial	igneous and metamorphic	
Abstract Concepts	Light Level	Overall light level (Sum of hillshade features)	Categorical	High	High	Medium	

## Chapter 5

# Pyreal: A System for Interpretable Explanations

As concluded in the previous chapter, current systems for explaining ML models often result in explanations that are difficult to understand. When ML models use uninterpretable features, most explainable ML algorithms will produce explanations that use those same uninterpretable features.

In some cases, retraining an ML model to use only interpretable features may be straightforward and even improve performance; however, this is not always possible or practical. Sometimes using interpretable features will reduce model performance significantly, or the model cannot be retrained for external reasons such as regulatory requirements or lack of resources.

It may also be possible to feed an entire ML pipeline (including both transformers and the ML model) into an explainable ML algorithm, resulting in an explanation that uses the input features to the pipeline — but many explainable ML algorithms either do not allow this, or experience increased runtime or decreased accuracy as a result. For example, KernelSHAP [16] explains a full pipeline including transformers rather than just a model, resulting in explanations that use untransformed features, but this algorithm can be computationally expensive and is not able to take advantage of properties of the model architecture itself to improve runtime and accuracy.

We therefore aim to create a system that can convert existing ML explanations that use uninterpretable features to equivalents that use interpretable features — what we call *interpretable explanations*. Fig. 5.1 shows an example. We posit, and demonstrate through user studies, that interpretable explanations are preferred and more easily understood by many users than explanations presented using uninterpretable features.

Creating interpretable explanations requires understanding and working with multiple feature spaces. As introduced in the previous chapter, a feature space is defined as a set of features in specific formats that serves a specific purpose. For example, one feature space may have all categorical features one-hot encoded and all numeric features standardized, and will therefore be supported by a model that expects numeric inputs with constant mean and variance.

In this chapter, we introduce Pyreal, a system for generating interpretable explanations through manipulating feature spaces. Pyreal is built on three key principles:

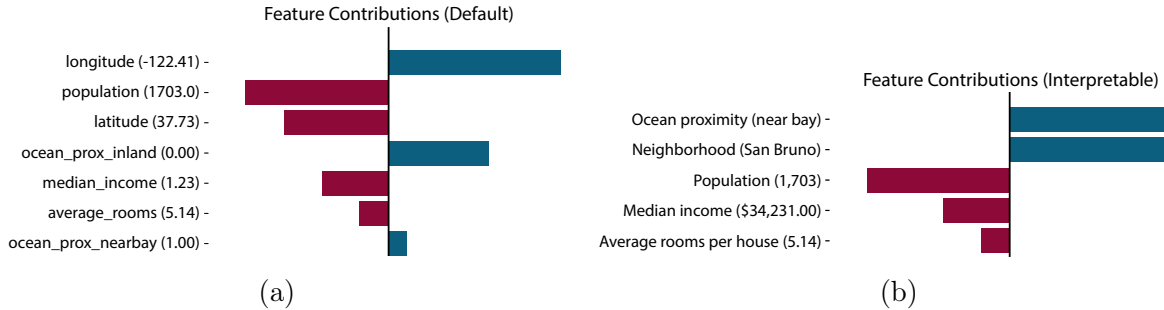


Figure 5.1: To illustrate the benefits of interpretable explanations, consider this hypothetical: A real estate agent is helping their client put in a winning offer for a house using an ML model trained to predict the median price of houses on a block. To better understand the contributing factors, the agent looks at the explanation shown in (a), a feature contribution explanation. They struggle to make sense of what they see there: for instance, how latitude (-122.41) contributes negatively, or longitude (37.73) contributes positively. Additionally, what do `ocean_prox_inland` (0.00) and `ocean_prox_nearbay` (1.00) mean, and how can a median income be 1.23? Now, imagine the same agent looking at the same ML model prediction via the explanation shown in (b). The agent now understands that the block is near the bay and in San Bruno, and its location has contributed to a higher price. Meanwhile, the median income for the block is now a meaningful \$34,231.

- We update the existing concept of transformers<sup>1</sup>. Our new transformers have two new key methods, alongside the traditional `fit` and `transform`. These methods *transform explanations* of diverse types between the input and output feature space of the transformer. As part of this contribution, we also add additional transformers with the primary objective of increasing interpretability.
- We introduce the concept of flagging transformers with whether they increase or decrease interpretability, as different users find different features interpretable.
- We develop and implement an algorithm that uses these new transformers to generate interpretable explanations.

We begin by introducing our proposed updates to traditional transformers. We then define the four feature spaces used by our transformers. Next, we describe our algorithm for interpretable explanations, built on these transformers. We then touch on Pyreal’s high-level API. Finally, we evaluate Pyreal’s usefulness, usability, and runtime performance.

Our examples in this chapter use the **California Housing Dataset** [70]. This dataset has nine features, with each row describing a block of houses in California. Our model predicts the median price of housing for each block.

<sup>1</sup>In this chapter the transformers we refer to are those that transform data between feature spaces, rather than transformer architectures such as those used by LLMs



Table 5.1: Summary of properties of Pyreal transformers

<b>Pyreal Transformer Architecture</b>
ML constructs
Operate between two feature spaces: A and B
Have three methods: data transform, explanation transform, inverse explanation transform
Can be annotated as model-ready, algorithm-ready, and/or interpretable

## 5.1 An Updated Transformer

Traditionally in ML, transformers are constructs that take in data in one format **A** and convert it into a different format **B**. This action is called a *transform* method. For example, a one-hot encoder takes in a categorical feature and returns Boolean features, one per category (**A**: categorical feature, **B**: one-hot encoded features). Similarly, a standard scaler takes in numeric features and returns those numeric features with consistent means and standard deviations (**A**: original numeric values. **B**: numeric values with consistent means/stds). Transformers are included in many ML libraries, such as `scikit-learn`, and are used in many ML pipelines. For clarity, in the remainder of this paper we will refer to this traditional transform method as a *data transform* method.

We extend traditional transformers. The key architecture aspects of Pyreal transformers is summarized in Table 5.1.

Pyreal transformers support two additional transform categories: explanation transforms and inverse explanation transforms.

Explanation transform methods take in explanations in terms of the transformer’s input space, and return an equivalent explanation in terms of its output space. Inverse explanation transforms take in explanations in terms of the transformer’s output space, and return an equivalent explanation in terms of its input space, effectively “undoing” the effects of the transform.

For example, consider the two explanations in Fig. 5.1. Both show feature contribution explanations for the same ML output. Fig. 5.1a shows the explanation in terms of one-hot encoded features (`ocean_prox_inland`, `ocean_prox_nearbay`), while Fig. 5.1b shows the equivalent explanation after applying a one-hot encoder’s inverse explanation transform method, which transforms back to the original categorical feature (`Ocean proximity`).

Details of how these new transforms work depend on the explanation algorithm. Explanations generated by different explanation algorithms have different theoretical requirements for transforming — transforming a feature contribution explanation generated by SHAP [16] is different than transforming a feature contribution explanation generated by LIME [15], which in turn is different than a counterfactual explanation [18]. Therefore, more formally,

Table 5.2: Sample data and inverse explanation transforms applied to two categories of explanations.

Transformer	Data Transformation	Additive Feature Contributions (SHAP)	Example-Based Explanations (Counterfactuals, Prototypes, etc.)
<b>One-Hot Encoder</b>	Transforms categorical features to set of Boolean features	Sums together contributions of one-hot encoded features	Decodes one-hot encoded features back to categorical
<b>Standardizer</b>	Standardizes numeric features to have consistent mean and variance	No change needed	Undoes normalization/standardization using saved constants
<b>Feature Selector</b>	Selects a subset of features to feed into the model	Sets contribution/importance score to 0	Sets unused features to “any”
<b>Numeric binning</b>	Assigns numeric values to categorical bins based on values	Takes contribution from bin corresponding to original value	Leaves as bin (specify thresholds)

our proposed updated transformer has the following methods:

A **data transform method** that transforms data in feature space A to equivalent data in feature space B (as in existing transformers).

An **explanation transform method**, For every explanation algorithm that works on data in feature space A, or on data that can be transformed into A, we write an explanation transform that converts the outputs of this explanation algorithm from feature space A to feature space B.

An **inverse explanation method**. For every explanation algorithm that works on data in feature space B or data that can be transformed to B, we write an inverse explanation transform that converts the outputs of this explanation algorithm from feature space B to feature space A.

The motivation behind adding both forward and inverse explanation transforms will be discussed in Section 5.3.

Note that transforms on explanations are not always theoretically possible, and often an explanation looks the same in multiple feature spaces. Our system is robust to these common scenarios, as explained in Section 5.3.


Table 5.2 shows examples of inverse explanation transforms methods for some common transformers on two common explanation types.

### 5.1.1 Transforming for Interpretability

In addition to updating existing transformers to improve explanation transformers, we developed additional transformers that have the primary goal of making data and explanations more interpretable.

A few examples of such transformers include:

1. A transformer that takes features in data or explanations that are already one-hot encoded and “decodes” them back into a categorical feature (`color_red: True, color_blue: False` to Red).

2. A transformer that converts RGB or HEX color codes to semantic color names or color swatches (#E1144D to **Crimson** or ).
3. A transformer that converts latitude and longitude coordinates to meaningful city, neighborhood, or place names (42.37 N, 71.11 W to **Cambridge, MA**).
4. A transformer that converts any kind of data or explanations into natural language descriptions, for example using a Large Language Model (LLM) (`color: red, contribution: -12` to `The model's prediction is reduced by 12 because of the object's red color`).

We note that these transformers are not inherently different from the transformers standard to existing ML systems — they have all the same functionality, and can be used in the same way.

As described in detail in Chapter 4, “interpretable” is a subjective term. Therefore, it is not enough to give our new transformers built-in labels as to whether their input feature space **A** is more interpretable than their output feature space **B** or vice-versa. Also, the addition of new transformers that are added only for interpretability — and potentially not supported by ML models and other algorithms — adds additional complexity. Therefore, users need an easy way to define their feature spaces and purposes of transformers. In the next section, we introduce this process.

## 5.2 Defining Feature Spaces

There are four types of feature spaces that must be considered when developing interpretable explanations using the Pyreal algorithm:

- **Original Feature Space.** The format in which the features are passed into Pyreal, at the first stage of featurization from raw data.
- **Model-Ready Feature Space.** The format that the model expects features to take, in order to take them in and produce outputs. This is the feature space usually required by classic algorithms that create ML models. For many models, this feature space requires all features be numeric, with no missing data, and possibly with standardized or otherwise scaled values.
- **Algorithm-Ready Feature Space.** This is the format that various algorithms that produce explanations or other ML augmentations expect features to take. Different algorithms require different feature spaces, though most of the explanation algorithms we work with expect the same feature space as the ML model they seek to explain.
- **Interpretable Feature Space.** The format of features that is most interpretable for users, as described in depth in the previous chapter. Note that this is an especially subjective feature space — different groups of individuals will have very different expectations of what is interpretable to them; therefore, as described later in this section, Pyreal allows users to easily flag which transformers increase the interpretability of features.

To better explain how these feature spaces work in practice, consider the example of features that convey information about the location of a block of houses, such as in the California Housing Dataset. Also, consider the common situation of an ML model that requires all features be passed to it in a numeric format, and an explanation algorithm that determines feature importance through permuting feature values (such as permutation feature importance [17]). These location features may take the following formats in each of the feature spaces:

- **Original:** Numeric latitude and longitude features  
`lat = 37.7, long = -122.5`
- **Model-ready:** Boolean one-hot encoded neighborhood  
`neighborhood_is_oceanview = True`
- **Algorithm-ready:** Categorical neighborhood feature  
`neighborhood = Oceanview`
- **Interpretable:** Categorical neighborhood with city label  
`neighborhood = Oceanview (San Francisco)`

Note that the algorithm feature space is categorical, as permuting one-hot encoded features may result in impossible rows of data where multiple or zero one-hot encoded columns are set to True. The model feature space could be latitude/longitude values or one-hot-encoded neighborhood values, depending on which improves model performance more.

Pyreal users can easily identify which transformers are required for the model or for interpretability through the `model` and `interpret` transformer flags, each set to True or False for every transformer. Combining the model flag with the known parameters of each explanation algorithm allows Pyreal to deduce whether each transformer is required for any particular algorithm (for example, permutation-based explanation algorithms know to avoid one-hot encoding data before permuting); Pyreal therefore automatically sets another flag, the `algorithm` flag, for any particular explanation algorithm.

### 5.3 The Interpretable Explanation Algorithm

Now that we have introduced our new transformers, we can use them to generate interpretable explanations. A few technical details add complexity to this process:

1. Not all transforms will be possible on explanations generated by all explanation algorithms.
2. Local explanations have two components: the explanation, and the values of the input data (see Figure 5.1, where the explanation is shown in bars and the input values are shown in parenthesis). These two components must be shown using the exact same set of features, even when one or more explanation transforms are not possible.

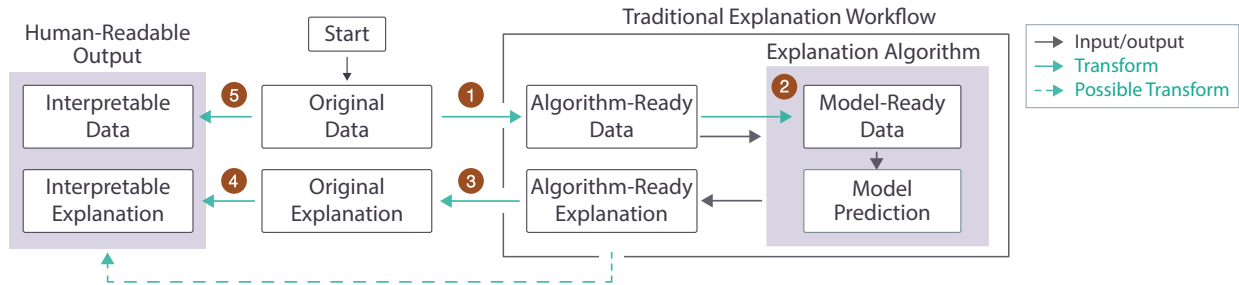


Figure 5.2: The interpretable explanation algorithm.

3. Some explanation algorithms expect a different feature space than the model expects (ie., as explained in the previous chapter, the algorithm and model feature spaces may be different).

We developed a algorithm for interpretable explanations that addresses the challenges posed by these technical details. This algorithm may involve both training and input data. Training data is the labeled dataset used to train the ML model that may be needed by some explainable ML algorithms. Input data is the new data for which the model generates outputs that we wish to explain<sup>2</sup>.

Here, we provide a simple summary of our algorithm (visualized in Figure 5.2), which we will then walk through more formally.

- 1 **First, the training and input data is prepared for the explanation algorithm** by running the data transform methods of the appropriate transformers.
- 2 **Then, the explanation algorithm is run on the input and training data.** Many algorithms require running the model to get ML outputs, possibly on perturbed versions of the training or input data. In this case, some additional transformers' data-transform methods are run. The result is an explanation in the explanation-algorithm feature space.
- 3 **Next, any transforms that make the data and explanation less interpretable are “undone” on the explanation, to the extent possible.** This is done by running the inverse explanation transform methods of the appropriate transformers, up until the point where one of these transforms is impossible. At this point, the input data is transformed with any transforms that could not be undone, to ensure the data and explanation are in the same feature space.
- 4 **Any additional transforms that improve interpretability, but were not used by the explanation algorithm, are applied to the explanation** by running the explanation transform method of the appropriate transformers up until the point where the explanation transform becomes impossible.

<sup>2</sup>We describe the algorithm for local explanations that include both input and training data; for global explanations, only the training data is used

---

**Algorithm 1** Feature Space Transformation and Explanation Algorithm

---

**Require:** Original data  $X$

**Require:** List of Transformers  $T$  with model, algorithm, and interpret flags

**Require:** Model to explain  $M$

```
1: procedure DATA TRANSFORM( $X$ , flags)
2:    $X' \leftarrow$  copy of  $X$ 
3:   for  $t$  in  $T$  if  $t$  has flags do
4:      $X' \leftarrow t.data\_transform(X')$ 
5:   return  $X'$ 
6: procedure MODEL PREDICT( $X_A$ )
7:    $X_M \leftarrow$  DATA TRANSFORM( $X\_A$ , flags: [model=True, algorithm=False])
8:   return  $M.predict(X_M)$ 
9:  $\triangleright$  Step ①. Transform data for explanation algorithm ◁
10:  $X_A \leftarrow$  DATA TRANSFORM( $X\_A$ , flags: [algorithm=True])
11:  $\triangleright$  Step ②. Run the explanation algorithm, possibly using MODEL PREDICT ◁
12:  $E_A \leftarrow$  produce_explanation( $X_A$ )
13:  $\triangleright$  Step ③. Inverse explanation transforms ◁
14:  $E_{ORIG} \leftarrow$  copy  $E_A$ 
15: for REVERSE( $t$  in  $T$  if  $t.algorithm$  is True and  $t.interpret$  is False) do
16:    $E_{ORIG} \leftarrow t.inverse\_explanation\_transform(E_{ORIG})$ 
17:  $\triangleright$  Step ④. Transform the explanation to the interpretable space ◁
18:  $\mathbb{E}_I \leftarrow$  copy  $\mathbb{E}_{ORIG}$ 
19: for  $t$  in  $T$  if  $t.algorithm$  is False and  $t.interpret$  is True do
20:    $E_I \leftarrow t.explanation\_transform(E_I)$ 
21:  $\triangleright$  Step ⑤. Transform input data to the interpretable space ◁
22:  $X_I \leftarrow$  DATA TRANSFORM( $X$ , flags: [interpret=True])
23: return  $E_I, X_I$ 
```

---

- ⑤ **The input data is transformed to the interpretable feature space.** The input data is transformed with the data transform methods of all transformers that were also successfully run on the explanation. The final data and explanation are presented together.

This algorithm is defined more formally in Algorithm 1.

### 5.3.1 Formal Equation for the Pyreal Algorithm

We now state the Pyreal algorithm for interpretable explanations in an alternative, formal notation.

As in traditional ML pipelines, let  $X$  represent model-ready tabular training data, and  $x$  represent some input data vector to the explanation algorithm. Let  $D$ ,  $I$ , and  $A$  represent the equivalent data in the original, interpretable, and algorithm feature spaces respectively, for some given explanation algorithm. Let  $m(X)$  be an ML model that takes a model-ready data and returns some output vector  $y$ . Therefore:

$$m(X) \rightarrow y$$

Let  $e(A, m, g_{ax})$  be an explanation algorithm that generates some explanation given a model, algorithm-ready data, and the method required to transform algorithm-ready data into model-ready data. Let  $E_A$  be an explanation presented in the same feature space as matrix  $A$ . Therefore,

$$e(A, m, g_{ax}) \rightarrow E_A$$

**Definition A.1 [Transform method]** A transform method  $g_{uv}(\cdot)$  converts the data from the feature space  $u$  to the feature space  $v$ . Pyreal uses the following transform methods:

$$\begin{aligned} g_{da}(D) &\rightarrow A \\ g_{ax}(A) &\rightarrow X \\ g_{di}(D) &\rightarrow I \end{aligned}$$

**Definition A.2 [Explanation transform method]** An explanation transform method  $h_{uv}^e(\cdot)$  converts an explanation generated by  $e$  from feature space  $u$  to feature space  $v$ . Pyreal uses the following explanation transform methods:

$$h_{di}^e(E_D) \rightarrow E_I$$

**Definition A.2 [Inverse explanation transform method]** An inverse explanation transform method  $h_{uv}'^e(\cdot)$  converts an explanation generated by  $e$  from feature space  $v$  to feature space  $u$ . Pyreal uses the following inverse explanation transform methods:

$$h_{da}'^e(E_A) \rightarrow E_D$$

Our objective is to get interpretable data and explanations  $I$  and  $E_I$  given the original data  $D$  and a model  $m$ . Thus, our formal problem formulation can be written as:

$$\begin{aligned} I &= g_{di}(D) \\ E_I &= (h_{id}^e \circ h_{ad}'^e \circ e(g_{da}(D), m, g_{ax})) \end{aligned}$$

In the next section, we go through how this algorithm for interpretable explanations is run through Pyreal’s high-level API.

## 5.4 RealApps: The Interpretable Explanation Algorithm Encapsulated

In this section, we introduce the Pyreal objects that run our algorithm to generate interpretable explanations. Pyreal’s high-level API is realized through `RealApps` — fully encapsulated interpretable explanation objects.

```

1  from pyreal import visualize
2
3  # Load in premade realapp and data (external functions)
4  app = load_realapp()
5  X_to_explain = load_data_to_be_explained()
6
7  exp1 = app.produce_feature_contributions(X_to_explain)
8  exp2 = app.produce_similar_examples(X_to_explain)
9
10 visualize.feature_bar_plot(exp1)

```

Figure 5.3: Decision-maker workflow with Pyreal. Decision-makers use RealApp objects to produce a variety of explanations, each with a single line of code. Figure 5.1b has a sample output of the final visualize function.

Users looking to use and understand ML models, generally decision-makers and bridges, use RealApp objects primarily through their *produce* methods, one of which exists per *explanation type* (the kind of information offered by the explanation). For example, the *produce feature importance* method returns one positive value for each feature, representing that feature’s overall importance to the ML model, and the *produce similar examples* method gives a few examples from the training dataset that are similar to the input data. Produce methods kick off the full interpretable explanation algorithm introduced in Section 5.3. The Python code used to generate explanations using RealApp objects is shown in Figure 5.3.

These users do not need to consider the specific explanation algorithm used (ex. SHAP, LIME, etc.). We chose this design as decision-makers and bridges are more concerned about what information they want to see rather than the specific algorithmic details used to get that information, which requires more extensive ML expertise.

RealApps return explanations in data structures that make them easy to access and use. For example, feature contribution explanations are indexed by input row IDs and include tables with feature name, value, and contribution columns for each input row of data. These tables can then be plugged into and visualized with systems such as Pyreal’s visualization module, Tableau, or our generalizable GUI system (see Chapter 6)

RealApps contain all the components needed to produce interpretable explanations, including one or more ML models, transformers with appropriate flags, and training data. Users with more ML experience, such as ML developers, can set up RealApp objects for use by decision-makers by passing in all these components, as well as optional configurations such as a dictionary of readable feature names. Figure 5.4 shows the process of creating a new RealApp object. Note that transformers can either be selected from Pyreal’s pre-defined set, or new transformers can be created by extending the base transformer class and defining their data and explanation transform methods.

### 5.4.1 Implementation

In this Section, we dive a bit deeper into the specific architecture used by our library implementation of the Pyreal system.



```

1 from pyreal import RealApp
2 from pyreal.transformers import Imputer, OneHotEncoder, NeighborhoodEncoder
3
4 transformers = [Imputer(model=True, interpret=True),
5                 OneHotEncoder(columns="ocean_proximity",
6                               model=True, interpret=False),
7                 LatLongToNeighborhoodEncoder(model=False, interpret=True)]
8
9 # Load in data and model (external functions)
10 data = load_training_data()
11 model = train_model(data, transformers)
12
13 feature_descs = {"med_income": "Median income",
14                 "ocean_prox": "Ocean Proximity", ...}
15
16 app = RealApp(model, X_train_orig=data,
17               transformers=transformers,
18               feature_descriptions=feature_descs)

```

Figure 5.4: ML Developer workflow. **Step 1 (lines 5-10):** the Pyreal transformers are initialized and flagged accordingly. In this example, the data is imputed and one-hot-encoded for the ML model, and the latitude and longitude features are converted to a neighborhood feature for interpretability. **Step 2 (lines 12-13):** The ML data and model are prepared as in traditional ML workflows. **Step 3 (lines 15-21):** A `RealApp` object is initialized with the transformers and a feature description dictionary to improve the readability of feature names, which can now be used to produce interpretable explanations as in Figure 5.3.

Pyreal’s internal implementation includes three class types: transformers, as previously introduced; explainers, which run explanation algorithms; and explanation types, which unify the explanation transform process between explanations with similar properties. Modularity through these classes enables Pyreal to stay up-to-date with state-of-the-art explanation algorithms while remaining backward compatible, and prevents the need for major refactors or code reuse. Currently, we offer a starting set of five explainer families, twelve transformers, and five families of explanation output types, as summarized in Tables 5.3, 5.4, and 5.5. More details on the structure of Pyreal can be found in our documentation<sup>3</sup>.

The Pyreal class structure is summarized in Figure 5.5, and discussed in further detail through the rest of this section. `RealApp` and transformer objects were introduced in earlier sections; we now discuss the remaining two class types.

**Explainer** objects generate interpretable explanations. They have a `fit` method that performs any computation-intensive procedures to prepare for generating explanations and a `produce` method that generates an explanation. For local explainers, `produce` takes in input data and generates an interpretable explanation for the ML output on that input; for global explainers, the method takes no required arguments and generates the an explanation for the ML model’s general logic.

Explainers come in three types. Algorithm explainers implement a specific explanation

---

<sup>3</sup><https://dtaill.gitbook.io/pyreal>

Table 5.3: List of Pyreal Explainer classes currently supported.

<b>Generic Explainer</b>	<b>Algorithm Explainer</b>	<b>Functionality</b>
Local Feature Contribution	SHAP Feature Contribution	Uses SHAP
	LIME Feature Contribution	Uses LIME
	Simple Permutation	Permutates feature values
Global Feature Importance	SHAP Feature Importance	Uses SHAP
	Permutation Feature Importance	Using permutation feature importance
Decision Tree	Surrogate Decision Tree	Trains a surrogate decision tree model
Partial Dependence	Partial Dependence	Generates partial dependence plots
Similar Examples	Nearest Neighbors	Finds similar examples using NN

Table 5.4: List of Pyreal Transformer classes currently supported.

<b>Category</b>	<b>Transformer</b>	<b>Functionality</b>
Feature Selection	Feature Select	Select important features
	Column Drop	Drop specific columns
Scalers	Min-Max Scaler	Scale features using min-max normalization
	Standard Scaler	Scale features using standard scaling
	Normalizer	Normalize feature values
Imputers	Imputer	Impute missing values
One-Hot Encoders	One-Hot Encoder	One-hot encode categorical variables
	Mappings One-Hot Encoder	Encode using specific mappings
Type Casters	Bool to Int	Cast boolean to integer
Interpretability	Lat/Long to Place	Convert lat/long coordinates to place names
	Mappings One-Hot Decoder	Decode one-hot encoded variables
	Narrative	Convert to natural-language using LLM

Table 5.5: List of Pyreal Explanation Types currently supported.

Category	Class Name	Functionality
Feature-Based	Feature Importance	Global feature importance scores
	Additive Feature Importance	Feature importance scores that can be meaningfully added together
	Feature Contribution	Local feature contribution scores
	Additive Feature Contribution	Feature contribution scores that can be meaningfully added together
Feature-Value-Based	Partial Dependence	Partial dependence plots
Decision Tree	Decision Tree	Decision tree surrogates of models
Example-Based	Similar Examples	Similar examples
	Counterfactuals	Counterfactual explanations
	Prototypical	Prototypical examples

algorithm to generate explanations. Generic explainers generate explanations of a broader type, selecting the algorithm to use based on the current state-of-the-art. Base explainers are abstract classes that handle overhead code that is shared by all explainers that generate explanations of a common type. For example, the `FeatureImportance` generic explainer class extends the abstract `FeatureImportanceBase` base explainer class and wraps a `ShapFeatureImportance` or `PermutationFeatureImportance` algorithm explainer (which implement the SHAP [16] or Permutation Feature Importance [71] algorithms). All base explainers extend the `BaseExplainer` parent class, which contains all general-use explainer functionality such as transforming data between feature spaces, running the ML model, and formatting data.

This structure offers two benefits. First, Pyreal abstracts out the choice of specific explanation algorithms in favor of a focus on outputs. Second, adding new explanation algorithms requires minimal effort and new code, because as much shared code as possible is handled by parent classes.

**Explanation types** unify outputs from explainers that represent similar kinds of information. The other components of Pyreal, such as transformers and visualization functions, use these objects for abstracted implementations. This unification reduces the number of times contributors have to write new functionality such as [inverse] explanation transform methods.

Explanation types are hierarchical. For example, local feature contributions and global feature importance are both subtypes of the *feature-based* explanation type which encapsulates any explanation that gives importance scores to features. Additive local feature contributions or global feature importance further extend these types with the additional property of being able to have their importance scores added together meaningfully. Counterfactuals [18] and similar examples are both subtypes of the *example-based* explanation

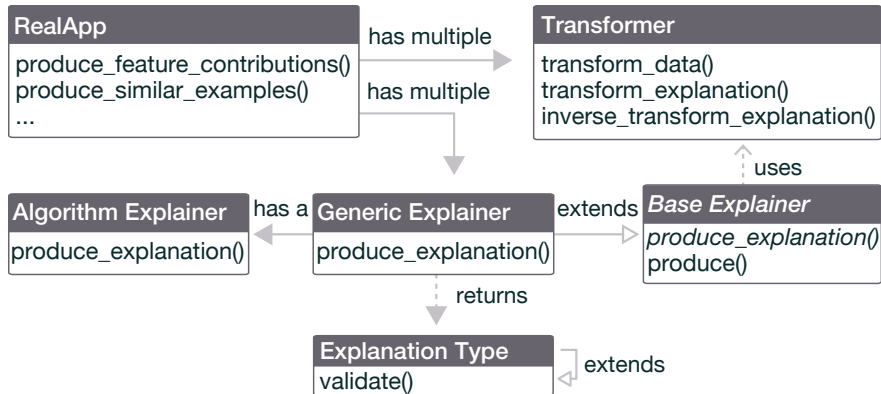


Figure 5.5: Pyreal class structure. Italics refers to abstract classes or methods. The workflow of interactions between these classes is as follows: RealApps contain all the necessary explainers and transformers for the interpretable explanation pipeline. Explainers use transformers to transform input and training data to the feature space they require — to avoid code reuse, this process is handled by parent base explainer classes — and then run explanation algorithms on this data. They return the explanation using the appropriate explanation type class from the type hierarchy. The explainer then uses the same transformers on this explanation type object to transform it to the interpretable feature space.

type that explains model predictions with illustrative examples.

The hierarchical structure of explanation types allows functionality to be reused. For instance, for any example-based explanation, the explanation is a valid row of input data to the model, so the transform-explanation method is the same as the data-transform method. Or, for any feature-based explanation that gives importance scores to features, the same bar-plotting functionality can be used.

Having introduced our Pyreal library implementation, we now turn to our evaluations of this library, as well as the explanations it produces.

## 5.5 Evaluation

We evaluated Pyreal on several axes: the interpretability of its explanations, the ease-of-use of its API, its runtime performance, and its real-world usefulness through a series of user studies, experiments, and case studies.

Throughout our evaluations, we used ML models trained on three datasets: the California Housing dataset which includes 9 features about blocks of houses as introduced earlier in this chapter; the Ames Housing dataset [72] which is a more complex housing-price dataset with 79 input features as introduced earlier in this thesis; and the Student Performance Dataset [73], which has 30 features, with each row describing a student and a target variable of whether the student will pass or fail their Portuguese class.

Table 5.6: Summary of participant groups in the evaluation of explanation interpretability. *Low data expertise* includes participants who ranked their experience with data science, ML, AI, or statistics at 2 or less (out of 5). *High data expertise* includes participants who ranked their experience with data science, ML, AI, or statistics at 3 or more. Real estate expertise was determined based on the answer to the question “do you have any experience working in the real estate industry”. General populace participants have low data expertise and low real-estate expertise. RE: real-estate.

<b>13</b>	<b>36</b>
General populace	Data experts
<i>low data expertise and low RE expertise</i>	<i>high data expertise and low RE expertise</i>
<b>17</b>	<b>13</b>
RE experts without data expertise	RE experts with data expertise
<i>low data expertise and high RE expertise</i>	<i>low data expertise and high RE expertise</i>

### 5.5.1 Evaluating the Interpretability of Pyreal Explanations

We begin by comparing the interpretability of Pyreal-generated explanations to those directly generated by the wrapped libraries that Pyreal uses under-the-hood.

#### Participants

We recruited 94 participants for this study from a random global pool using Prolific<sup>4</sup>. When reporting results, we sort them into categories based on their answers to two questions. First, we asked participants to rate the extent of their experience working with data science, statistics, ML, or AI on a scale from 0 (none) to 5 (expert). We label users who rated their experience at 2 or less as having low data expertise. Those who rated their experience as 3 or more are labelled as having high data expertise. We then asked participants if they had experience working in the real estate industry; those who answered no are labeled as having low RE (real estate) expertise, those who answered yes have high RE expertise. Categorizing on these two axes, we get four participants groups, as summarized in Table 5.6.

#### Method

For this evaluation, we generated explanations of ML models trained on the California Housing dataset and the Student Performance dataset. The explanations were feature contribution explanations generated using the SHAP algorithm, either in the form of a bar plot (explaining a single prediction, as shown in Figure 5.1) or a scatter plot (showing contributions across the dataset for a single feature, as shown in Figure 5.6). Each explanation was

<sup>4</sup>www.prolific.com

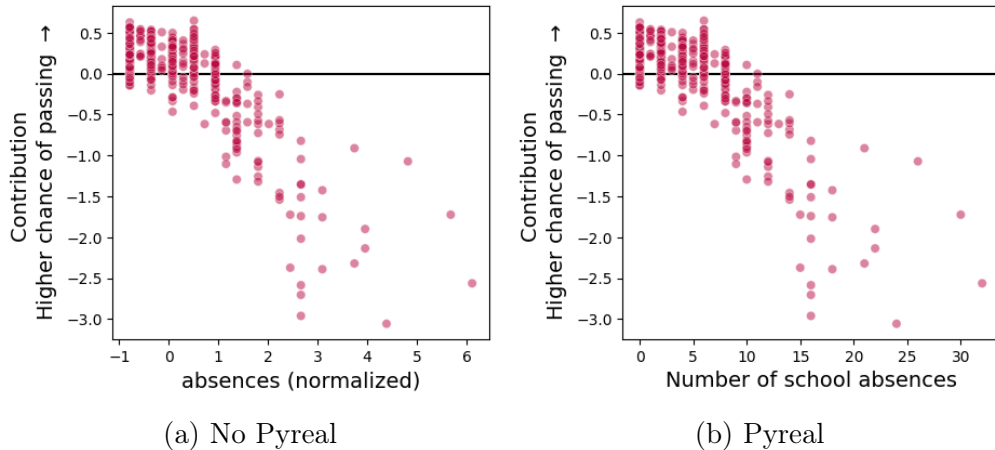


Figure 5.6: Sample figures shown to users in the explanation-interpretability user study. Participants were shown either the Pyreal explanation or the no-Pyreal explanation for each explanation.

either in the interpretable feature space (Pyreal condition) or in the model-ready feature space (no-Pyreal condition).

For this study, our interpretable feature space was set to one that we believed to be most understandable by the general populace of non-ML-experts. This meant the explanations showed features in categorical formats rather than one-hot encoded, and feature values were not standardized. For the California Housing dataset, latitude and longitude values were transformed to neighborhood names.

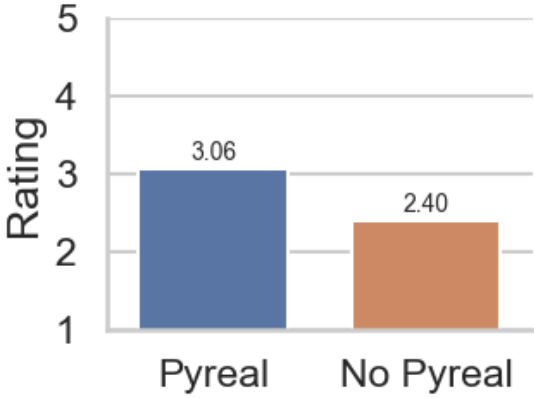
In total, this study included 2 datasets  $\times$  2 conditions  $\times$  5 individual plots (3 bar, 2 scatter) = 20 explanations. Each participant was shown one condition for one dataset, and the other condition for the other dataset (10 total explanations per participant). The within-subjects design allowed us to account for differences between participants, while the between-datasets design prevented participants from learning about the meaning of features from Pyreal explanations or vice versa.

For each explanation, participants were asked to 1) rate the usefulness of the explanation on a Likert-type scale from 1 (not at all useful) to 5 (extremely useful), 2) describe what made the explanation more useful, and 3) describe what made the explanation less useful. Answers 2 and 3 were given in open response format.

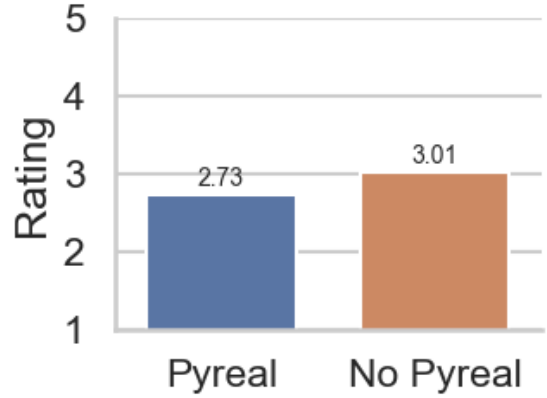
## Results from participants without RE expertise

We begin by analyzing the results from participants without RE expertise, with extra consideration for those who additionally do not have data experience participants, for whom we tuned the interpretable feature space used.

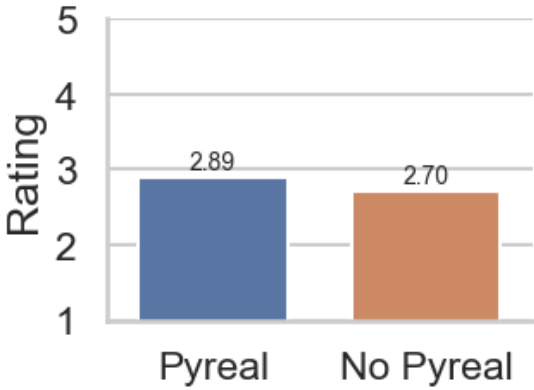
The average usefulness scores given by each participant group in summarized in Figure 5.7. General populace participants (those with low data expertise and low RE expertise) considered Pyreal explanations to be more useful on average than no-Pyreal explanations. They rated Pyreal explanations as being extremely useful and very useful for 7.23% and 33.06% of explanations respectively, compared to 0% and 19.51% of explanations for the



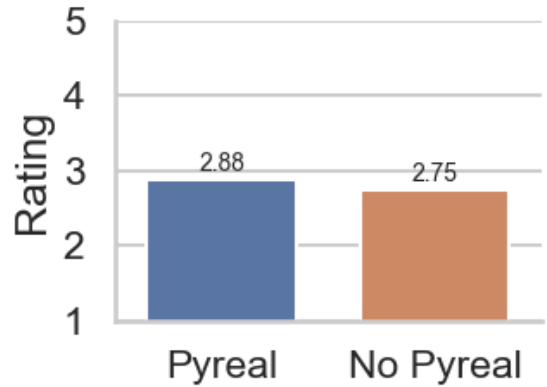
(a) General populace participants (low data expertise, low RE expertise)



(b) Data expert participants (high data expertise, low RE expertise)



(c) RE expert participants without data expertise (low data expertise, high RE expertise)



(d) RE expert participants with data expertise (high data expertise, high RE expertise)

Figure 5.7: Average usefulness scores given by each participants group to Pyreal and no-Pyreal explanations in response to the question “how useful did you find the explanation?” A score of 1 represents a response of “not at all useful”, and a score of 5 represents a response of “extremely useful”

no-Pyreal explanations.

Data expert participants (those with high data expertise and low RE expertise) saw less usefulness from Pyreal. This is likely because the explanations in this study use an interpretable feature space tuned for an audience of individuals without data experience, and therefore do not use the kind of precise features that data experts may be used to.

We conducted a thematic analysis on the open-response answers to further investigate what made explanations more or less useful to the general populace participants in particular. Here, we describe the common themes seen in responses to open-response questions on both Pyreal and no-Pyreal explanations.

**Interpretable features are essential for understanding.** One common theme we identified for no-Pyreal explanations was confusion related to the model-ready feature space

used, which included normalized feature values. Of the 23 general populace participants who answered the question “what made the explanation less useful” on any no-Pyreal explanation, 21 reported difficulties understanding the information (features) presented or difficulties interpreting the normalized feature values. For example, P22 said, “*what does -0.58 bedrooms mean? No range, no context*”. P14 said, “*what does ‘fjob\_other’ mean?*” and “*longitude and latitude separately is strange*”.

In comparison, only 9 general populace participants reported any confusion related to understanding features on the Pyreal explanations, and most of these concerns related to wanting more detailed information about how the features were computed or what they meant in context. For example, P7 reported that they did not understand the relationship of the characteristics (features) to the purpose of the model (predicting median house price). We believe these kinds of concerns would be less of an issue in a real-world use case where decision-makers are domain experts, or would otherwise receive further context on the problem beyond the brief introduction we were able to give for this study.

Only one comment suggested that using the interpretable feature space reduced the usefulness of the explanation in any way, commenting that the Pyreal explanation (showing the neighborhood rather than lat/long values) did not provide an “accurate location”.

These findings motivate our belief that explanations are more useful to decision-makers without ML expertise when presented using the interpretable feature space.

**General populace participants did not shy away from calling the no-Pyreal explanations “useless”** Another common theme seen only in comments on no-Pyreal explanations were direct and unspecific complaints about the explanations’ lack of clarity. P6 commented on a no-Pyreal explanation “*this is a very useless method of data visualization*”. P3 said on another no-Pyreal explanation “*the factors are very vague and not very helpful*”. P13 described the no-Pyreal explanations as “*totally not relatable*”, and P4 said simply “*everything [makes this explanation less useful], I do not understand this*”.

**With Pyreal explanations, general populace participants were able to dig deeper into the ML model itself.** A theme in comments on Pyreal explanations was concerns about the logic the ML model used to produce outputs (7 cases for Pyreal explanations, but only 1 for no-Pyreal). For example, P14 said with regards to a Pyreal explanation “*Median income has too great impact on price prediction*”. The goal of ML explanations is to enable participants to understand where ML outputs come from, and because the model logic was the same in both conditions, this finding suggests that Pyreal explanations were more successful.

We note that despite instructions to the contrary, some participants did report reducing their usefulness rating of explanations due to disagreeing with the model’s logic; we believe factoring this in could increase the average rating of Pyreal’s usefulness.

## Results from participants with RE expertise

We now consider the results from participants with real estate expertise. For these participants, we only consider responses to explanations on the California Housing dataset, on which their domain expertise applied.

The reported usefulness of Pyreal was slightly lower among RE experts, as shown in Figure 5.7. This finding was not surprising, as it is expected that people have higher standards



for information about their field<sup>5</sup>. Additionally, as discussed below, RE experts may want more detail about the context around the hypothetical decision problem.

We now go through some of the findings from the responses to the open-response questions from the RE expert participants.

**RE expert participants felt no-Pyreal explanations were not appropriate for them.** Several RE expert participants with and without data expertise mentioned that they felt they lacked the necessary knowledge to understand the no-Pyreal explanations, despite being experts in RE. P87 said about a no-Pyreal explanation: “*Not understandable values for mathematical non-literate people.*”. P89 said, “*I’ll be honest that I’m not a numbers guy and I don’t understand what ‘normalised’ means.*” This theme did not appear in responses on Pyreal explanations.

**The decision problem context is especially important to RE experts, and carefully crafted features can help establish this.** One source of confusion we saw from RE experts with the Pyreal explanations on the California Housing dataset related to misunderstanding or needing more details about the broader context. For example, in the California Housing Dataset, each row of data refers to a *block of houses*, but most real estate experts tend to be used to reasoning about the prices of *individual houses*. The study instructions gave this context, but the scenario may have been so far out of the norm for participants that they regardless expected features to be presented in terms of individual houses. For example, P88 incorrectly assumed that “...*‘number of households’ refers to the number of households that would occupy a single home*” and P95 reported being unsure why the total number of bedrooms was 190 (a very high value for a single house). This finding reveals two lessons: first, ML test datasets like this one often do not effectively represent real-world decision problems, which may affect the effectiveness of user studies. Second, this further reveals the impact that the format of features may have on understanding.

**RE expert participants had similar needs for interpretable features.** Like our participants without RE expertise, the RE expert participants frequently cited a lack of understanding about what the features or their normalized values referred to. 13 out of 19 participants who commented on no-Pyreal explanations made at least one comment about not understanding the features or their values. For example, P91 said, “...*what does -0.62 mean for bedrooms? That the home has less than one room?*” For comparison, 5 out of the 10 participants who commented on Pyreal graphs expressed confusion related to not understanding features, though 3 of these 5 appeared to be confused about the context as explained above (thinking rows referred to individual houses instead of blocks, and thus being confused about large feature values).

### 5.5.2 Directly comparing the interpretability of Pyreal and no-Pyreal Explanations

To further investigate the usefulness of Pyreal explanations, we conducted a second study that enabled participants to more directly compare Pyreal explanations to no-Pyreal. For this study, we recruited 20 new participants, 12 of whom rated their ML expertise as 2 or

---

<sup>5</sup>Indeed, the RE expert participants gave slightly higher average usefulness scores on the student performance predictions, of 2.95 (Pyreal) and 2.96 (no-Pyreal)

Table 5.7: Summary of participants in each condition of our library usability study.

Condition	Recruitment source	# of Participants	Years of ML experience (mean)
no-Pyreal	Prolific	10	4.6
Pyreal	Upwork	17	0.36

less out of 5.

For this study, we presented the same set of explanations, but showed two explanations at a time side-by-side. One was generated using Pyreal and used the interpretable feature space, and the other was the equivalent explanation generated directly with wrapped libraries and used the model-ready feature space. Each participant was shown 10 pairings.

For each pair of explanations, we asked participants to select the one they thought was more useful. Among the 12 low-data-expertise participants, the Pyreal explanation was selected as more useful in 110 out of 120 total pairings (91.67%). Among high-data-expertise, the Pyreal explanation was selected as more useful in 61 out of 80 total pairings (75.25%).

### 5.5.3 Evaluating the Usability of the Pyreal library

We ran a user study to understand the process participants use when producing interpretable explanations without Pyreal, and to compare the time and effort required to produce interpretable explanations with and without Pyreal. We refer to this study as our library usability study.

This study included two conditions, with two different participant groups, as summarized in Table 5.7. For the no-Pyreal condition of this study, we posted a job to Upwork<sup>6</sup> requesting ML developers with experience working with the Python packages that are useful for generating ML explanations (`pandas`, `scikit-learn`, `shap`). Out of 48 candidate proposals we randomly selected 10. The selected developers had between 2 and 8 years of ML development and python experience (mean: 4.6 years). Additionally, all participants had experience working with the `pandas` and `scikit-learn` Python packages, and 6 had experience working with the `shap` python library.

For the Pyreal condition we hired from a wider pool through Prolific, filtering only for Python coding experience. Using Prolific allowed us to take more participants, and because the low-code Pyreal API is designed to be usable by developers without ML expertise, we no longer had to select for this. 17 participants completed this condition of the study. The participants had between 6 months and 10 years of Python coding experience (mean: 2.62 years). Twelve participants reported having some ML development experience, with a mean ML experience across all participants of 0.36 years.

<sup>6</sup><https://www.upwork.com/>

```

1 # Create new explanation application object from the sklearn pipeline
2 # In the study, the pipeline objects are provided in a pickle file
3 realapp = RealApp.from_sklearn(pipeline, X_train=X_train, y_train=y_train)
4
5 # Produce a feature contribution explanations
6 exp = realapp.produce_feature_contributions(days_of_interest)
7
8 # Visualize the explanation for the first block
9 feature_bar_plot(exp[0], num_features=8, select_by="absolute")
10
11 # Visualize the explanation for the second block
12 feature_bar_plot(exp[1], num_features=8, select_by="absolute")

```

Figure 5.8: Tutorial code given to library usability study participants in the Pyreal condition.

## Method

In both conditions, participants were asked to generate and visualize three interpretable explanations on each of three datasets, for a total of nine explanations. All input data and expected output explanations were the same across both conditions. The explanations were local feature contribution explanations generated using SHAP. We asked participants to generate multiple explanations per dataset to motivate developing abstractions in the no-Pyreal condition, so we could compare these abstractions to the ones used by Pyreal.

For each dataset, participants were given a small dataset and a fitted scikit-learn `Pipeline` object that included the ML model and all transformers required to transform the data to make model predictions.

The no-Pyreal condition participants were then given the specifications for interpretable explanations (presenting values unstandardized and not imputed, with categorical feature values presented in their original form) and asked to generate a basic bar plot of such explanation on three inputs, using SHAP. Participants were asked to avoid using `KernelExplainers` (which can take an sklearn pipeline directly) to mimic scenarios when `KernelExplainers` are not desirable due to slow runtime and potentially less accurate explanations.

The Pyreal condition participants were given sample code generating interpretable explanations using Pyreal (including using the `RealApp.from_sklearn()` function, that converts an sklearn pipeline to a prepared `RealApp` object). This brief sample code served to familiarize users with the library, and is shown in Figure 5.8. The only changes participants had to make from the sample code was 1) substitute in the correct input data for each dataset and 2) add one additional visualization line.

For this study, we replaced the Ames Housing dataset with the Iranian Churn dataset [74], as the latter required fewer transformers and offered an easier first task to warm up participants. The Iranian Churn dataset provides information about customers for a cell phone company, and whether or not they churned within a timeframe. This dataset required imputing and standardization. The California Housing dataset required imputing, standardization, and one-hot encoding and the Student Performance dataset required one-hot encoding, ordinal encoding, and standardization.

## Results

We analyzed participant code to understand their process, and evaluated the accuracy of their code and time taken. Here we summarize our key findings.

Table 5.8: Summary of mistakes made by participants in the no-Pyreal condition of our developer study. Some participants made multiple mistakes.

Type of Mistake	Number of Participants
Incorrectly ran the same transformer multiple times in different parts of the explanation generation process	1
Missed one or more data transforms required to run the model	4
Did not correctly remove features that had been incorporated into other features	2
Displayed transformed data in the final explanations	1
Did not correctly match feature names to contribution values	1

**Generating interpretable explanations is an error-prone process, but not with Pyreal.** 8 out of 10 of our no-Pyreal condition participants made mistakes on one or more datasets that significantly modified the explanation output. The types of mistakes made are summarized in Table 5.8. Most mistakes related to incorrectly over- or under-transforming data.

Notably, in most cases these were fairly minor mistakes that resulted in explanations that seemed reasonable, but were in fact not accurate SHAP values for the model’s logic. Such errors are especially dangerous because they are difficult to identify.

In contrast, only 1 out of 17 Pyreal condition participants made mistakes, despite the participants being less experienced on average. Generating interpretable explanations with Pyreal follows an almost identical process across diverse datasets, so following sample code is sufficient for error-free explanations. The one participant who made errors successfully initialized a RealApp using the standard constructor, but incorrectly called a non-existent explain method instead of `produce_feature_contributions`.

**Generating interpretable explanations is a slow process that is sped up by Pyreal.** Time taken to generated interpretable explanations in the no-Pyreal condition on all three datasets ranged from 4 hours to 5.5 hours (mean: 4.87 hours); one participant stopped early, taking 5 hours to complete 2 datasets. This is especially notable as, despite the

datasets being relatively straightforward compared to many real-world datasets, experienced developers required more than 1.5 hours per dataset on average to generate interpretable explanations.

In the Pyreal condition, participants only required 23.88 minutes on average (min: 5 minutes, max: 55 minutes, std: 15.93 minutes) to generate all interpretable explanations, which included the same preparation code as in the no-Pyreal condition. The complete distribution of explanation generation times for both conditions is presented in Figure 5.9.

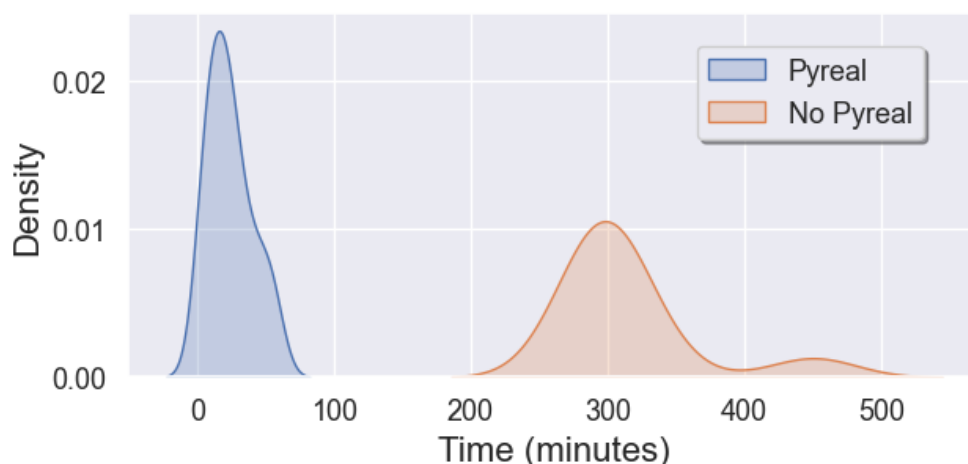


Figure 5.9: Distribution of times to generate explanations across both conditions. For participants who did not complete all three datasets, we scaled the finish time based on the time for the datasets completed.

**Experienced ML developers implement Pyreal’s logic.** Participant code in the no-Pyreal condition mirrored the process used by Pyreal. For example, participants maintained the original feature values for use alongside explanations and “undid” one-hot encodings. These participants wrote out this process for each dataset, manually adjusting the columns used.

**Participants considered Pyreal easy-to-use and reported being likely to use it more in the future.** In the Pyreal condition, we finished the study by asking participants two retrospective questions about Pyreal:

1. **Ease of Use:** How easy was it to generate explanations using Pyreal? Please answer on a scale from 1 to 5, where 1 means very difficult and 5 means very easy, and briefly explain your rating.
2. **Likelihood of Use:** If you found yourself in a situation where you needed to get explanations of ML model predictions, how likely would you be to use Pyreal? Please answer on a scale from 1 to 5, where 1 means very unlikely and 5 means very likely, and briefly explain your rating.

The Pyreal condition participants scored Pyreal’s ease of use at 4.07 out of 5 on average. They scored their likelihood of using Pyreal in the future at 4.13 out of 5 on average. Only

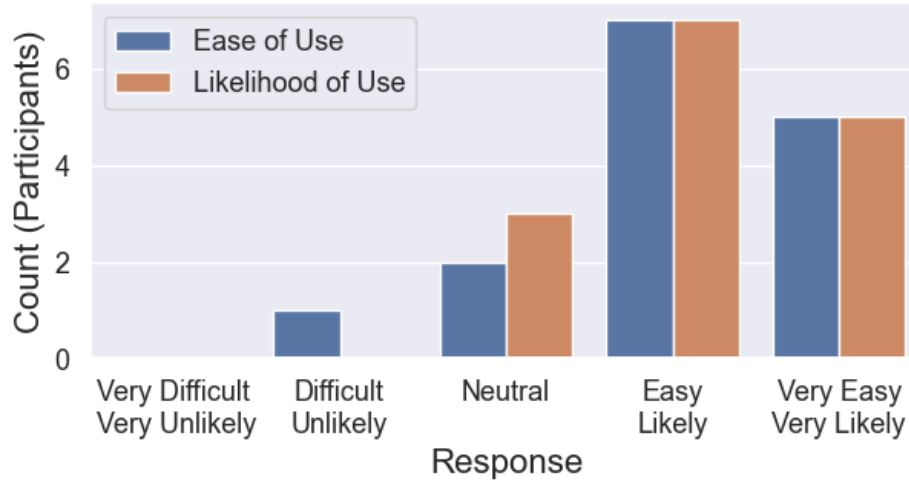


Figure 5.10: Result of retrospective questions asked to participants in our developer user study.

one participant scored ease-of-use at 2; no other participant scored either metric lower than 3. Figure 5.10 shows the exact distribution of responses to the retrospective questions.

Additionally, participants’ clarifications of their scores indicated sentiments among participants that Pyreal was straightforward to use and generated useful information about model predictions. For example, P16 reported that *“Pyreal seems like a versatile and user-friendly library for generating explanations of ML model predictions. It provides useful visualization tools and integrates well with scikit-learn pipelines, making it a valuable tool for understanding model behavior.”* P9 scored their likelihood of using Pyreal in the future as 5 because *“it is very easy to use and gives a quick and comprehensible contribution explanation.”* P5 stated that *“the presentation and usefulness of the results are well worthy the five minutes it could take to use [Pyreal].”* P17 reported that they *“... would genuinely really look to use it - simply because of how efficient it is & how fast [they] can get data off the ground with minimum programming required.”*

We further investigated participants who scored either metric at 3 or lower. Participants reported scoring Pyreal’s ease-of-use lower because they had not spent enough time with Pyreal or its documentation to commit a higher score. P4 scored Pyreal’s ease-of-use at 2 because *“I am not quite familiar with Pyreal yet but the [sample code] helped a lot and the visualization of data made it easier to understand the data”*, suggesting they may increase their score with more opportunity to use the library. Similarly, P6 scored ease-of-use as 3 and reported a need for more explanation of what the code specifically did (*“I think the sample code is very good to learn but it needs more comments to make it easy to understand, even so I understood Pyreal only by reading it...”*). P7 scored ease-of-use at 3, but primarily reported difficulties with the testing notebook environment (outside of the scope of Pyreal).

Participants who scored their likelihood of using Pyreal in the future as neutral (3 out of 5) reported either a general lack of comfort with Python or a desire to get more experience with Pyreal before committing a higher score. P1 said, *“Maybe I would use it, but I don’t really use python so I’m not really encouraged, but if I did use python often, I would probably*

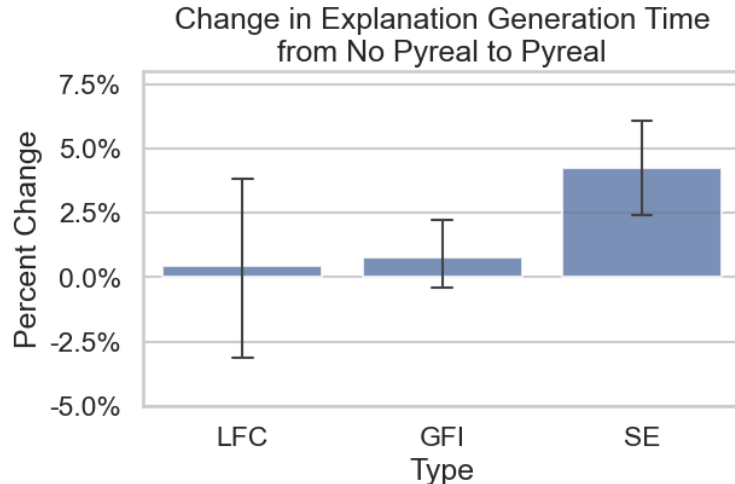


Figure 5.11: Average percent change in explanation generation time for generating an interpretable explanation with Pyreal (Pyreal) compared to generating the same explanation without Pyreal (no-Pyreal-interpret). Pyreal incurred at most an 8.64% increase in runtime. LFC: local feature contributions. GFI: global feature importance. SE: similar examples.

*change my score to 4 or 5*". P4 and P6 both scored 3 and explained that they would want more experience to become comfortable with the library.

Overall, our developer study suggests that Pyreal makes the process of generating interpretable explanations faster and less error-prone, and developers who use it expressed a desire to use it again in the future.

### 5.5.4 Evaluating the Runtime Performance of Explanation Generation

We evaluated the runtime performance of Pyreal and interpretable explanation generation in general to determine 1) the extent to which generating interpretable explanations is slower than the default, model-space explanations and 2) whether Pyreal has any inefficiencies that cause a slower runtime than would occur from generating the same explanations without Pyreal.

#### Method

In these evaluations, we consider the runtime of four tasks. We generate interpretable explanations directly using wrapped libraries (no-Pyreal-interpret), using Pyreal (Pyreal), and using Pyreal with additional output formatting for usability (pyreal-format). We also generate explanations in the model-ready feature space, the default explanations from wrapped libraries (no-pyreal).

Our evaluation used three explanation types: local feature contributions (LFC) using Tree SHAP or Linear SHAP depending on the dataset, global feature importance (GFI) using `sklearn`'s permutation feature importance, and similar examples (SE) using `sklearn`'s `KDTree` module [75]

We used all three evaluation datasets and made 10,000 and 20,000 row versions of each. In other words, we generated explanations on a 10,000 row testing dataset using a 10,000 row training dataset (both datasets were randomly sampled from the original dataset). We then repeated the process with 20,000 row testing and training datasets<sup>7</sup>.

For each experiment (task/explanation-type/dataset/data size), we produced the explanation 10 times and computed the average runtime.

## Results

The increase in runtime when generating an interpretable explanation compared to a model-space explanation varies depending on the transforms used and the runtime of the base explanation algorithm. Our highest percent change from no-pyreal to Pyreal was 86.86% on the Ames Housing dataset (10,000 rows) for LFC. Our explanation transforms in this case ran in linear time, while the base explanation algorithm (Linear SHAP) runs in constant time, resulting in a relatively higher percent increase in runtime. In contrast, we saw almost no change in runtime for the GFI tasks. This was expected, as the permutation feature importance algorithm’s runtime scales with respect to both number of features and number of rows, while explanation transforms on GFI explanations have a constant runtime, as these explanations include only one row of data. We saw a slight increase in runtime for SE explanations, as we needed to transform the dataset to the interpretable space to return interpretable examples.

Pyreal generates interpretable explanations in a similar amount of time as no-Pyreal approaches, with the percent change in generation time between no-Pyreal-interpret and Pyreal ranging from  $-7.74\%$  to  $8.64\%$ . Figure 5.11 shows this result.

For GFI and SE, there was little increase in runtime from formatting. For LFC, formatting introduced a linear-time process (converting a DataFrame to a dictionary of DataFrames per row being explained) to a constant-time explanation algorithm (LinearSHAP), so we saw higher slowdowns of up to 106% per 1,000 rows of data. This formatting will generally not be run on more data than a human can process, so real time increases should be small (up to about .4 seconds per 1,000 rows on a AMD 5800x processor).

## 5.6 Discussion

Our evaluations suggest that Pyreal creates interpretable explanations that are perceived by participants without ML experience to be more interpretable and less confusing than no-Pyreal equivalents. Answers to free-response questions suggested that participants are more able to identify potential correct and incorrect elements of the model’s logic with Pyreal explanations.

Past work has suggested that individuals with ML expertise “associate mathematical representations with logic and intelligence [11]”, which may help explain why the high-data-expertise group showed relatively less preference for our interpretable explanations. This

---

<sup>7</sup>For permutation feature importance on the Ames Housing dataset, we used smaller datasets of 1,000 and 2,000 rows, as this algorithm is very slow on datasets with many columns



further highlights the importance of considering the specific needs of the target audience in explanations.

Our developer study suggests that Pyreal is fast and easy to use, and significantly reduces the likelihood of error when generating interpretable explanations. Notably most errors made by developers when generating interpretable explanations without Pyreal resulted in explanations that seemed reasonable, but were not accurate SHAP values for the model. Such errors are especially dangerous because they are difficult to identify, and therefore a system that prevents them is valuable.

We are continuing to develop Pyreal’s library implementation, adding additional explanation algorithms, types, and transformers. Each added component increases the scope of domains where Pyreal can be useful, opening up new domains for real-world use-cases and evaluations. Future work may also expand Pyreal to other data modalities such as image and text.



# Chapter 6

## Sibyl: A System for Usable ML

Developing a usable ML tool for child welfare screening required more than six months of development time, along with extensive effort from a team of UI and UX designers, front-end developers, ML developers, and ML contributors. In addition, several months were spent evaluating and iterating on the ML tool, which uncovered suboptimal design choices that needed to be adapted and fixed.

Developing ML tools for use in real-world decision problems can be slow and inefficient for several reasons. First, decision processes are complicated and diverse; therefore, ML tools may need to support a wide range of inputs and configurations, while generating many different kinds of outputs that support a wide range of interactions.

Second, existing systems for usable ML tend to follow a *horizontal* structure, meaning they focus on a specific component of the ML tool development process — be it running models, generating explanations, or creating interfaces. Glue code is required to connect these different horizontal components, leading to a cumbersome and slow process.

Third, the intricate nature of real-world decision problems means that input from decision-makers or bridges is essential for tuning and configuring ML tools. Current processes for developing ML tools do not always enable these groups to contribute smoothly. Generally, they must interface extensively through ML developers to make even minor adjustments, leading to many rounds of requirement gathering and iterating on feedback.

Finally, the many complex steps involved in the current method for customizing ML tools for each new decision problem mean there are many opportunities for errors to be introduced. Errors in ML explanations are especially dangerous because 1) explanations are often difficult or impossible to validate in a strictly quantifiable way; i.e. “unit testing” for explanations can be difficult and 2) explanations themselves exist in part to verify a lack of errors.

We sought to address all these issues with Sibyl, our system for developing usable ML tools. The goal of Sibyl is to enable faster and more efficient development of ML tools tuned for individual decision problems.

We designed Sibyl with four key properties in mind:

1. **Configurability.** Sibyl can be applied to a wide range of decision problems quickly and effectively. Those using Sibyl to develop a ML tool can tune across a wide range of configurations, ranging from the types of information shown, the format the information takes, the terms used, and other visualization details.

2. **Comprehensiveness.** Sibyl is designed to support the wide range of interactions required across diverse decision processes using ML outputs. It not only generates interpretable ML explanations, but enables supporting interactions such as updating feature descriptions and values, annotating predictions, filtering features and entities, etc.
3. **Lightweightness.** New ML tools can be set up with Sibyl in a low-effort manner, including by people without expertise in coding or ML. To the extent possible, configurations are optional to allow users to set up basic ML tools quickly.
4. **Extensibility.** Future ML contributors can continue to build on Sibyl, adding support for new configurations and interactions. Because new research is constantly being done on usable and explainable ML, and a single team of ML contributors can never create an entirely comprehensive system, we developed Sibyl to be easily extendable.

In order to achieve these four properties, Sibyl is a vertical system that includes multiple components. Each component can stand alone or be used with the others, or potentially alongside other software components. The Sibyl components are as follows:

1. **Pyreal**, (described in Chapter 5), is a system with a corresponding Python library implementation that handles all transformations required to generate interpretable ML explanations and other augmentations.
2. **Sibyl-API** (described in Section 6.2.1) is a REST-API wrapper around Pyreal’s functionalities that enables all common interactions in the usable ML decision process.
3. **Sibylapp** (described in Section 6.2.2) is a configurable front-end for ML tools.

We begin in Section 6.1 by listing out the requirements for a system for usable ML; in other words, the inputs, outputs, configurations, and interactions that must be considered by a system that seeks to create usable ML tools for diverse domains.

These elements include the inputs to and outputs from the system, along with configurations that may be required when making a new ML tool, and the interactions users (generally decision-makers and bridges) may wish to have with them. In Section 6.2, we discuss the Sibyl system itself and how it supports these elements. Finally, in Section 6.3, we go through a few case studies, applying Sibyl to different real-world applications.

## 6.1 Requirements of a System for Usable ML

In this section, we formally list the elements of a usable ML system, including inputs to the system, outputs from the system, interactions a user may have with the system, and configurations that should be supported by the system. We note that there are different ways to describe and organize this content, and propose one possible organization and set of terms, currently used by Sibyl.

### 6.1.1 Inputs

We begin by discussing the full range of inputs our system supports. Our objective here is to formalize our terminology and define the scope of our system.

**Features** are individual pieces of information that are inputted to the ML model. For example, the square footage of a house or the neighborhood it’s in are features. All features have a **feature type** — Sibyl focuses on tabular data and therefore supports **numeric**, **categorical**, and **Boolean** features. Features may have **feature descriptions**, which are readable versions of their possibly confusing default names. Boolean features may also have **negated descriptions**, which direct how the feature should be described when its value is False (ie. if the feature description is “House has central heat”, the negated description would be “House does not have central heat”). Features may be categorized into **feature categories**; for example, the size of the lot, the size of the first floor, and the size of the second floor may all be part of a **size** category.

**Entities** are the primary units of data input — the concepts, places, things, or persons that the ML model makes predictions about. For example, for a model that predicts house prices, the entities would be houses. In the Sibyl system, entities are identified by unique **EIDs** (entity IDs). Each entity may have one or more **observations**, or a single row of data (as an example of a scenario where entities may have multiple observations, a single house may have a set of feature values and a predicted price at multiple time points). Each observation has a set of **feature values**, or one value per feature as described above, and may possibly have a **label**, which is the ground truth of the variable the ML model outputs for this observation, if known. **Training sets** are the set of observations that include ground-truth labels, which are used for fitting constructs such as explainers or when visualizing historic cases.

**ML models** are objects that take in feature values and return some output. Often, these models require features to be given to them in a very specific format — **transformers** are objects that convert data from one format to another.

**Context configurations** are a broad category of inputs that describe configurations related to the decision problem for which the ML tool is being used — for example, laying out the correct terminology (ie. calling entities "houses"), formatting preferences (ie. formatting the model outputs as rounded US dollar amounts), and the sentiment of model outputs (ie. higher output values are “good”). A full list of configurations is in the next section.

We note that, to support our goal of being lightweight, Sibyl *requires* as few of these inputs as possible. At the same time, in keeping with our goals of being configurable and comprehensive, Sibyl *supports* as many inputs as possible. In particular, only a table of entities and their feature values, a list of feature names, and an ML model and any required transformers are required to initially set up Sibyl.

### 6.1.2 Outputs

Outputs include anything produced or returned by the ML system. Again, we seek to formally define our terminology and explain the full scope of activities we would like to support with Sibyl.

The ML models produce **ML outputs** (also called predictions) based on input data.

Some models, such as classification models, may also provide **output probabilities**, or the likelihood that some output matches the ground truth, according to the model. Additionally, some models may provide **output uncertainty metrics**, or the level of confidence the model has in its output predictions. As defined earlier, **ML augmentations** comprise any other information alongside the ML outputs that helps decision-makers use and understand these outputs. **ML explanations** are a type of augmentation. **Local ML explanations** explain why an ML model gave a specific output, and **global ML explanations** explain how the model predicts outputs in general. **Interpretable ML explanations** are explanations that have been specially tuned to use features that are easy for the target audience to understand (see previous chapter). **Performance metrics** describe how accurate or well-performing the model is expected to be overall. **Dataset visualizations** show detailed information about historic or current data.

**ML tools** are user interfaces (such as a graphical user interface or code notebook) that include ML outputs along with one or more augmentations that aid in the use of these outputs for decision-making. Bridges and ML Developers work together to set up ML tools using systems for usable ML like Sibyl.

### 6.1.3 Interactions

A comprehensive system for usable ML should be developed to support a wide range of interactions users may need ML tools to support. An extensible system should seamlessly support the addition of new interactions that may not have been considered originally.

For the purposes of this thesis, we consider five categories of interactions, described here. This list was compiled from real-world discussions with decision-makers and bridges who use ML outputs in their jobs, and observations of real-world decision processes that involve ML outputs. These categories cover the applications discussed in previous chapters such as child welfare screening, satellite monitoring, and electric health record analysis. The complete list of interactions we considered are shown in Table 6.1.

1. **Getting Information:** Users of ML tools should be have direct access to any system inputs they need to complete their work, including information about entities and their observation, information about features, and information about the ML model.
2. **Model Use, Vetting, and Understanding:** Users should be able to use and understand the ML model, which includes being able to easily compute ML outputs on new observations and getting access to any ML explanations or other augmentations they may need in order to calibrate their trust, use the ML outputs effectively, and respond when they disagree with those outputs.
3. **Logging and Annotating:** Users must be able to contribute their own opinions, expertise, analyses, and other inputs to the ML tool. We seek to support annotations of ML outputs, as well as enabling users to record detailed information about their actions in the past, which may contribute to decisions made in the future.

Table 6.1: List of interactions users have with ML that we aim to support with Sibyl.

<b>Getting Information</b>	
<b>I1</b>	Get general information about an entity.
<b>I2</b>	Explore, search, and filter feature values on entity observations.
<b>I3</b>	Get general information about features (feature types, collection method, etc.).
<b>I4</b>	Get general information about the model (model type, etc.).
<b>Model Use, Vetting, and Understanding</b>	
<b>I5</b>	Get model prediction(s) on an entity’s observations.
<b>I6</b>	Get explanations of an ML model prediction (local explanations).
<b>I7</b>	Get explanations of the ML model’s logic overall (global explanations).
<b>I8</b>	See other cases similar to the current entity/observation, corresponding actions taken, and their outcomes.
<b>I9</b>	Experiment with modifying observations to see how the model’s prediction would change.
<b>I10</b>	Compare observations and entities.
<b>I11</b>	Obtain performance metrics for the model.
<b>I12</b>	Visualize or understand the training data used for the model, such as feature distributions.
<b>I13</b>	See historic model predictions and outcomes.
<b>I14</b>	Understand model uncertainty for a given prediction.
<b>Logging and Annotating</b>	
<b>I15</b>	Annotate model predictions and explanations with external info or insights.
<b>I16</b>	Flag predictions or explanations with categories such as helpful, accurate, notable, or suspect.
<b>I17</b>	Record actions taken based on model output, and the resulting outcomes.
<b>Updating</b>	
<b>I18</b>	Add new models and model pipelines.
<b>I19</b>	Update models with new training data.
<b>I20</b>	Add or modify entities.
<b>I21</b>	Add labels to entities once the ground truth is known.
<b>I22</b>	Update feature list to support models that take in new information.
<b>Tuning</b>	
<b>I23</b>	Modify feature descriptions and categories.
<b>I24</b>	Select which explanation types and augmenting information to display.
<b>I25</b>	Modify visual settings of pages (such as color and size of components).
<b>I26</b>	Modify the size of the dataset used based on computational limitations.
<b>I27</b>	Modify terms and formatting used.

4. **Updating:** As entities, features, models, and other inputs to the system change, users should be able to easily update the ML tool to reflect these changes without having to modify anything from scratch.
5. **Tuning:** Users should be able to easily modify feature descriptions and other terms used to meet their preferences.

In the next section, we explain how Sibyl enables these interactions through its REST-API layer.

### 6.1.4 Configurations

Studying the literature and undertaking our own case studies has allowed us to identify the many configurations that a fully **configurable** usable-ML system must be able to support. We formally codify these in Table 6.2.

We categorize configurations based on what information they affect:

1. **Feature Configurations** affect how features are presented and how different features should be interpreted. These configurations include concepts introduced in previous chapters, such as the feature space used and the subjective properties of features, as well as feature descriptions and categorizations.
2. **ML Output Configurations** affect how information related to the ML output is presented. For example, allowing users to specify the sentiment of the ML output can help determine the colors associated with positive and negative feature contributions (e.g. “bad” contributions should be red).
3. **Entity Configurations** affect how information related to entities and their observations is organized.
4. **Explanation Configurations** affect which ML explanations and augmentations are displayed. Note that explanations are also impacted by other configuration categories, like feature configurations.
5. **Interface Configurations** include more general configurations for information presentation, such as what terms, colors, and sizes to use.

Throughout the next section, we introduce the various ways different Sibyl components enable configurations. When using the full Sibyl system, most configurations are handled on the Sibyl-API layer, so that users can switch between different front-ends without having to reconfigure every time. Some user-specific configurations, like colorblindness support settings and component sizes, may be handled on the interface level, such as by Sibylapp. The feature space used, as discussed in the previous chapter, is configured when creating a Pyreal `RealApp` object.



Table 6.2: Configurations to be considered when developing ML tools for new domains.

<b>Feature Configurations</b>	
<b>C1</b>	Feature space used (ex. avoid one-hot encodings, avoid standardization) [76]
<b>C2</b>	Properties of features (interpretability, actionability, etc.) [77], [78]
<b>C3</b>	Human-readable feature descriptions (ex. MedInc $\rightarrow$ Median Income) [77]
<b>C4</b>	Categorizations of features (ex. size category includes first floor size, second floor size, and lot size)
<b>ML Output Configurations</b>	
<b>C5</b>	Model prediction formatting (ex. 0/1 $\rightarrow$ normal/failure, 3456 $\rightarrow$ \$3,456)
<b>C6</b>	Sentiment of model outcome (what predictions are good/bad?) [79]
<b>C7</b>	Whether to show class probabilities [80]
<b>Entity Configurations</b>	
<b>C8</b>	Groupings of entities (ex. houses 1-10 are part of the Placedale neighborhood)
<b>C9</b>	Data organization (are there multiple observations per entity? are these observations organized by time?)
<b>Explanation Configurations</b>	
<b>C10</b>	Appropriate explanation types (ex. for sensitive data, avoid similar entities. For high-risk, avoid LLM-generated explanations) [12], [20]
<b>C11</b>	Explanation types based on user preference [81]
<b>Interface Configurations</b>	
<b>C12</b>	Terms used for general concepts (ex. features $\rightarrow$ factors, entities $\rightarrow$ houses, positively contributing features $\rightarrow$ risk factors) [12], [79]
<b>C13</b>	Colors used (i.e., for colorblindness accessibility)
<b>C14</b>	Size of components for screen size accessibility
<b>C15</b>	Balancing accuracy versus speed (ex. use fewer training data points on slower computing/lower-risk applications)

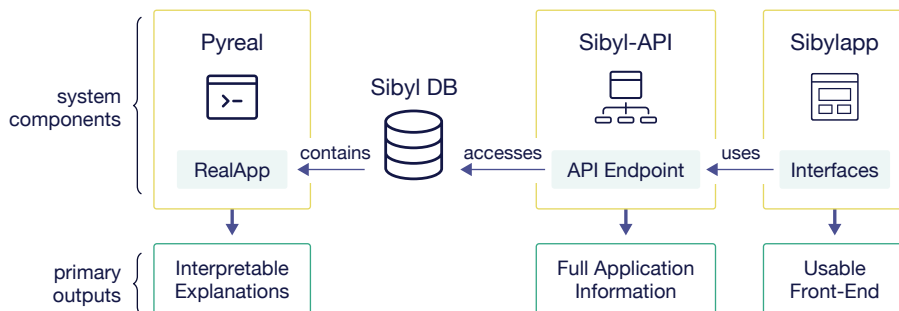


Figure 6.1: Summary of the interactions between components in the Sibyl system.

## 6.2 Sibyl System

In this section, we will discuss the Sibyl system, and how it supports the requirements introduced in the previous section. We already discussed Pyreal in the previous chapter — here, we introduce Sibyl-API and its corresponding database structure, and Sibylapp, our front-end. For each of these, we discuss how they enable the interactions and configurations described in Section 6.1.

The full Sibyl system is summarized in Figure 6.1. The Sibyl database stores pre-trained Pyreal `RealApp` objects (introduced in the previous chapter) to generate interpretable explanations. Sibyl-API endpoints access the database, and use `RealApp` objects to generate explanations. Sibylapp accesses these endpoints.

### 6.2.1 Sibyl-API

Sibyl-API is a general-use back-end Representational State Transfer (REST) API that enables the creation of ML tools, allowing people to apply usable and explainable ML to diverse decision problems without needing to re-implement explanation algorithms or write code in any specific language.

#### Motivation for a Usable ML REST-API

Sibyl-API offers all the standard benefits of a REST API layer. It supports Sibyl’s lightweightness as it conforms to the HTTP standard and therefore can be incorporated into a range of systems. REST APIs are frameworks that enable stateless communication, meaning they allow tasks to be completed without any server-side saved information. Specifically, Sibyl-API maintains all state information in the Sibyl database, and client-side calls to the server using Sibyl-API include all information needed to complete tasks using this database. This keeps the API modular and flexible to changes to other parts of the system. It allows for easy integration of usable ML between different interfaces and other output systems, including our GUI Sibylapp along with other external visualization tools.

For usable ML specifically, a REST-API layer offers significant flexibility and configurability in interface design. As discussed in the next section, we developed our front-end interface Sibylapp to act as a general-use, configurable front-end for a wide variety of decision

Table 6.3: List of Sibyl-API routes, with the interactions they seek to enable (see Table 6.1) [GET endpoints](#), [PUT endpoints](#), [POST endpoints](#).

API Route	Interactions
entities/	<a href="#">I1</a> , <a href="#">I2</a> , <a href="#">I20</a> , <a href="#">I21</a>
features/	<a href="#">I3</a> , <a href="#">I22</a> , <a href="#">I23</a>
dataset/	<a href="#">I12</a> , <a href="#">I13</a> , <a href="#">I19</a>
models/	<a href="#">I4</a> , <a href="#">I11</a> , <a href="#">I18</a>
models/importance/	<a href="#">I7</a>
models/prediction/	<a href="#">I5</a>
models/uncertainty	<a href="#">I14</a>
contexts/	<a href="#">I24</a> , <a href="#">I25</a> , <a href="#">I26</a> , <a href="#">I27</a>
explain/contributions	<a href="#">I6</a>
explain/modified_prediction	<a href="#">I9</a>
explain/similar_entities	<a href="#">I8</a>
explain/compare	<a href="#">I10</a>
log/	<a href="#">I15</a> , <a href="#">I16</a> , <a href="#">I17</a>

problems; however, with the massive diversity of real-world requirements, many applications require more specific interfaces. Additionally, real-world constraints may mean that a specific front-end platform that is not Python or Streamlit-based is required. Creating a separate REST-API layer allows Sibyl to contribute even in domains with such constraints.

## Sibyl-API Endpoints

Sibyl-API endpoints take in parameters from the client (front-end application) to provide information supporting all the interactions described in Table 6.1. ML developers using the API can specify entities by EIDs, and models, features, and contexts by name. They can get information and use and understand the model through GET endpoints, which extract relevant information from the endpoint and may use Pyreal to run computations. Logging and annotating interactions are mostly handled by logging PUT endpoints. Finally, developers can tune and update information through the relevant POST endpoints.

Sibyl-API endpoints are categorized into several routes:

1. **Entity** endpoints provide information about entities and the observations associated with them.
2. **Feature** endpoints provide information about input features, such as their readable descriptions, types, and categorizations.
3. **Model** endpoints provide ML outputs using the ML model, and other information about the model such as feature importance scores.
4. **Context** endpoints provide context-specific information for setting up ML tools. All terminology used can be given a domain-specific alternative — for example, entities

could be “houses”, “turbines”, “customers”, etc., and features could instead be referred to as “factors” or “properties”. Additionally, these endpoints provide information such as how to format ML outputs.

5. **Explain** endpoints provide ML explanations and other augmentations. The back-end code for these endpoints usually uses the saved `RealApp` object.
6. **Log** endpoints log user interactions with ML tools.

The complete list of interactions supported by each of these endpoints is shown in Table 6.3. Some endpoints support multiple interactions by providing multiple types of data or taking in optional parameters.

## Sibyl Database

Sibyl-API accesses and updates information from a database, which is populated using one of several methods (as discussed in detail in Section 6.2.1). Our implementation of the Sibyl database is built using MongoDB, and is based on the schema shown in Figure 6.2.

## Implementation: Setting Up Sibyl-API Servers

In this section, we go through details of how Sibyl-API servers are set up and configured for new applications. Sibyl-API is implemented in Python and Flask<sup>1</sup>, and Sibyl DB is implemented in MongoDB<sup>2</sup>.

We have developed three approaches to configuring a new Sibyl-API server, each requiring decreasing levels of development expertise at the cost of decreased flexibility. These are a full-Python approach, a configuration file approach, and a GUI-based setup wizard.

**Option 1: Full Python Configuration** A Sibyl-API server can be fully configured using a low-code Python API. This is the most flexible configuration option for ML developers who are comfortable with coding and have requirements for complex transforms.

In this option, the developer prepares the Sibyl-API database and sets context configurations using the Sibyl-API database utility library functions. Data can be passed into these functions as Python objects such as `pandas` DataFrames and dictionaries, or as references to files such as CSVs that have the data saved.

**Option 2: Configuration File** Users preparing Sibyl-API servers can also use YAML configuration files by running the Sibyl-API command line utility. The configuration files specify the location of data CSVs and serialized `RealApp` objects, as well as information about context configurations.

Figure 6.8 shows a snippet of these configuration files for a sample application.

**Option 3: Setup Wizard** Users with less coding experience can also opt to use the Sibyl-API setup wizard, which takes the form of a graphical user interface (GUI). This option offers a fast and easy setup approach. Users run the setup wizard, and are given a step-by-step series of questions to answer about their requirements. This process generates the

---

<sup>1</sup><https://flask.palletsprojects.com/>

<sup>2</sup><https://www.mongodb.com/>

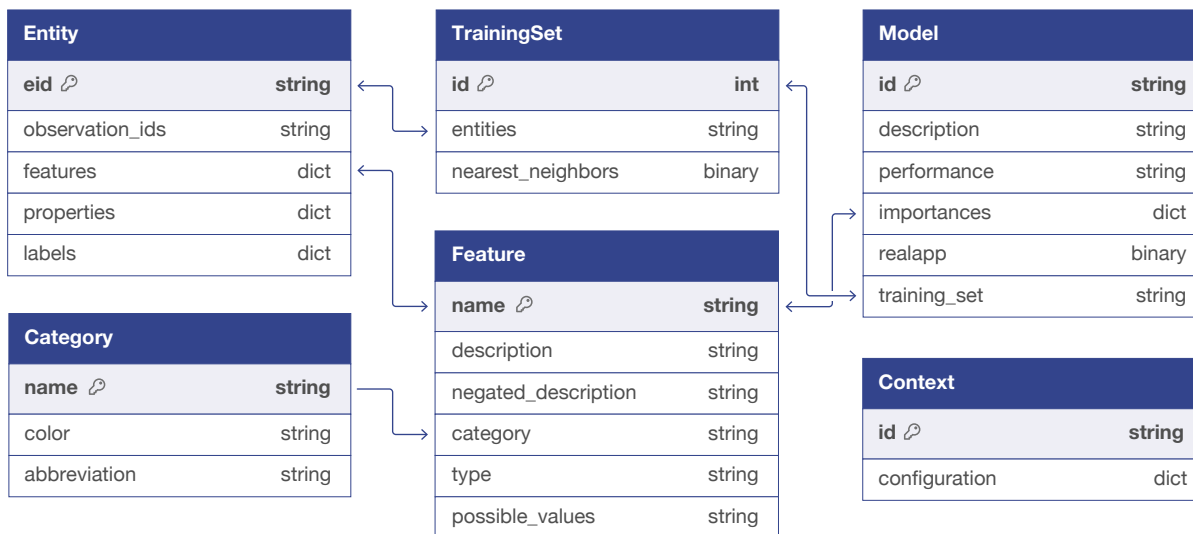


Figure 6.2: Entity relational diagram for the Sibyl database. Entity documents include information about entities, including their observations and feature values, ground-truth labels (if available), and other properties. Feature documents include feature configurations, including feature type (Boolean, categorical, or numeric), readable descriptions and negated descriptions (for Boolean features), category, and possible values for categorical features. Category documents keep track of configurations related to feature categories. Model documents include serialized, pre-fit `RealApp` objects, descriptions of the model and its performance, and pre-computed feature importance scores. TrainingSet documents contain a list of entities with ground-truth labels that should be used for computations, including a prefit list of nearest neighbors. Finally, Context documents describe configurations for a specific application context.

configuration files automatically, and then prepares the database. Figure 6.5 shows a sample from this setup wizard.

Having gone through our structure for Sibyl-API, we now move to our front-end interface system, which uses a Sibyl-API server to enable a wide variety of user interactions.

## 6.2.2 Sibylapp

Sibylapp is the lightweight, highly configurable front-end component of Sibyl. Sibylapp offers many different explanation interfaces that form ML tools.

Sibylapp is fully implemented in Streamlit [82], a pure-Python, easy-to-develop framework for generating user interfaces. In the event that configurability is required beyond what we offer, users can easily use Streamlit to further adjust their interfaces.

Sibylapp accesses Sibyl-API endpoints to get information and context configurations. Therefore, no further inputs are required to set up an GUI once a Sibyl-API endpoints have been established.

Sibylapp interfaces allow users to use, understand, and interact with their models. Appendix C.1 includes screenshots of all Sibylapp interfaces implemented at the time of writing.

Table 6.4: List of Sibylapp interfaces currently supported, with the base augmentation or explanation type they focus on showing, and the interactions they enable (see Table 6.1). Note that Updating interactions are currently handled when setting up the Sibyl-API server (using configuration methods like the Setup Wizard makes it possible to do this without writing code). Logging and Annotating interactions are not yet supported by Sibylapp.

<b>Interface</b>	<b>Base Augmentation</b>	<b>Interactions</b>
Prediction Summary	ML Outputs	<b>I5</b>
Explore a Prediction	Feature Contributions	<b>I1, I2, I6</b>
Similar Entities	Nearest Neighbors	<b>I8</b>
Compare Observations	Feature Contributions	<b>I10</b>
Experiment with Changes	Counterfactuals	<b>I9</b>
Change over Time	Feature Contributions	<b>I6</b>
Understand the Model	Feature Importance/Contributions	<b>I4, I7, I11, I12</b>
Edit Features	N/A	<b>I23</b>
Settings	N/A	<b>I24, I25, I26, I27</b>

The next chapter goes through several of these interfaces in depth, as applied to one specific sample application. These interfaces currently support most of the interactions shown in Table 6.1 through calling the appropriate Sibyl-API endpoint. Table 6.4 shows the current list of interfaces we have developed, along with the interactions they support.

Sibylapp includes a Settings page that supports certain individual-level interface configurations (C13-15).

## 6.3 Case Studies

To evaluate Sibyl’s configurability, we walk through case studies demonstrating the process of using Sibyl to set up ML tools for three applications. In the next chapter, we explore a fourth domain, bringing in detailed user feedback.

### Sibyl for Child Welfare

In Chapter 3, we described how we set up a usable-ML tool for child welfare screening. At the time of that case study, we used an older, less generalizable and less lightweight version of Sibyl based on React, which required setting up the interfaces mostly from scratch. Here, we discuss the steps required to set up a ML tool using our new Sibyl system.

**User Persona:** Our decision-maker audience is described in depth in Chapter 3; here we offer a brief reminder. The child welfare screeners we work with have expertise in child welfare, and generally no expertise in ML or data science. Their objective is to decide whether to screen in (investigate further) or screen out (do not investigate further) a case of reported child abuse. To aid with this process, they use a 1-20 risk score output by an ML model. In addition to receiving this score, they are interested in understanding relevant factors about the case that they may have missed.

**Inputs:** The ML model used is a Linear Regression model trained with the Lasso optimizer on around 400 features. It outputs the probability that a child will be removed from their home in the next two years, conditioned on them being investigated. Categorical features were one-hot encoded; otherwise, no further data transforms were used. We set up a RealApp for this case using the code shown in Figure 6.3. We created an `entities.csv` document including the feature values for a set of historic data. We then created a `features.csv` document that includes the feature descriptions corresponding to all unreadable feature names, and the feature types. We categorized the features into 13 categories, such as `placement history`, `referral history`, and `demographics`. For Boolean features, we manually wrote out negated versions of their descriptions (for example, "adult in case has open child welfare case" → "adult in case does not have open child welfare case"). Finally, we created a context config YAML with specifications such as labeling the output sentiment as negative, and using terms like “Factor”, “Risk”, and “Protective” rather than “Feature” “Positively contributing” and “Negatively contributing.” This context config also specifies the binning thresholds that will convert the ML output of probabilities to 1-20 risk scores when displaying the ML outputs in Sibylapp. Finally, this context config lists the desired interfaces to use — for example, it specifies to not include the Similar Cases interface, as our past work demonstrated this was not appropriate for child welfare screening.

**Setting up Sibyl:** With the inputs described above, we can set up the database and run Sibyl-API for child welfare using the Python configuration method and the code shown in Figure 6.3. From there, we run Sibylapp using the command line utility, and all configurations are handled. Use of the ML tool remains similar to the original described in Chapter 3, though with our new configurations we are able to automatically ensure features remain fully human-worded, as shown in Figure 6.4.

## Sibyl for House Price Prediction

Next, we applied Sibyl to the decision problem of real estate pricing using the Ames Housing Dataset [72], as used in examples throughout this thesis. This dataset includes information on almost 3,000 houses sold in Ames, Iowa from 2006 to 2010. For each house, the dataset includes the price it sold for, along with 79 features related to house size, location, utilities, construction date and more.

**User Persona:** For this application, we are taking on the persona of a real estate agent looking to help homebuyers put in a winning bid on a house in Ames in 2011. The house in question is newly constructed, and the agent wants to use ML — not just to get an idea of what price the house will sell for, but also to better understand where this price is coming from, so the agent can add this information to their own domain expertise and use it to get the best deal for their clients.

**Inputs:** We trained a Ridge regression model to predict the price of houses in this dataset. We used Pyreal to set up a RealApp that one-hot encoded categorical features, imputed missing data using an understanding of the data to select the strategy (for example, houses with missing basement sizes usually do not have basements, so those values were imputed with 0s), and standardized all resulting numeric features. We saved the serialized RealApp in a file called `realapp.pkl`.

```

1  from pyreal.transformers import OneHotEncoder, fit_transformers
2  from sibyl import db
3  from sibyl import run
4
5
6  X_train = ...          # Load in training data
7  model = ...           # Load in Linear Regression model
8  transformers = OneHotEncoder(columns="all_categorical")
9  fit_transformers(transformers, X_train)
10 realapp = RealApp(model, transformers=transformers, id_column="eid")
11
12 db.connect_to_db("family-screening")
13 db.insert_features_from_csv("features.csv")
14 db.insert_entities_from_csv("entities.csv", label_column="PRO_PLSM_NEXT730_DUMMY")
15 db.insert_model_from_object(realapp)
16 db.insert_context_from_yaml("context_config.yml")
17 db.disconnect_from_db()
18
19 sibyl.run(db="family-screening")

```

Figure 6.3: Sample configuration code to set Sibyl up for child welfare screening using the Python configuration approach.

We then saved the original dataset in a file called `entities.csv`, and created a `features.csv` file that included descriptive versions of all feature names, as well as labels for the feature types and categories.

**Setting up Sibyl:** For this application, we set up the Sibyl server using the setup wizard, which walks the user through uploading their data and selecting context configurations. Figure 6.5 shows a screenshot from the context-configuration section of this process. In this case, we do not need any interfaces that explain over time, as we are looking at house prices for a single moment in time. Our user is interested in receiving local explanations of the house at hand, as well as in developing an understanding of the overall housing market with global explanations.

Sort by  Show numeric contributions? ⓘ

Absolute
  Risk
  Protective
  Side-by-side

Decrease in risk
  Increase in risk

Category	Factor	Value	Contribution
demographics	Age range of child in focus	<1 year	■ ■ ■ ■ ■ ■ ■ ■ ↑
roles	Number of individuals with the role of parent associated with the referral	3+	↓ □ □ □ □ ■ ■ ■ ■
demographics	Counts of the number of parents that are 20<=age<25	1	↓ □ □ □ □ □ ■ ■ ■

Figure 6.4: Screenshot from Sibyl applied to child welfare, demonstrating human-worded features and domain-specific terminology.



## Configure Context

Does your data have multiple rows?

No  Yes

What is your model's output type?

Numeric  Boolean  Categorical

How should we format the output?

1,234  1,234.56  \$1,234  \$1,234.56  No Formatting  Custom

Does an increasing model prediction refer to a positive or negative outcome?

Positive  Negative  Neither

Select here to modify terms ▼

Select pages to enable:

Prediction Summary

Explore a Prediction

Similar Entities

Compare Entities

Experiment with Changes

Change over Time

Understand the Model

Edit Features

Settings

Download config

Should users be able to enable additional pages?

No  Yes

Figure 6.5: Snippet of setup wizard for a house pricing application, demonstrating how users can specify the application context configurations.

Search and filter ^

Search by factor

square feet

Filter by category

size ×
× ▾

Sort by  Show numeric contributions? ?

Absolute  
  Beneficial  
  Detrimental  
  Side-by-side

Decrease in sale price  
  Increase in sale price

Category	Factor	Value	Contribution	☰ Show feature plot?
size	Second floor square feet	752	<span style="display: inline-block; width: 100px; height: 15px; background: linear-gradient(to right, #007bff 75%, #fff 75%);"></span> ↑	<input type="checkbox"/>
size	Above grade (ground) living area square feet	1,774	<span style="display: inline-block; width: 100px; height: 15px; background: linear-gradient(to right, #007bff 75%, #fff 75%);"></span> ↑	<input type="checkbox"/>
size	First Floor square feet	1,022	↓ <span style="display: inline-block; width: 100px; height: 15px; background: linear-gradient(to right, #fff 75%, #e00 75%);"></span>	<input type="checkbox"/>
size	Low quality finished square feet (all floors)	0	↓ <span style="display: inline-block; width: 100px; height: 15px; background: linear-gradient(to right, #fff 100%);"></span>	<input type="checkbox"/>

Figure 6.6: Sample page from Sibylapp for the housing sample application.

**User story:** An example of a Sibylapp interface for this application can be seen in Figure 6.6. We will now walk through a sample story of our user persona (a real estate agent) using Sibyl to complete their decision task (selecting a bid price for their client). They begin by looking at the ML output, which predicts a price of around \$157,000 for the house at hand. They know from their experience that this price is a bit lower than the average house in Ames; they can see from the feature contribution table on the Explore a Prediction page that this lower price comes primarily from the size of the house and the quality of its materials. To better understand how this house size compares to other houses, and how house size influences the model’s price predictions, they select the “Show feature plot” option to see more details. They see that the model’s house-price prediction scales linearly with the square footage, with houses larger than around 1,600 square feet being considered above average — at that threshold, house size starts bringing the house price prediction over the average.

The agent sees that the dataset considers the quality of the house’s building materials to be 5 out of 10, but is aware that renovations are being done that will increase the quality. They use the Experiment with Changes page to see that increasing the building material quality to 6 or 7 would change the predicted house price to around \$164,000-\$171,000, and factor this knowledge into their pricing suggestions.

## Sibyl for national emissions data

Our final case study involves observing correlations between various factors pertaining to countries and their overall CO2 emissions.

**User persona:** For this application, we take on the persona of a climate researcher looking to better understand the factors relating to carbon emissions across nations. Compared to previous applications, this persona is less interested in seeing or understanding individual ML outputs, but is especially interested in understanding the correlations between countries, their factors, and their emissions.

**Inputs:** We gathered information about 239 countries from 1991 to 2024. Data came from the World Bank <sup>3</sup>. For each country, we collected data on CO2 emissions (in metric tons per capita) for each year during the selected time frame, along with 38 other factors, such as the percentage of the populace with access to electricity and the primary industries in the countries.

Similar to previous applications, we placed our data in `entities.csv`. We set the `eid` (entity ID) column to the country names, and the `observation` ID column to the year.

**Setting up Sibyl:** For this application, we use the config YAML method to set up the Sibyl-API server. Figure 6.8 shows the configs used.

In this case, we include interfaces (pages) that allow the climate researcher to understand changes in ML outputs over time. We also focus on interfaces that may offer insights about what contributes to emissions, such as the feature-contribution-based “Explore a Prediction” and feature-importance-based “About the Model” interfaces, rather than interfaces that focus on ML outputs (such as the “Prediction Summary” page). We also include the “Experiment with Pages” page, so that our user persona can investigate possible interventions. Finally, because this use case has a time-based element, we include the Change over Time interface, which allows the researcher to understand how emissions and related factors have shifted over time.

**User Stories:** The climate researcher wants to understand what factors correlate with countries that have high emissions. As such, they begin by focusing on the “Understand the Model” interface, to visualize the overarching trends. As expected, when looking at feature importance scores, they see significant importance given to a feature that tracks the percentage of the population that has access to electricity. More surprisingly, they see that average precipitation per year is the second most important feature, and wonder what may cause this correlation. They visualize this feature in the feature-level plots (Figure 6.7) and notice that this feature actually does not contribute to the ML output for most countries, except for a small number of countries in the dry Middle East, which tend to have higher-than-average predicted emissions (Qatar, UAE, Kuwait, Bahrain, and Saudi Arabia). They decide this may be a spurious correlation, but make a note to look into it further.

## 6.4 Discussion

Developing ML tools using Sibyl for multiple diverse domains reveals the wide range of considerations and difficulties involved in making a truly configurable system.

---

<sup>3</sup>Data available at <https://data.worldbank.org/>

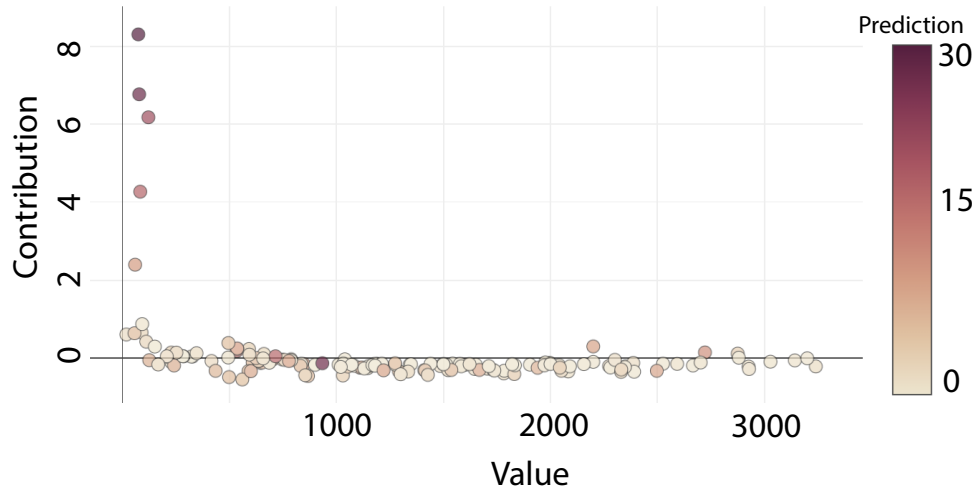


Figure 6.7: Feature-level plot explaining how the ML model uses the **Average Precipitation per Year (cm)** feature for the emissions sample application. Each dot represents one country (hovering over the dot will show the country name). The x-axis shows values for the feature, and the y-axis shows the feature’s contribution to the model’s prediction of total emissions. We can see that a few very dry, high-emitting countries push this feature’s overall contribution up significantly.

Here, we summarize some of the specific challenges we were presented with when moving from a traditional ML mindset, where the goal is to develop a high-performing model, to a usable ML mindset, where the goal is to improve decision-making outcomes using ML outputs.

1. In a more traditional ML mindset, we think of explanations in terms of the information used by the ML model — the closer to the “real” logic of the ML model, the more accurate and better the explanation is. In the usable ML mindset, having an explanation that can be understood by the target decision-makers may be even more important than having one that is maximally true to the model.
2. In the traditional ML mindset, we often think of ML explanations as mostly serving the purpose of verifying the ML outputs. In the usable ML mindset, the explanations may serve many purposes, such as acting as feature flags or providing additional information to guide actions taken by decision-makers.
3. In the traditional ML mindset, we think of features as being positively or negatively contributing to the prediction. In the usable ML mindset, these terms are confusing, as a “positively contributing” feature may have a negative sentiment — such as when it corresponds to an increased risk of child abuse. Terms and colors used on interfaces must reflect the semantic meaning of a feature, not the numeric one.
4. Many ML augmentations that are popular in the traditional ML literature can be inappropriate or harmful for usable ML applications. Attempting to explain a model’s

```
1 # Name of database to create
2 database_name: "emissions"
3 # Name of entity file
4 entity_fn: "entities.csv"
5 # Name of feature file
6 feature_fn: "features.csv"
7 # filename of pickled realapp
8 realapp_fn: "realapp.pkl"
9 # Name of context config file.
10 context_config_fn: "context_config.yml"
11 # column containing labels in entity file
12 label_column: "CO2 emissions (metric tons per capita)"

1 # Model prediction output type, one of "numeric", "boolean", "categorical"
2 output_type: "numeric"
3 # Python f-string for formatting numeric model outputs
4 output_format_string: "{:.3f}"
5 # If false, increase in model predictions correspond to a positive outcome.
6 output_sentiment_is_negative: True
7
8 # Context-specific overrides for common terminology
terms:
10   entity: "Country"
11   feature: "Factor"
12   prediction: "CO2 emissions (metric tons per capita)"
13   positive: "Increasing" # Features with positive contribution
14   negative: "Decreasing" # Features with negative contribution
15
16 pages_to_show:
17   - Explore a Prediction
18   - Change over Time
19   - Experiment with Changes
20   - Understand the Model

1 sibyl prepare-db emissions-config.yml
```

Figure 6.8: Configuration file approach to preparing the emissions Sibyl application. Top: The contents of a configuration YAML file `emissions-config.yml`. Middle: The contents of the context configuration file, `context_config.yml`. Bottom: The command to prepare the application from the config.

logic through similar historic cases may encourage negative decision-making by setting up unfair comparisons between individuals (as in the child welfare case), or may reveal private information. Counterfactuals may suggest giving actionable advice, even when the underlying model is not causal. A usable ML system must support a wide range of augmentations and allow users to easily configure which ones to show.

5. In the traditional ML mindset, more information is helpful. In the usable ML mindset, more information is often overwhelming, and may do more harm than good if it inadvertently reduces user trust in or comfort with the model. Usable ML tools should default to showing the minimum valuable information, and allow users to search and filter down to relevant subsets of information, such as through feature categorizations.
6. The traditional ML literature has a specific and commonly used set of terminology — features, data columns and rows, and model predictions. When developing usable ML tools, the terminology of the domain must be used — for instance, swapping in the word “factors” for “features.”
7. Users need to be able to input their own thoughts and analyses alongside the model’s predictions, and usable ML tools need to support these kinds of interactions. While Sibyl-API supports these inputs, future work should add natural ways to support these interactions on the interface level.

# Chapter 7

## Application 2: Usable ML for Wind Turbine Monitoring

Fully understanding the requirements for usable ML tools and evaluating their impact on decision-making requires studying real-world applications. Toy datasets (however realistic) and formal user studies (however carefully chosen) cannot provide the necessary depth of feedback. Therefore, we decided to apply the lessons we learned and the systems we developed in another real-world context: wind turbine monitoring.

To keep turbines running effectively, operators analyze turbine sensor data — referred to as Supervisory Control and Data Acquisition (SCADA) data — to determine when a potential failure may occur. One type of failure occurs when a turbine brakepad prematurely wears out. This can be prevented by sending technicians up the turbines for investigation and repair — but this is an expensive and potentially dangerous task. A usable ML tool could reduce downtime from such failures while keeping costs low, alerting relevant personnel to potential brakepad failures and providing the information they require to make efficient decisions about brakepad inspection and replacement.

In order to develop an effective ML tool for this decision problem, our team worked in parallel on two tasks: developing the ML model and creating a ML tool with additional explanations and augmentations. In this thesis, we will focus on the latter task after briefly introducing the data and ML model.

Our research questions from this application were:

- RQ1** How does wind turbine monitoring (specifically, turbine brake pad monitoring) fit within the considerations for usable ML we discussed in the introduction? Who are the people and what are the processes involved?
- RQ2** What design choices must be made when developing ML tools for use in wind turbine monitoring?
- RQ3** How do bridges and decision-makers in the domain of wind turbine monitoring interact with ML tools developed using Sibyl?
- RQ4** How do ML tools influence decision-making in the domain of wind turbine monitoring?

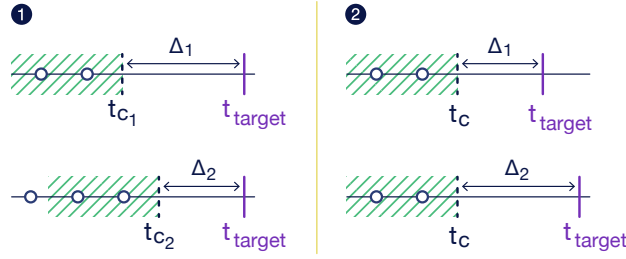


Figure 7.1: Visualization of the interactions between the cut-off time, the target time, and the lead time over two possible prediction scenarios. Figure taken from [84].

**Study Participants.** Over the course of several years, we collaborated with a team of three wind turbine data analysts at Iberdrola, a renewable energy company. All members of this team regularly reviewed and made decisions regarding wind turbine monitoring and maintenance, as described in depth in the next section. Throughout the collaboration, we met weekly with the team to gain additional insights, feedback, and requirements.

## 7.1 Data and Modelling

In this section, we briefly introduce the data and ML model used for this application. The ML model architecture and features used here were more complex than those used in our child welfare application, which allowed us to work through additional considerations relevant to usable ML tool development.

The data used to predict wind turbine brake pad failures was collected based on time series SCADA data using the *label, segment, featurize* approach [83]. For this algorithm, there are three key terms to define:

- The *cut-off time* ( $t_c$ ) represents the latest time in the historic training data used for training or inference; i.e., no data after this time is included in computing model features. [83]
- The *target time* ( $t_{target}$ ) represents the moment in time that the ML output is predicting; i.e., at what time are we predicting whether the brakepad will have failed?
- The *lead time* (**lead**) is length of time between the cut-off and target times ( $t_{target} - t_c$ ), i.e. the length of time into the future we wish to predict for.

In summary, using SCADA data collected up until time  $t_c$ , we predict whether the brakepad will fail in the next **lead** amount of time, at time  $t_{target}$ . The interactions between these three concepts are visualized in Figure 7.1. This figure also demonstrates the two scenarios wind turbine data analysts are interested in: 1) predicting brakepad failures at different times in the future starting from a set time (constant  $t_c$ ), and 2) repeatedly predicting failures at a specific future time point over time (constant  $t_{target}$ ).

The data featurization and labeling for this task was handled by Zephyr [85], an ML system developed specifically for wind turbine prediction tasks. The Zephyr workflow follows four primary steps:



Table 7.1: Considerations for usable ML for wind turbine monitoring.

Process	
<b>Decision problem</b>	Whether a turbine is at risk of brake pad failure and why
<b>Action space</b>	Do nothing, add information to report, make urgent alert
<b>Measured outcome</b>	Overall turbine downtime and maintenance costs
<b>Time to make a decision</b>	Unbounded
<b>Associated risk</b>	High
<b>ML Integration</b>	Proxy (Predicted failures)
Roles	
<b>Decision-maker(s)</b>	Wind turbine engineers (Operations and Maintenance team)
- <i>ML/data expertise</i>	Some
- <i>Domain expertise</i>	Expert
<b>Bridges</b>	Wind turbine data analysts (Data and Analysis team)
<b>ML Developers</b>	ML developers (our team), Data and Analysis team
<b>ML Contributors</b>	Authors
<b>Affected Individuals</b>	Turbine contractors, site teams, energy consumers

1. First, the data is uploaded and formatted into a relational data structure called an `EntitySet` [85]. For turbine brake pad monitoring, this data is primarily SCADA data, which consists of signal data sampled at regular time intervals. This data includes diverse signals such as vibration of components, wind speed, and active power. At each time step, the signal values are aggregated across the selected interval, using several aggregation methods (minimum, maximum, average, and standard deviation) [85].
2. Now that the data has been compiled and formatted, we can generate labeled training data. The labels we seek to generate show whether a turbine was experiencing a brake pad failure at a given time. At various time steps, the data up to the cutoff time is saved for training, and then the label is extracted by looking at the data `lead` time points ahead of the cutoff time; i.e., the label is taken from time  $t_c + \text{lead}$ . [85]
3. Next, the data prior to each cutoff time is featurized. For each training data row, we take some constant number of time steps prior to the cutoff time, specified by a set window size. These time steps are then aggregated using a series of aggregation functions such as minimum, maximum, mean, and standard deviation. [85]. In this case, we ended up with about 300 features total.
4. Finally, given the featurized data and labels, a series of XGBoost classifiers [86] were trained to predict whether a brake pad is likely to fail after a given lead time at a given cutoff time. Each lead time required a separate classifier, trained on the dataset of the specified lead time. For our study, we used lead times of 0-30 days, at 5-day intervals.

With our set of ML models, we are now ready to take a deeper look at our collaborators' requirements and develop a ML tool to help with wind turbine brake pad monitoring decision problem.

## 7.2 Understanding Context and End-User Needs

We began by seeking to answer **RQ1** by understanding the people and processes involved in wind turbine monitoring, to ensure our ML tool would be usable for the decision problem. All findings from this section are summarized in Table 7.1.

We first conducted an hour-long unstructured interview with one of our collaborating wind turbine data analysts at Iberdrola. This allowed us to identify the people involved in or affected by the wind turbine monitoring decision problem, as well as their needs and expertise. We identified several teams, groups, or organizations that may be affected:

1. The **Monitoring and Analysis Team (MA team)** (our primary collaborators) is a team of data analysts that fill both our defined bridge role and the decision-maker role. They analyze data about wind turbines around the world to find anomalies, potential sources of failures, and broad performance trends. They compile monthly reports on the overall turbine fleet, which have information about performance trends, sites that require additional inspection, and more. They also send additional alerts related to urgent situations, such as an elevated risk of turbine failure. A significant part of the MA team’s job involves filtering and prioritizing information and alerts to send further down the pipeline.

This team makes up the primary direct audience for the ML outputs, which highlight potential brake pad failures. The MA team must then filter and prioritize these alerts, and compile additional information such as urgency and potential cause.

2. The **Operations and Maintenance Team (OM team)** team consists of turbine engineers with expertise in various turbine subsystems. They look at the reports and alerts from the MA team and combine this information with their expertise to determine what inspections and maintenance need to be performed on turbines.

This team generally does not interact with ML outputs directly, but will look at some information from these sources, as compiled by the MA team. As such, they may also fill the decision-maker role.

Unlike in our child welfare application, the turbine engineers and data analysts have more experience with data science, and are more comfortable with feature aggregations and engineering. As such, while the feature space we use for this domain may appear less human-worded, it remains usable to our intended audience.

3. Each wind farm has its own **Site Team** that specializes in the specific requirements and properties of the turbines at that site. This team takes recommendations from the OM team, and hires out contracted turbine maintenance personnel to inspect, maintain, and repair turbines as needed. For the most part, this group fits in the affected individuals role.
4. **Contractors** are another group of affected individuals who carry out inspections, repairs and other tasks. This often requires going up the turbines in person — a potentially dangerous job.

5. Three broader categories of affected individuals include **Consumers**, which covers any person who uses energy generated by Iberdrola’s wind turbines, who may be affected by outages or costs in the energy industry; **Iberdrola** itself, as decisions made in the workflow will directly impact the company’s profits and growth; and **The renewable energy industry** as a whole, as repeated failures in renewable energy systems can reduce overall trust and support for such efforts.
6. **ML developers** on our team conducted the data and modeling tasks described in the previous section.

Throughout the rest of the development process, we ensured that our ML tool was tuned for the specific decision-maker and bridge teams identified.

### 7.2.1 Process

Having formally identified the people involved in wind turbine monitoring, we continued our interviews to better understand the decision-making process. The following describes our findings:

- Step 1.** The MA team looks at data, ML outputs, and other sources (such as our usable ML tool) to identify turbines that may experience brake pad failure or other potential problems. They filter and prioritize these alerts, and send those that pass a threshold of urgency to the OM team for further investigation. Additionally, they compile summaries of less urgent information, such as general trends, less urgent inspections, and areas for improvement, and send these to the OM team on a monthly basis.
- Step 2.** The OM team looks at alerts and summaries and determines what inspections, maintenance, and other tasks need to be done on the turbine fleet. They then work with the appropriate site teams to make an action plan.
- Step 3.** The appropriate site team hires contractors to carry out the repairs and other tasks. For example, this may involve contractors going up the turbines for physical maintenance.

Figure 7.2 summarizes the process. In particular, we are focused on the decision problems faced in steps 1 and 2 by the MA team and the OM team: identifying turbines that need attention in order to avoid brake pad failures. Generally, decisions are made on the order of days, rather than minutes or hours. The risk associated with decisions, however, is fairly high, as in-person inspections are dangerous and expensive and the cost of brake pad failures is high. The ML output (the likelihood of brake pad failure) is proxy information to the actual decision problem; it does not make a recommendation of when and whether to inspect turbines.

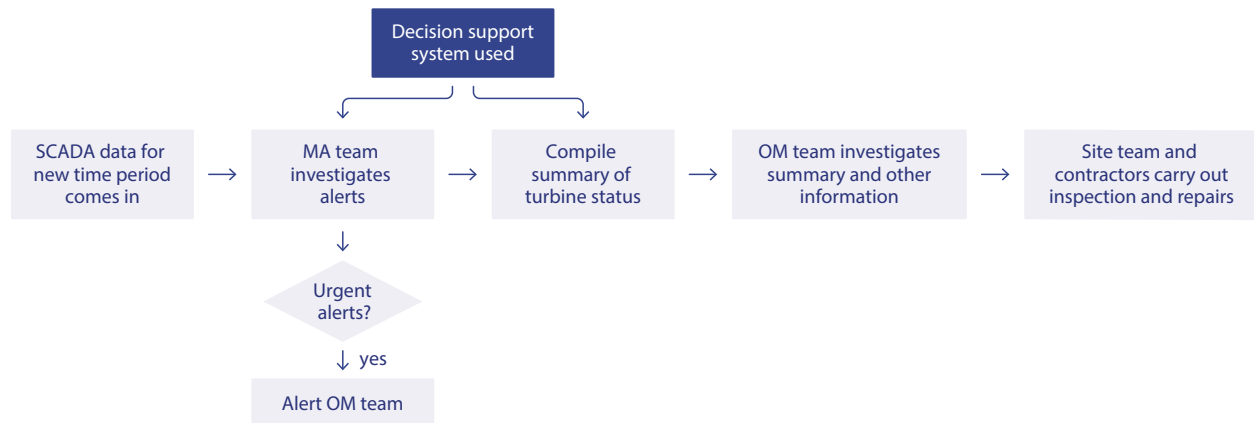


Figure 7.2: Summary of the wind turbine monitoring process.

## 7.3 Developing Usable ML Interfaces

We used Sibyl to set up a ML tool for wind turbine brake pad monitoring. Based on our findings from our interviews and context-understanding sessions, we used the context configuration shown and discussed in depth in Figure 7.3.

In this section, we introduce the interfaces that made up the ML tool we evaluated for this application.

### 7.3.1 User Feedback

Throughout the development process, we met regularly with our collaborators at Iberdrola to get feedback on our ML tool. Feedback we believed to be application-specific we incorporated into the configuration process for Sibyl; feedback we believed would generalize we integrated into Sibyl’s general implementation. As we describe our interfaces in the following sections, we bring in the feedback we received throughout the process and the design choices we made to address this feedback, which aided our ability to address of **RQ2**.

### 7.3.2 Interfaces

We elected to run our evaluations on the following interface types, though we intended to reduce this set based on evaluation results.

#### Prediction Summary

One piece of feedback we received early in the development process was that it was important for users to be able to quickly identify turbines that were predicted to experience failures, and the predicted times of these failures. This application involves ML models predicting on many turbines at many lead times, so it is easy to lose track of failure predictions. To this end, the Prediction Summary interface visualizes ML outputs over all turbines and lead times, and allows users to filter to see only turbines that have failures predicted at any time. This visualization is shown in Figure 7.4

```

1  # Model prediction type
2  output_type: boolean
3  # The label corresponding to a positive (True) prediction output
4  output_pos_label: failure
5  # The label corresponding to a negative (False) prediction output
6  output_neg_label: normal
7  # Increased model predictions correspond to a negative outcome (brakepad failures)
8  output_sentiment_is_negative: True
9  # Whether to each entity has multiple observations
10 use_rows: True
11 # Label to use for observation rows
12 row_label: "Timestamp"
13 # Whether to allow users to see prediction probabilities in addition to class
14 show_probs: True
15
16 # Context-specific overrides for common terminology
17 terms:
18   entity: "Turbine"
19   prediction: "Predicted Outcome"
20   positive: "Risk" # As in features that increase model prediction
21   negative: "Protective" # As in features that decrease model prediction

```

Figure 7.3: Context configuration used for the wind turbine monitoring application. **Lines 1-8** set configurations related to the ML output. Specifically, the model predicts True if the turbine brake pad is expected to fail at some target time, and False otherwise; therefore, higher predictions (closer to 1) correspond with a negative sentiment. **Lines 9-12** specify that entities (turbines) have multiple observation rows (timestamps) and that all rows should be shown in the interfaces. **Lines 13-14** specify that in addition to the class prediction (Normal/Failure), the *probability* of failure should also be shown, as our interviews suggested this would be useful information. **Lines 16-21** define context-specific terminology to use.

### Explore a Prediction (Feature contributions)

This interface delves into individual predictions made by the model on specific turbine–lead-time combinations, highlighting the contributions of each feature. As in the child welfare application, positive contributions (representing a higher chance of failure) are visualized in red and labeled as Risk features, while negative contributions are blue and labeled as Protective features. Per user feedback, a legend explains these colors, using clear language (i.e. *towards **failure** as predicted outcome*, rather than *positive contributions*). Users can filter by feature category, each of which corresponds to a different turbine component and data source, and search for specific features. By default, features are sorted by largest absolute contribution — but users can also elect to visualize only Risk or Protective features, or see both side-by-side, as was found helpful for child welfare screeners.

Based on feedback from collaborators, we also added the ability to directly pull up the feature-level explanation for any feature directly from this interface. Interactions like these enable faster and smoother interface use. We describe this feature-level explanation in more depth in the About the Model section below.

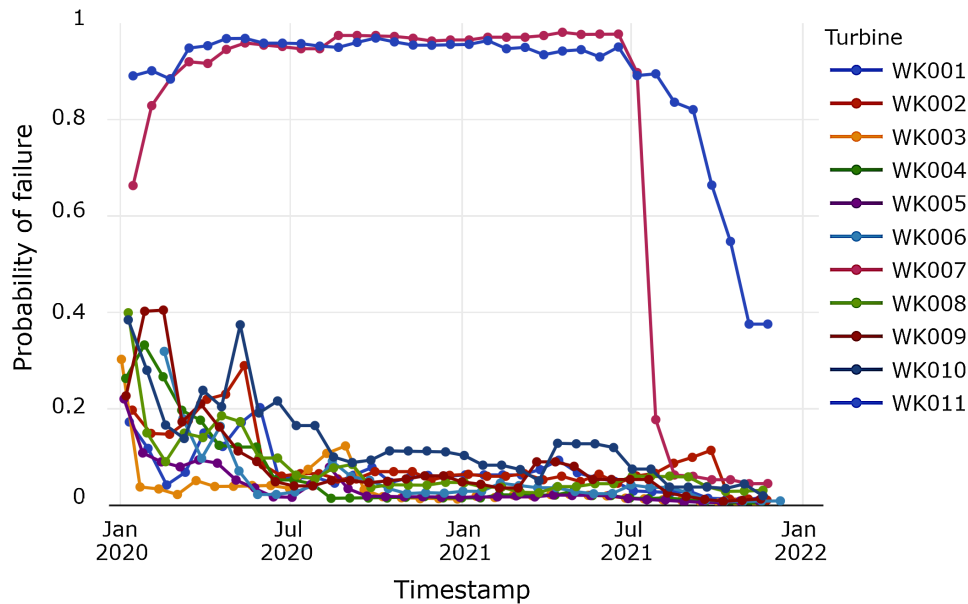


Figure 7.4: Snippet of the Prediction Summary interface for brakepad failure prediction.

### Compare Observations

This interface allows users to see how a turbine’s feature values and ML outputs compare at different times — in other words, it allows users to compare values between different observations on a turbine. This is the first of two interfaces we added with the objective of helping users understand what caused the model prediction to change over time — in particular, between predictions of no failure and predictions of failure. This interface, like the Explore a Prediction interface, shows feature contribution changes, although it also highlights which features’ contributions changed significantly between times. This allows users to quickly identify features that are more likely to correspond to brake pad failures, as these are the features that result in the most significant change in ML output.

In response to user feedback, we added the ability to compare different turbines at the same moment in time. Such comparisons allow users to investigate specific turbines that they know to be relevant.

### Change Over Time

To help users track changes in ML outputs over time even more easily, we also added a second interface, the Change Over Time interface. This plots the ML output at different lead times, as well as the change in contributions of various significant features over those time frames. Users can elect to see the features with the largest absolute contributions, the features with the highest risk or protective values (largest or smallest contributions), or the features whose contributions change the most between different lead times.

This interface allows users to investigate how models trained to predict to different lead times treat features differently, and to see which features correlate most with changes in ML output.

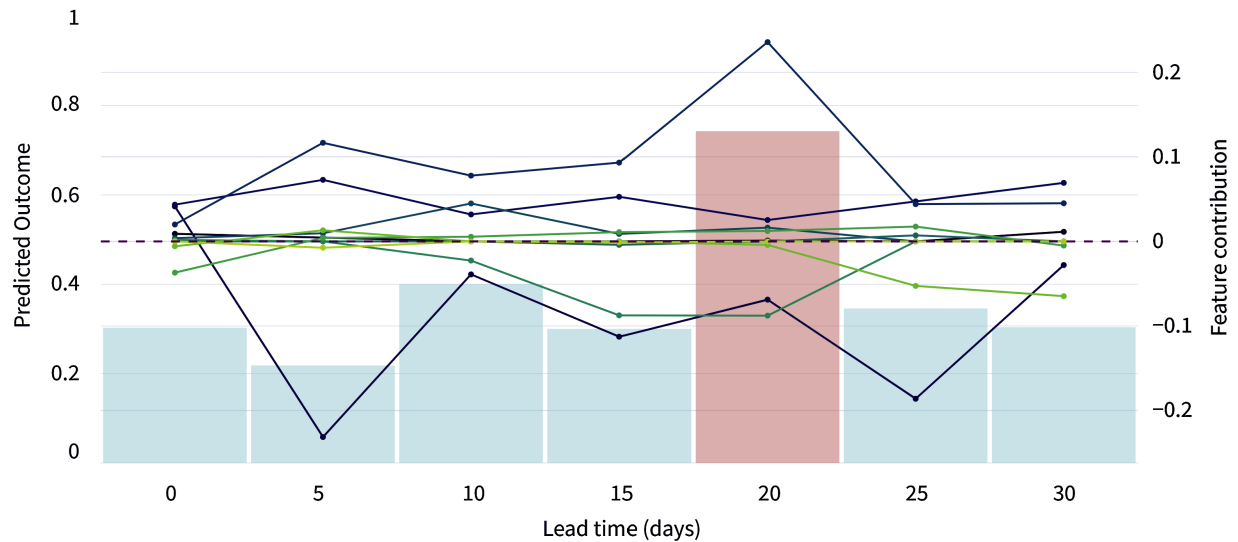


Figure 7.5: Snippet of the Sibylapp "Explore a Prediction" interface for wind turbine monitoring.

### Experiment with Changes

With this interface, users can simulate changes to feature values to see how the ML output would change under different circumstances. They can select up to 5 features at once and investigate how the ML output would change if those features had different values. As in the child welfare screeniapplication, This interface, was added with the goal of allowing users to help reconcile their disagreements with the model by testing out their hypothesized understanding of its logic.

### Understand the Model

This interface provides a deeper dive into the ML model itself with three sub-interfaces. The first shows the overall global importance of all features. Based on feedback from collaborators, we additionally added a global feature contribution sub-interface, which separates out the average contributions over the entire dataset into positive and negative components. This allows users to identify features that tend to push the model prediction up or down across the training dataset. Finally, we added a feature-level exploration sub-interface, which shows the distribution of values and the contributions of each feature across all its values in the dataset. This allows users to better understand the full range of how individual features contribute to the ML output.

### Edit Features

We created the feature names and categories, using a code-book and initial feedback from our collaborators on the MA team. However, future users are likely to prefer different terms or categorizations, as was pointed out to us during one feedback session. The Edit Feature

interface allows users to update feature descriptions and categories in the database to improve the readability of Sibyl as a whole.

We did not include this interface in our initial evaluation, as we were already using feature descriptions and categorizations selected by the MA team and wanted to keep our evaluations focused on day-to-day tasks.

## Settings

The Settings interface provides some interface configurations to support specific user needs, such as offering alternate color schemes, reducing the size of the training database to improve performance, and changing the interfaces displayed (see the last section of Table 6.2). For consistency, we did not include this interface in our formal evaluation.

## 7.4 Evaluation

In complex domains, evaluating the real-world impact of ML tools is challenging [13], [87]. As we learned from our past study in child welfare screening, formal user studies, while valuable for gaining general understanding of how ML tools are used, may be ineffective for evaluating their real-world benefits. User studies are often held in unrealistic lab settings, and require additional time and attention from users — time that often must be given outside of work hours. Additionally, formal user studies cannot capture the full spectrum of complexity involved in real-world decision problems. At the same time, studying within a simulated environment can make it easier to isolate certain aspects of ML tool use, and ML tools must be validated before being deployed. Therefore, at this stage we evaluated our ML tool for wind turbine monitoring with a realistic simulated case study.

In this work, we focus on three evaluation metric categories, addressing our final three research questions:

1. **Feedback metrics (RQ2):** What feedback did users have about the ML tool? These metrics were collected using retrospective surveys.
2. **Usage metrics (RQ3):** What interactions did users have with the ML tool? These metrics were collected through the Sibyl-API logging endpoints.
3. **Decision metrics (RQ4):** How did the ML tool influence decision-making for the wind turbine brake pad monitoring? These metrics were collected by asking users to make decisions using the tool, and then analyzing those decisions.

These metrics are elaborated on in Table 7.2.

We ran a simulated case study based on real historical data, and recorded our results on these three metric types.

### 7.4.1 Method: Simulated Case Study

We assembled an experimental dataset of 20 turbines taken from the training dataset. This set of turbines was intentionally selected to include a diverse set of turbines, including 1)



Table 7.2: Summary of Sibyl evaluation metrics for brake pad failure prediction simulated study.

Metric Category	Primary Question	Example Metrics
<b>Decision metrics</b>	How does usable ML impact decision-making?	Which turbines were included in alerts (with what prediction patterns)? What information from the usable ML interfaces was included in summaries?
<b>Feedback metrics</b>	What feedback did users have about Sibyl?	What information did users find useful or useless? What information did users think was missing? What did users find confusing? What interactions did users wish were available?
<b>Usage metrics</b>	How did users interact with Sibyl?	What search, filter, and sort options did users use? What turbines did users investigate on each page? What feature modifications did users experiment with? How much did users use each page?

turbines that were predicted to fail across the entire considered timeframe (1 turbine), 2) turbines that were predicted to start failing at some point during the considered timeframe (2 turbines), 3) turbines that were predicted to fail but then shift to normal at some point during the considered time frame (9 turbines) and 4) turbines that were not predicted to fail at any time during the considered timeframe (8 turbines).

We then recruited 2 wind turbine analysts from the collaborating MA team and asked them to use the ML tool, with this selected turbine set loaded in, for 40 minutes. We asked them to review the turbines as they would in their regular work, to answer a series of questions about decisions they would make using the tool, and to provide additional feedback.

### Results: Decision Metrics

After the analysts familiarized themselves with the ML tool, we asked the following questions to track decision metrics:

1. What information would you include in a summary to send to the appropriate teams?
2. What, if any, urgent alerts would you send?
3. What other information would you look into when assembling your summary?

For each question, we asked for open-response answers. We concluded that the information shown in the ML tool did not provide sufficient reason for the analysts to want to send any urgent alerts — this was not unexpected, as such alerts tend to be rare and consequential, and as such we would expect the analysts to look into more familiar sources of information before making such a decision.

P1 reported that they would not include any information from the explanations in summaries to the OM team, as they would not wish to reveal the model’s inner workings to collaborators until more trust in the system has been gained. This finding indicates that the ML augmentations are primarily being viewed as explanations in the traditional sense (i.e., explaining how the ML model works), as opposed to providing additional information that is revealing in itself. We discuss this further in the discussion section.

P2 indicated that they would include information about turbines with predicted brake pad failures, a list of components reported to be relevant (feature categories), and a comparison of the feature values for these components in turbines predicted to fail vs. other turbines in the wind farm.

For question 3, P1 did not specifically list any further information. P2 emphasized a desire for an uncertainty measure on the ML output as well as an explanation of this uncertainty; we consider this interaction (I14), but did not include it in the interfaces provided at this stage.

## Feedback metrics

Throughout the simulated case study, we recorded comments and feedback given by the analysts about the ML tool. After the analysts spent 40 minutes working with the tool and making decisions, we also asked a series of retrospective questions, in order to track feedback metrics:

1. On a scale from 1-5, where 1 means not useful at all and 5 means very useful, how would you rate each page in the tool?
2. Was there any information you would have liked to see that you did not find within the tool?
3. What information did you find confusing?
4. Were there any other kinds of interactions you would have liked to see?
5. Do you have any other feedback?

Figure 7.6 summarizes the usefulness scores given by participants to each of the interfaces in the ML tool. Participants felt that the interfaces overall were very useful, scoring three interfaces (Prediction Summary, Explore a Prediction, and Understand the Model) as “extremely useful”, and two interfaces (Experiment with Changes and Change over Time) as better than “very useful” on average. One participant reported not using the Compare Times interface.

We received some feedback about designs and interactions that we believe is generally applicable to usable ML. Even though our tool used domain-specific terms to refer to concepts such as entities, features, and positively/negatively contributing features, certain concepts remained confusing. Thinking about features as having both values and contributions was confusing, especially since the values of the aggregated features were not especially meaningful to the analysts. This finding suggested that we should add another configuration to our list — flexibility around whether to show feature values at all, as well as whether to show raw data. For some applications, including this one, feature values are not interpretable even with Pyreal, and may be best left out or shown in the form of the original time series.

Well-selected interactions between interfaces are valuable. For example, the analysts liked that they were able to quickly get feature-level explanation plots from the Explore a Prediction interface, allowing them to better understand how the model used highly-contributing features. Other interactions between interfaces were requested, such as the ability to more

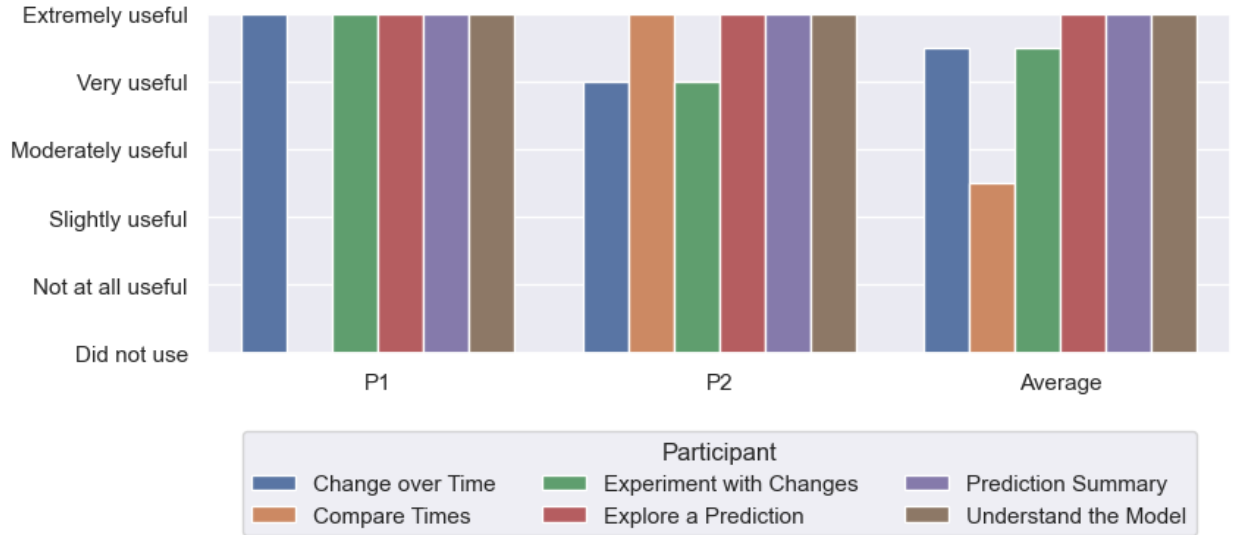


Figure 7.6: Summary of reported usefulness of each Sibyl interface in the simulated case study. The average score considers “did not use” as a score of 0.

directly see change-over-time explanations in the Prediction Summary interface. Participants were surprised that ML outputs shifted from failure to normal during the observed timeframe, and wanted to more easily access a simple explanation for why that might be.

As described above, some challenges for feature interpretability still remain. Because our features were aggregated from time series signal data, features from the same signal sometimes had very different contributions depending on the aggregations, which confused the wind turbine analysts. One possible way to address this problem, especially since we found that feature values were not used directly, is by combining the contributions across different signal aggregations into a single contribution per original signal, and either not showing the values at all or showing a simplified signal visualization.

### Usage metrics

Finally, to enable deeper investigation into how the ML tool was used, we set it up to log all the interactions the team had with the tool, which we recorded as the usage metrics. For example, we investigated questions such as:

1. How many times each interface was accessed, and for how long
2. What filters, sorts, searches, and other interactions users applied
3. What the wind turbine analysts looked at on each interface, including how much they interacted with turbines predicted to fail compared to those not predicted to fail.

Figure 7.7 summarizes the time spent on each interface during the simulated case study. Participants spent the most time looking at the overall summary of ML outputs (Prediction Summary), as well as exploring individual outputs (Explore a Prediction). Contrary to our hypothesis, participants spent significantly less time on time-comparison interfaces.

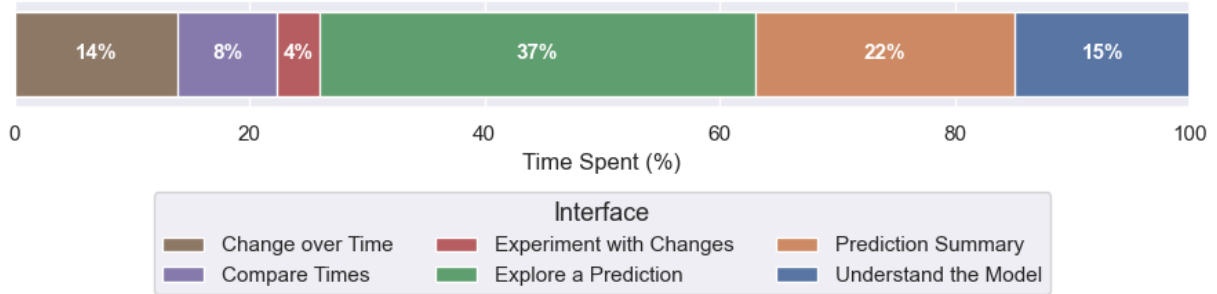


Figure 7.7: Summary of distribution of time spent on each interface during the simulated case study. Overall, participants spent the most time exploring individual predictions and prediction summaries.

Here, we summarize the filters and searches used:

- Participants engaged in many of the interactions listed in Table 6.1; specifically, **I1** (getting information about turbines), **I2** (searching and filtering feature values), **I5** (obtaining and investigating ML outputs), **I6** (investigating local explanations), **I7** (investigating global explanation), and **I12** (investigating trends in historical data used to train the model). They also reported that in long-term use, they saw value in **I22** (updating feature names to match terms used by individual users).
- We also analyzed feedback and usage to determine whether any desired interactions were not included in Table 6.1. The analysts reported that they would like the ability to flag specific features, so that they can more easily find or filter for them on explanations. They also reported a desire to see some degree of data lineage; in other words, to be able to access to the original signal values that features were aggregated from, in order to address specific questions. Finally, at times, the analysts had further questions about why the model behaved in a certain way, considered some value to be useful, or was uncertain about certain predictions. Providing such second-level explanations is a promising future area of work, which we discuss further in the discussion section.
- Participants took advantage of the ability to filter features — particularly filtering the Prediction Summary interface to focus only on turbines with failure predicted, and filtering the Change Over Time interface to focus on features that most contributed to changing ML outputs.
- We did not see much searching of features. This may have been because the participants did not know enough about the information used to be interested in specific features, or because the feature category filters fulfilled this need.
- Participants spent more time looking at information about turbines without failures predicted. However, this may be skewed by participants spending exploratory time on turbine WK001, which had no failures predicted. Filtering out results from this turbine, participants spent significantly more time on turbines predicted to fail - 17

minutes on failing-turbine-specific explanations, compared to 1.33 minutes on normal-turbine explanations. With or without turbine WK001, participants engaged in significantly more actions related to failing turbines, with 103 actions (including filtering, sorting, experimenting with changes, switching between interfaces, etc.) on failing turbines compared to 68 on normal turbines (and only 7 if not considering turbine WK001).

Overall, we see evidence that the wind turbine analysts took advantage of many of the interactions offered by the ML tool, especially on turbines predicted to fail. However, more work is needed to prune down the less useful information within the tool, and to further improve the interpretability of the features shown.

## 7.5 Discussion

Here, we discuss a few key findings from this work developing a ML tool for wind turbine monitoring that likely extend to usable ML development work in general.

**Participants treat ML tools as providing *explanations* (meant to explain the ML model) rather than *augmentations* (meant to provide additional information that is helpful on its own).** Throughout feedback sessions and user studies, we often used the word “explanation” to describe what we were providing — after all, the work we did was based on the *explainable* AI literature, and showed the results of *explanation* algorithms. However, findings in both this and the child welfare study suggested that framing information in this way resulted in decision-makers interpreting and using the ML tool differently than we had intended. Our objective was not to allow people to understand how the ML model predicts outputs, nor to justify these outputs, but rather to provide additional information that would be helpful for decision-making, per our definition of “usable ML.” In the case of wind turbine monitoring, our hope was that the ML tool would allow data analysts to provide richer, more useful summaries to other teams by helping them to pinpoint possible causes of predicted brake pad failures or to highlight components of interest. Instead, the analysts looked at the tool as a way of verifying and understanding the model itself. Going forward, we suggest that researchers be very careful about their language and push further towards interpretable explanations. Further work into developing and explaining causal ML models may also help with this goal.

**We uncovered some potentially valuable interactions and configurations that were not described in our system.** As elaborated in our results section, we found a few interactions and configurations that should be added to Tables 6.2 and 6.1, as summarized in Table 7.3. These findings suggest interesting future directions for the development of Sibyl.

For example, wind turbine analysts requested a way to flag interesting features, in order to easily find them across the ML tool. We also saw some evidence that the analysts wanted access to a richer data lineage — even though explanations were given in terms of tabular features, the analysts sometimes wanted to inspect the original signal those features were aggregated from. While providing this signal by default may result in cognitive overload, one possible extension to Sibyl may provide an option to investigate the full signal, or to assign the contributions in terms of the original signal, as in [41]. This is also a potential

Table 7.3: List of interactions and configurations that should be added to Sibyl, based on the findings in the wind turbine case study.

<b>Model Use, Vetting, and Understanding</b>	
<b>I28</b>	Look into sections of the raw data that specific features were computed from
<b>I29</b>	Provide more information about why the model does or does not use different features
<b>I30</b>	Explain why the model is more or less certain about different outputs, i.e. “explain the uncertainty,” as in [88]
<b>Logging and Annotating</b>	
<b>I31</b>	Flag features of interest
<b>Entity Configurations</b>	
<b>C16</b>	Indicate whether to show featurized values, raw data, or both

configuration worth adding — whether to show original raw values, featurized values, or both. Finally, as with the child welfare screeners, the wind turbine analysts showed some evidence of wanting further justification for why and how certain features may matter. One possible approach to providing richer explanations is through the use of large language models (LLMs), as explained in Chapter 9.

**Getting good feedback is challenging.** One challenge in this case study was how to get rich, useful feedback from a limited participant pool with limited time. Asking for feedback outright tends to result in vague or generically positive descriptions, particularly when working with long-term collaborators. Finding real feedback requires observing real or realistic decision-making tasks, forcing participants not only to consider information, but to use and synthesize it.

**Effectively evaluating the real-world impact of of a ML tool requires carefully selecting and tracking key performance indicators (KPIs).** In this thesis, we did not deploy the ML tool to evaluate; doing so would be a valuable extension. When we do evaluate the real-world impact of the ML tool, we plan to do so in terms of real-world KPIs. Here, we introduce some possible metrics.

Iberdrola’s highest-level objective is to improve the efficiency of their turbines. More specifically for the brake pad failure case study, the objective is to reduce the downtime and failure frequency of turbines, while also reducing the number of unnecessary turbine inspections. However, the metric of total turbine downtime encompasses many factors, making it very difficult to isolate the effects of an individual intervention (such as a ML tool). A more focused KPI, directly related to the intervention, would be more practical.

One possibility is to track the total number of brake pad failures that occurred in turbines during a set time frame, compared to the number of alerts that were sent by the MA team to the OM team team during the same time frame. However, actual brake pad failures are rare (around one per month across all turbines), so it may not be possible to achieve sufficient statistical power during a practical window of evaluation time.

Instead, we plan to evaluate based on one of the MA team’s existing KPIs — the portion of

alerts sent to the OM team that are deemed interesting enough to merit further investigation. This is a practical metric that is easy to track, and has been shown by the company to correlate well with turbine efficiency. It also allows us to evaluate the ML tool within the existing evaluation scheme used by our collaborating company.

In general, the primary future extensions from this work are to clean up the ML tool based on feedback from the wind turbine analysts, and then to evaluate the tool's real-world impact on KPIs.





## Chapter 8

# Explingo: System for Generating and Validating Natural-Language AI Explanations

Past work [89], [90] has suggested that some users may prefer explanations in a natural language format — what we refer to in this paper as *narratives*. (Fig. 8.1 shows an example of a narrative.) Such narratives offer many benefits: they align with how humans naturally communicate, allow users to quickly glean key takeaways, and are a first step in enabling interactive conversations where users can better understand models by asking follow-up questions.

Large Language Models (LLMs) offer a promising way of generating such narratives. Past work has suggested different ways to do so. Some work [91], [92] uses LLMs directly to explain ML predictions, without use of traditional XAI algorithms. While these approaches offer promising paths towards novel explanations, they also risk introducing inaccuracy or hallucination in explanations. An alternative approach, and the one we focus on in this paper, is to use LLMs to transform existing ML explanations generated using traditional, theoretically grounded XAI approaches into readable narratives.

We seek to address two primary research questions:

1. Can LLMs effectively transform traditional ML explanations into high-quality narrative versions of explanations?
2. How do we evaluate the quality of these narratives?

We address these questions with our two-part EXPLINGO system. To address the first question, we develop an LLM-based NARRATOR that transforms SHAP explanations from diverse datasets into narratives. We then investigate what metrics may be of value to assess the quality of narratives, and auto-grade the generated narratives using our GRADER to address the second question. Automatically grading narratives is helpful not only for tuning and evaluating the system, but also for sure as a guardrail in deployment by hiding any narratives that are not of sufficient quality.

We note that users across different domains may have significantly different preferences for how these narratives should look. Some factors — like the fact that narratives should

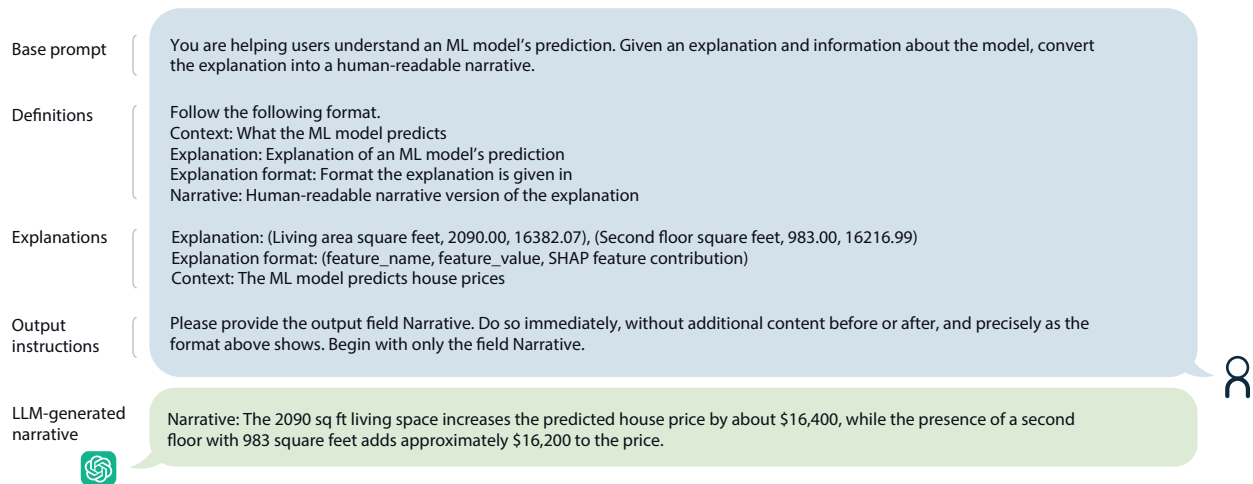


Figure 8.1: Sample NARRATOR inputs and outputs. The items in blue make up the prompt passed to the NARRATOR to transform ML explanations into narratives for a house-pricing example. The item in green is a narrative the NARRATOR LLM may generate based on this prompt. Some components, like the output instructions, are provided by DSPy [93].

accurately represent the original explanation and not be unnecessarily long — are fairly universal. On the other hand, other factors are highly subjective. Some users may wish for a narrative that includes very precise and technical details, while others prefer more general information. In this paper, we aim to generate narratives that match the style of some small, user-provided exemplar dataset of narratives. Our objective is to create a system where, for any new application, a user can write a small number of sample narratives (around three to five) and get high-quality narrative versions of future ML predictions that mimic the linguistic style and structure of those samples.

Our key contributions from this work are:

- We present our NARRATOR system that transforms ML explanations into user-readable narratives.
- We develop an independent, automated GRADER system that assesses the quality of the generated narratives.
- We curated a sample set of nine exemplar datasets that contain narratives using four public datasets, that can be used for future narrative generation tuning and evaluation work. These datasets are public and can be accessed here: <https://github.com/sibyl-dev/Explingo>.
- We provide a comprehensive set of experiments to demonstrate the impact of different prompting and few-shot strategies on narrative generation, and evaluate potential sources of reduced narrative quality.
- Our system has been integrated in Pyreal.

## 8.1 Background

Past work has aimed to apply LLM’s language-processing capabilities to XAI problems in different ways. Some work [94]–[96] has focused on using LLMs at the input stage, taking in user questions and generating appropriate explanations to answer them. Other work [91], [92] has sought to use LLMs directly to generate explanations of model logic.

In this paper, we instead build on work seeking to transform existing ML explanations into readable narratives. Burton et al. [97] used an LLM fine-tuned on a dataset of natural-language explanations to produce fluent and accurate narratives. Guo et al. [98] proposes a few-shot table-to-text generation approach that uses unlabeled domain-specific knowledge to convert tables into natural-language descriptions. We extend these works through an alternative scoring method based on a specific set of metrics, and use an approach based purely on a mix of manual and optimized prompting.

Past work [99]–[102] has also looked into using LLMs to evaluate LLM outputs, in order to reduce effort and time required from a potentially small group of human users. Fu et al. [103] found that GPT-3 was able to effectively evaluate the quality of LLM-generated text based on a wide variety of metrics such as correctness, understandability, and diversity. Wang et al. [104] similarly found that chatGPT is able to achieve state-of-the-art correlation with human graders. Liu et al. [105] found that advanced prompting techniques such as chain-of-thought prompting can further improve correlation between LLM and human grades. We applied these insights to optimize and evaluate our LLM-generated narratives using an LLM-based metric grader, eliminating the need for human graders and enabling evaluation across a wider range of domains without relying on domain experts.

## 8.2 The EXPLINGO System: NARRATOR and GRADER

To transform XAI explanations into narratives, the EXPLINGO system includes two sub-systems: 1) NARRATOR: a system to transform traditional explanations of ML model predictions into accurate, natural narratives. 2) GRADER: a metric grading system to assess the quality of narratives.

### 8.2.1 NARRATOR: Transforming ML Explanations to Narratives

Transforming traditional ML model explanations into readable narratives offers several potential benefits. First, these narratives allow key takeaways to be read quickly. Second, they are often the most natural format for parsing information for many users, who may prefer them to graphs or have difficulty understanding graphs that are not visualized effectively. Finally, offering narrative versions of explanations is the first step to allowing for full “conversations” with ML models, with users able to ask follow-up questions in natural-language and receive answers in kind.

The NARRATOR is build on an LLM model, in our case GPT-4o, that is prompted to generate a narrative. The basic prompt is composed of four components: a base prompt, a definition of the input component structure, three specific explanation components (the ML explanation, the explanation format, and the context description), and output instructions.

The prompt is then concluded with **Narrative:** invoking the LLM to start narrating. The first four blocks in Fig. 8.1 show an example prompt that follows this structure. As described in depth later, we additionally pass selected exemplar narratives that help the NARRATOR customize its narratives for specific applications.

The **ML explanation** — for example, a SHAP feature contribution explanation, as we focus on in this paper — is generated using some external library. We then parse this explanation into a consistent format, depending on the explanation type. For the SHAP explanation, we select the  $N$  most contributing features by absolute value, and parse them in sorted order into the following format:

*(feature name 1, feature value 1, SHAP contribution 1), (feature name 2, feature value 2, SHAP contribution 2), ...*

In addition to the explanation, the narrator takes in a brief **explanation format** describing how to parse the given explanation. Passing this format description separately makes the system generalizable to any kind of ML explanation that can be expressed in text. Moreover, it allows the explanation to be customized for additional information to be passed.

Finally, we write a brief description of the **context** of the explanation — generally, what the ML model being explained is predicting. For SHAP explanations, this helps provide units to the contributions.

The last box in Fig. 8.1 shows an example of the NARRATOR’s output narrative.

Once the NARRATOR generates narratives, we want to assess if these narratives are high-quality and expressing the right information. This is where the GRADER plays a role.

### 8.2.2 GRADER: Assessing the Quality of Narratives

To assess the quality of narratives, we adapted metrics proposed in [87], which include:

- *Accuracy*: how accurate the narrative is to the original ML explanation.
- *Completeness*: how much of the information from the original ML explanation is present in the narrative.
- *Fluency*: how well the narrative matches the desired linguistic style and structure.
- *Conciseness*: how long the narrative is considering the number of features in the explanation.

While there are other metrics that can be included, for the presentation of the GRADER, we focus on these four. The challenge with these metrics is that they are time consuming to manually compute, and manual assessment could introduce human bias into the evaluation.

Our GRADER automatically evaluates the quality of the narrative by prompting an independent LLM, GPT-4o in our case, to consider the ML explanation and the generated narrative and assign a value to the respective metric. Fig. 8.2 shows an example of a GRADER prompt evaluating the accuracy metric. In summary, it takes in the input and output of the NARRATOR along with a rubric criteria, and comes up with a score depending on which metric it is asked to assess. In Fig. 8.2, the grader return will return either **1**,

Table 8.1: Summary of exemplar datasets. In the Ames Housing Dataset [72], each row represents one house, with 79 input features. The label to be predicted is the price the hold sold for. The Student Performance Dataset [73] includes 30 features, and each row represents a student. The label to be predicted is whether the child will pass or fail a class. The Mushroom Toxicity Dataset [106] has 22 features about different species of mushrooms, and the predicted label is whether the mushroom is poisonous or not. Finally, the PDF Malware Dataset [107] has 37 features about different PDFs, and the prediction label is whether the PDF contains malware.

ID	Dataset	# of Entries	Sample Narrative
House 1	Ames Housing	35	The relatively smaller above ground living space in this house reduces its predicted price by about \$12,000. The lower-than-average material rating also reduced the house price by about \$10,000.
House 2	Ames Housing	22	The SHAP value indicates that the house price decreases because the above ground living area is 1256 sq ft and the material quality is rated 5.
House 3	Ames Housing	22	This house is cheaper because it has less above ground living space (size=1256) and has lower material quality (rating=5).
Mush 1	Mushroom Toxicity	30	This mushroom is more likely to be poisonous because its foul odor, silky stalk surface, and chocolate spore print color. Be careful!
Mush 2	Mushroom Toxicity	30	The absence of odor and broad gill size suggest the mushroom is less likely to be poisonous, but the brown spore print indicates a higher risk of toxicity.
PDF 1	PDF Malware	30	The PDF file is more likely to contain malware because it has a larger metadata size (262 KB), a larger total size (74 KB), and no Javascript keywords.
PDF 2	PDF Malware	30	The large metadata size (180 KB), the large total size (7 KB), and fewer objects suggest the PDF contains malware.
Student 1	Student Performance	30	We believe this child is less likely to pass because they do not have family support, and we have seen that male students tend to be more likely to fail.
Student 2	Student Performance	30	The lack of family support, and the sex (male) suggest the student is less likely to pass the class. But, the lack of a romantic relationship indicates an higher probability of passing.

indicating an accurate narrative, or **0** for an inaccurate narrative. Straight out-of-the-box, it can be difficult for LLMs to produce the exact desired output. Section 8.4 goes into details about how we tuned, validated, and used the GRADER system.

Putting effort into developing and validating a GRADER system offers several benefits. A well-validated grader can quickly adapt to a large variety of domains and dramatically reduce human-effort required to evaluate narratives. It can also be used as part of the fine-tuning process, as certain LLM optimization functions like bootstrapping few-shot examples optimize based on automated evaluation functions. Finally, and perhaps most importantly, a well-validated grader can be used as a guardrail in live deployment scenarios — rather than risking showing inaccurate or low-quality narratives to users, a system can automati-

cally revert to the default, graph-based ML explanation in situations where the generated narrative scores are unacceptable.

It is worth noting that in previous work, metrics such as METEOR [108] have been used to evaluate LLM output quality. However, to compute these metrics, we need a ground-truth label for every narrative. In real-world settings, we do not have these labels available, making it impractical in deployment. With GRADER, we alleviate this barrier.

### 8.2.3 Using EXPLINGO for a New Application

Having defined our two subsystems, we now describe the steps a user will take when applying EXPLINGO to a new ML decision-making problem<sup>1</sup>. We will consider a sample application of a user wishing to apply EXPLINGO to the task of house valuation. We assume the user has a ML model trained to predict house prices.

1. The user generates a set of five SHAP explanations, explaining the model’s prediction five (ideally diverse) different houses. For each of these explanations, the user assembles an exemplar by parsing them into the format described in the **Explanations** box of Fig. 8.1.
2. For each exemplar, the user manually writes a short narrative describing the explanation in the desired style. For example, for an explanation of (*neighborhood, Ocean View, 12039*), (*construction date, 2018, 3204*), the user may write a narrative like *This house is expected to sell for more than average due its prime location in Ocean View and because it was built fairly recently, in 2018.*
3. The user adjusts the weighting of the GRADER metrics according to their needs; for example, they may choose to value accuracy higher than fluency.
4. The user can now pass explanations into the NARRATOR, and get back narrative versions of these explanations. They can show these narratives to prospective homebuyers, real estate agents, or other stakeholders interested in understanding house prices.
5. Optionally, the user may set up the guardrails process. In this case, they will pass the generated narratives into the GRADER. Any narrative that does not achieve some threshold score will be rejected and not shown to stakeholders.

Other than potentially adjusting metric weighting, the user does not need to customize the GRADER for their application. Similarly, the only customization required for the NARRATOR is to pass in a small set of exemplars; the NARRATOR automatically uses these narratives to generate narratives in the desired style that achieve high scores on the weighted metrics. We designed both subsystems to be application-agnostic and validated them on multiple diverse applications

We now dive deeper into the EXPLINGO framework.

---

<sup>1</sup>We have integrated this process in the Pyreal library, described in more detail in Section 8.7.1, which removes the manual effort required to generate and format explanations.

Table 8.2: Summary of metric validation datasets for the accuracy and completeness metrics. For accuracy, we include both complete and incomplete narratives to ensure they are both be rated as accurate.

Metric	Type	Rubric Grade	Count
Accuracy	Accurate with all values	1	16
	Accurate with some values	1	9
	Accurate with approximate values	1	9
	Error in values	0	8
	Error in contribution direction	0	8
	Total		
Completeness	All features, values, and contributions	2	10
	All features, values, and contribution directions	2	5
	All features, but not all values or contribution directions	1	17
	Missing one or more features	0	18
	Total		

### 8.3 Exemplar and Evaluation Datasets

Narratives can be written in numerous ways and the desired style of narratives is highly subjective. Therefore, we introduce the concept of an exemplar dataset, which is a small dataset provided by the user containing high-quality examples of narratives. We steer both the GRADER fluency metric and the NARRATOR based on these exemplars.

For this work, we created nine datasets to take exemplars from and to use for evaluations. We started by picking four publicly available ML datasets. We selected these datasets to include a wide variety of contexts ranging from predicting prices of houses to predicting toxicity of mushrooms. From these we created exemplars. To create an exemplar dataset, we pick a data point and form an entry. Each entry represents one model prediction on one data point. Each entry includes a SHAP explanation, an explanation format descriptor, and a brief context sentence describing what the model task is (e.g. predicting house prices). Our nine exemplar datasets are summarized in Table 8.1.

**Hand-written narratives:** For each exemplar dataset, we picked a few entries and created hand-written narratives given their SHAP explanations. These narratives were written by different authors and had a distinctive writing style. Different contexts (i.e., what model predicts) require some amount of customization to achieve the best possible narratives automatically from an LLM. These hand-written narratives thus become a foundational piece in our system. To use our system for a new context (a new model), the user creates a few entries and writes narratives with their preferred style and structure. These entries can then be used to optimize our system to provide better narratives specific for the context and users’ preferred narration style.

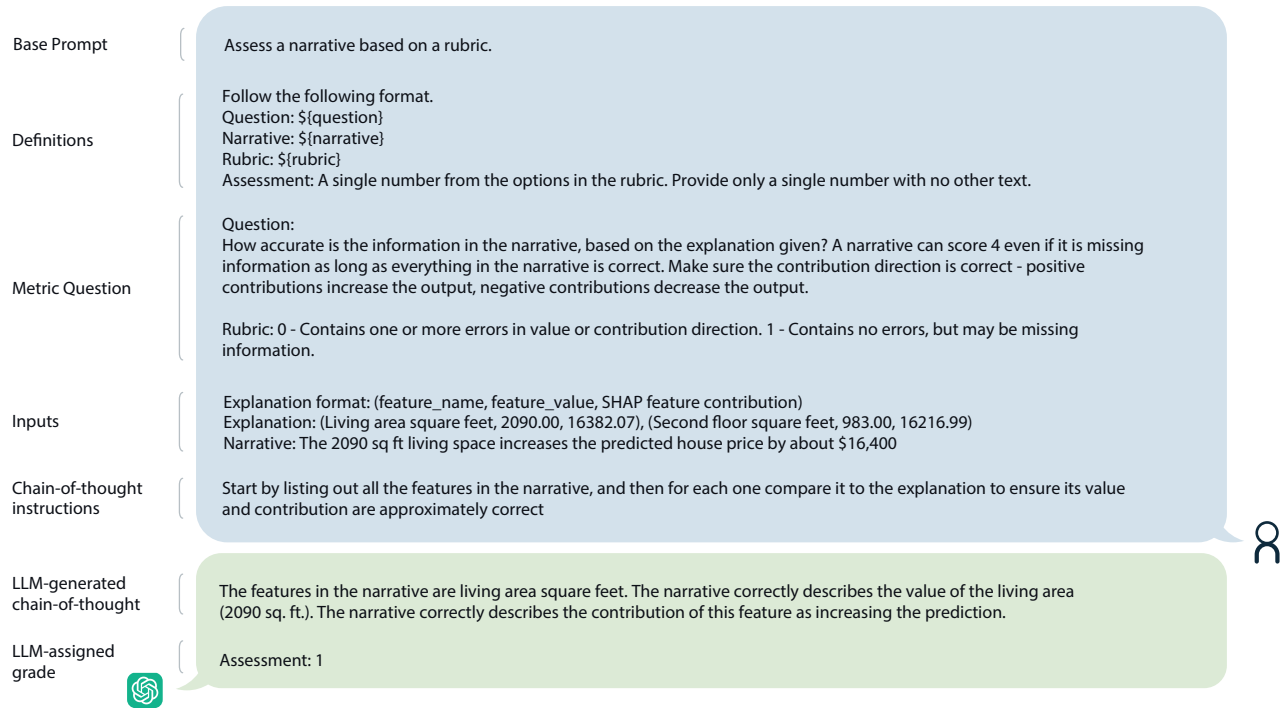


Figure 8.2: Prompt passed to the GRADER to compute the accuracy metric. Experiments suggested that asking for a grade from a 2-point rubric resulted in better performance than a simple yes/no question. We saw that the GRADER incorrectly considered narratives that were accurate but did not include all feature values from the input explanation as inaccurate, and regularly did not notice when contribution directions were wrong if the values were correct. We added explicit instructions to the prompt for these two scenarios accordingly. Finally, we determined that using a chain-of-thought prompting approach further improved the GRADER’s ability to correctly score accuracy.

## 8.4 Automated Grading of Narratives

In this paper, we focus on generating narratives that are optimized on four metrics: accuracy, completeness, fluency, and conciseness.

Scoring conciseness is a straightforward computation based on word-count; the other metrics were graded using a GPT-4o LLM. We had the LLM grade each narrative 5 times and then return the average grade. To ensure consistency in grades, we would check and alert if fewer than 4 out of these 5 scores agreed; however, this alert was never raised during our experimentation.

Our total grade  $G$  for a narrative is then computed by weighting and aggregating all metrics with the following equation:

$$G = \alpha_a A + \alpha_f F + \alpha_c C + \alpha_s S \quad (8.1)$$

where  $A$  is the Accuracy grade,  $F$  is the Fluency grade,  $C$  is the Completeness grade,  $S$  is the conciseness grade, and the  $\alpha$  parameters are weighting constants.



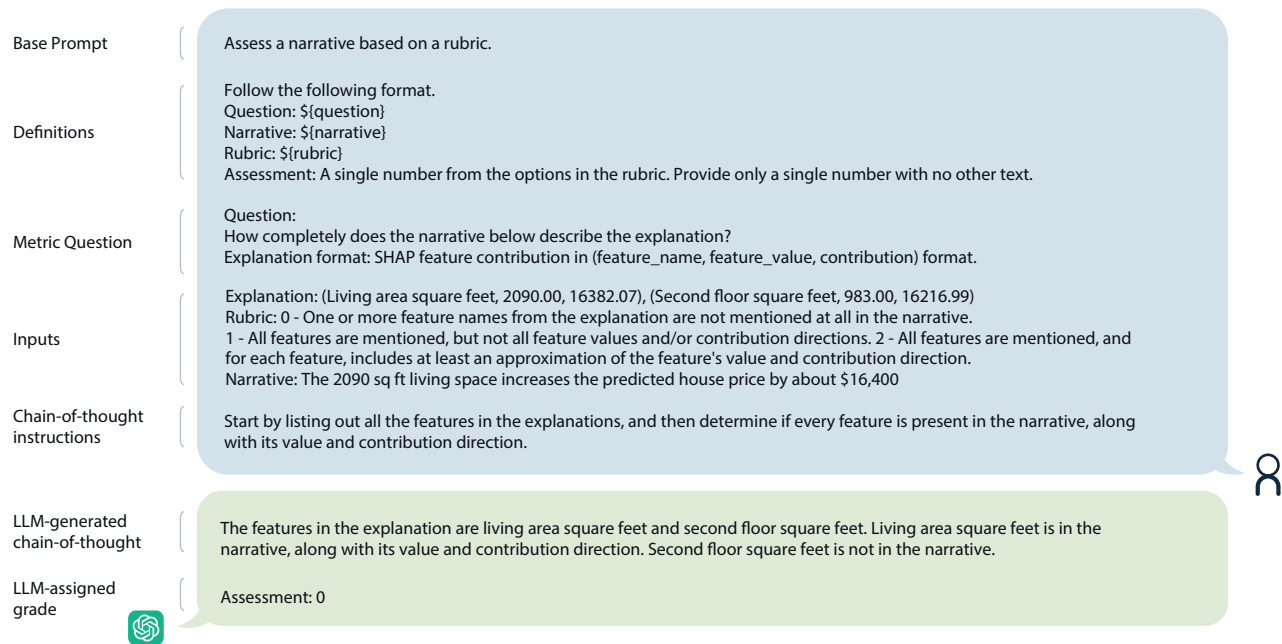


Figure 8.3: Prompt passed to the GRADER to compute the completeness metric. We found that while the original version of the prompt allowed the LLM to correctly identify missing feature values, it was regularly not identifying when features were missing altogether or did not have their feature directions listed. Through experimentation, we determined that a chain-of-thought [109] approach that explicitly asked the GRADER to list out and consider each feature in the explanation one-by-one addressed this issue well.

As described in more depth below, accuracy is scored on 2-point 0/1 scale, completeness is scored on a 3 point 0/1/2 scale, and fluency and conciseness are scored between 0 and 4. For our experiments, we therefore equally weighted all metrics by setting  $\alpha_a = 4$ ,  $\alpha_c = 2$ ,  $\alpha_f = \alpha_s = 1$ , and report these weighted metric values in our results.

In running and optimizing LLMs, we used DSPy<sup>2</sup> [93], an open-source LLM programming framework. DSPy enabled us to experiment with different LLM steering and tuning approaches and run evaluations in a structured manner.

In the rest of this section, we introduce the metrics we used to evaluate narratives, and our process for developing and validating our GRADER.

### 8.4.1 Accuracy

The accuracy metric evaluates if the information in the narrative is correct based on the input explanation. For example, in an accurate narrative, all feature values, SHAP feature contribution values, and contribution directions (positive or negative) are correct. Narratives that do not mention all features can still be considered accurate. This metric was scored on a Boolean scale of 0 (inaccurate) or 1 (accurate).

Accuracy is scored by the GRADER by passing in a prompt to the LLM. This prompt

<sup>2</sup><https://github.com/stanfordnlp/dspy>

includes the input ML explanation, the explanation format, and the narrative to grade, and asks the LLM whether or not the narrative was accurate.

In order to tune and validate the prompt passed into our GRADER for accuracy, we developed a metric validation dataset of 50 explanation-narrative pairs, along with human-labeled accuracy grades. These examples contained randomly selected SHAP explanations, with narratives. These narratives were intentionally written to span five levels of accuracy, as summarized in Table 8.2.

We then started with a basic prompt (*How accurate is the information based on the explanation given?*) and passed the dataset entries to the GRADER. We then compared the LLM-generated grades to the human-labeled grade for each entry to get a performance score across each level-of-accuracy category. Finally, we used these fine-grained results to manually adjust our prompt until we got sufficient performance (at least 95% agreement with human labels). Fig. 8.2 shows our final version of the prompt, as well as the manual adjustments we made.

The final version of our GRADER agreed with human labels on 96% of entries. A more detailed analysis of these results is shown in Table 8.3.

### 8.4.2 Completeness

The completeness metric grades how much of the information from the input ML explanation is present in the narrative. For our experiments, we define a complete explanation as one that 1) mentions all features and their values from the ML explanation and 2) mentions the contribution direction (increasing or decreasing the model prediction) for all features. We note that this is subjective and context-dependent; for example, in some domains, the exact contribution values may be needed. Our prompts use a custom rubric that can be adjusted for different needs.

Completeness is scored by prompting an LLM with the explanation, explanation format, and narrative, and asking it to score completeness on a scale of 0 (missing one or more features), 1 (contains all features but missing one or more feature values or contribution directions) or 2 (includes all features, values, and contribution directions).

Like for the accuracy metric, we created a metric validation dataset of 50 human-labeled explanation-narrative pairs, as summarized in Table 8.2, that included narratives of different levels of completeness.

Again, we started with a basic prompt (*How completely does the narrative below describe the explanation given?*) and passed the dataset entries to the GRADER. We compared the

Table 8.3: Agreement between GRADER and human-labeled validation set for Accuracy (left) and Completeness (right).

		GRADER				GRADER		
		0	1			0	1	2
Human	0	15	1	Human	0	18	0	0
	1	1	33		1	0	17	0
						2	0	13

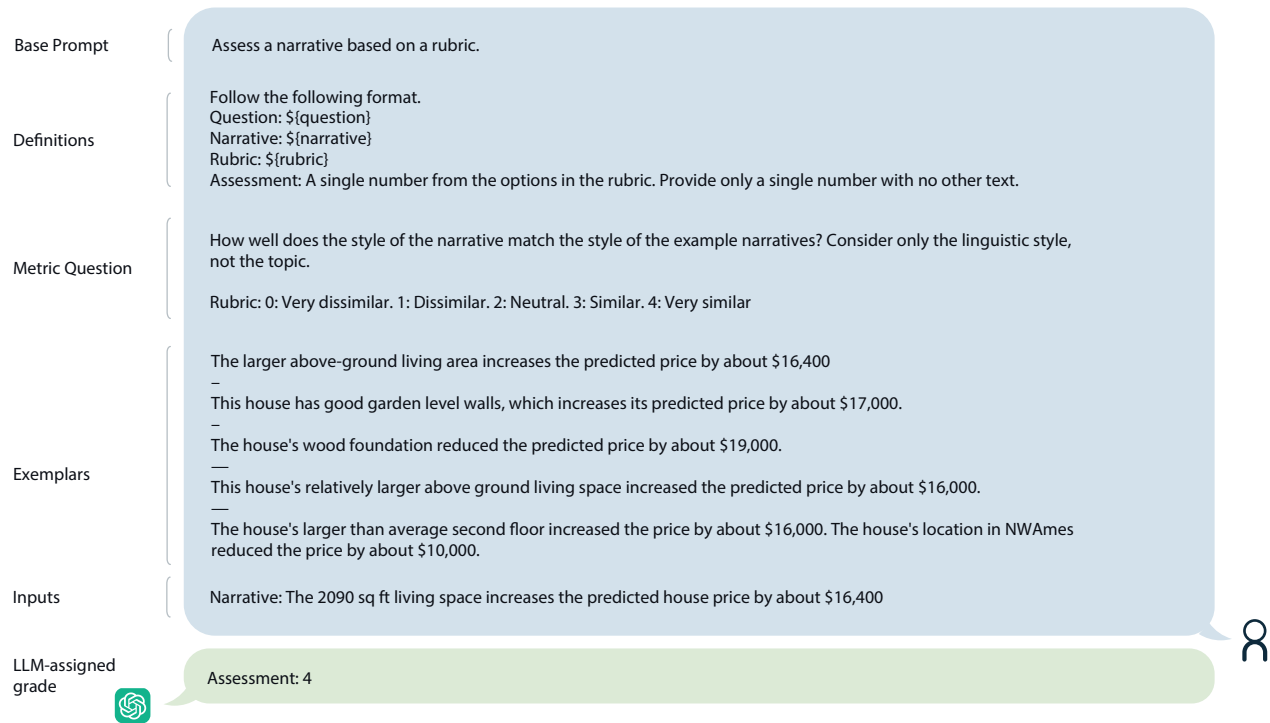


Figure 8.4: Prompt passed to the GRADER to compute the fluency metric. We found that the GRADER was over-emphasizing narrative content, as shown by a significant difference in score between narratives from different datasets with original prompt (*How well does the narrative match the style of the example narratives?*). We therefore added a statement to explicitly ignore topic. Our experiments determined that there were diminishing returns to effectiveness after 5 exemplar narratives.

LLM-generated grades to the human-labeled grade for each entry to get a performance score across each level-of-completeness category, and used these fine-grained results to manually adjust our prompt until we got sufficient performance (at least 95% agreement with human labels). Fig. 8.3 shows our final version of the prompt, as well as the manual adjustments we made.

We ended up with the prompt shown in Fig. 8.3, which we evaluated on our metric validation dataset. Using this prompt, the GRADER’s grades agreed with the human-labeled grades on 96% of entries. The full results of validation are shown in Table 8.3.

### 8.4.3 Fluency

Fluency is the degree to which the narrative sounds natural or human-written. This is highly subjective and context-dependent; therefore, for this paper we grade fluency as how closely the narrative’s linguistic style matches that of a few exemplar narratives. For example, consider the following two narrative styles for the same explanation:

**Precise:** The house’s size (1203 sq. ft.) increased the ML output by \$10,203. The size of the second floor (0 sq. ft.) decreased the ML output by \$4,782.

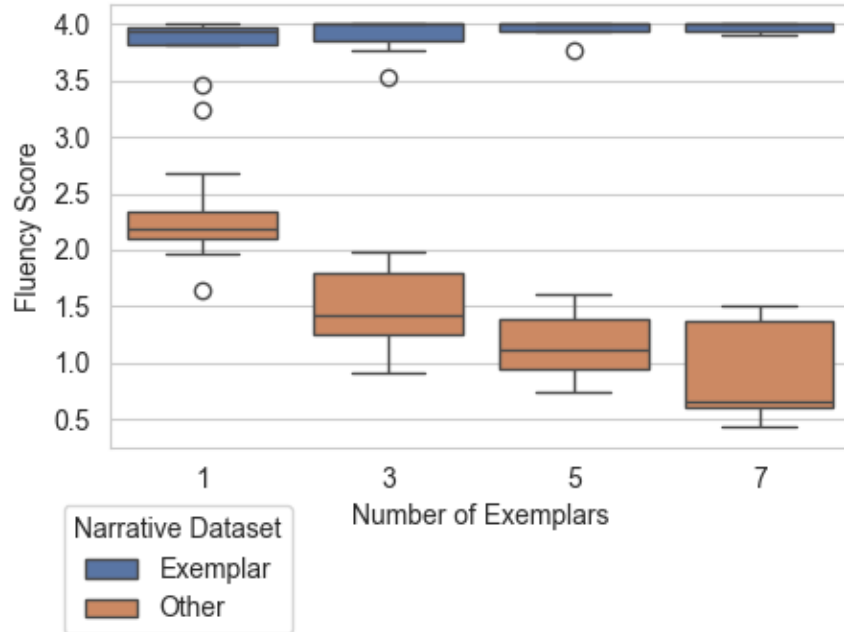


Figure 8.5: Difference between fluency scores on narratives compared to exemplars from their own datasets (blue) and those compared against other datasets (orange). We expect to see significantly higher fluency scores for the former compared to the latter. We see that adding more exemplars increases this difference, with diminishing returns after 5 exemplars.

**Casual:** The house’s large size increases the price prediction by about \$10,200, while its lack of a second floor decreases the prediction by about \$4,700.

Users pass in a small set of hand-written exemplar narratives that are then used by the GRADER to grade fluency. We prompted the LLM with these exemplar narratives, along with the narrative to be graded, asking it to score how well the narrative matches the style of the exemplars on a discrete scale from 0 to 4. Fig. 8.4 shows the prompt we used to grade fluency.

To validate our fluency scoring function and identify the right number of exemplars to include in the prompt, we used our nine exemplar datasets introduced in Section 8.3. From each dataset, we randomly selected  $N$  hand-written narratives as exemplars and asked the GRADER LLM to compare them to the remaining hand-written narratives. We then computed the mean and minimum difference between fluency scores on narratives evaluated against exemplars from their own dataset and fluency scores on narratives evaluated against other datasets.

We compared results from  $N = [1, 3, 5, 7]$ . The results are visualized in Fig. 8.5. We see that higher numbers of exemplars increases the ability of the GRADER to differentiate narratives of the same style as the exemplars from those in other styles, with diminishing returns after 5 exemplars. We therefore set our GRADER to use 5 exemplars (or as many as are available, if fewer than 5 are provided).

### 8.4.4 Conciseness

This metric grades how long the narrative is, scaled such that shorter narratives get higher grades. We believe that, all else being equal, users will tend to prefer shorter narratives.

This metric is computed deterministically and scaled to a continuous value between 0 and 4, using the following equation:

$$\text{grade} = \begin{cases} 0 & \text{if } L \geq 2FL_{max} \\ 4 \times \left(2 - \frac{L}{FL_{max}}\right) & \text{if } FL_{max} < L < 2FL_{max} \\ 4 & \text{if } L \leq FL_{max} \end{cases} \quad (8.2)$$

where  $L$  is the number of words in the narrative,  $F$  is the number of features in the input explanation, and  $L_{max}$  is some configured value representing the longest ideal number of words in the narrative per feature.

In effect, this equation grades a narrative as 4 if it is shorter than some hyperparameter-defined number of words per feature, 0 if it is longer than twice this value, with grades scaling linearly between these two extremes.

## 8.5 Optimizing the Narrator

Having validated our metric scoring functions, we sought to improve our NARRATOR to transform explanations into narratives that optimize these metrics. Like with our GRADER LLM, we used DSPy programming to assemble our prompts and used the GPT-4o API for our NARRATOR LLM<sup>3</sup>.

As shown in Fig. 8.1 we have a base prompt that instructs the LLM what it needs to do given the three inputs: the explanation, the explanation format, and the context. Next, we show a few base prompts we tried.

**Base Prompts** We used the following prompts as a first step for baselines:

**Base Prompt 1:** You are helping users understand an ML model’s prediction. Given an explanation and information about the model, convert the explanation into a human-readable narrative.

**Base Prompt 2:** You are helping users who do not have experience working with ML understand an ML model’s prediction. Given an explanation and information about the model, convert the explanation into a human-readable narrative. Make your answers sound as natural as possible.

**Base Prompt 3:** You are helping users understand an ML model’s prediction. Given an explanation and information about the model, convert the explanation

---

<sup>3</sup>We experimented with multiple larger API-based LLM models for the NARRATOR including GPT-3.5, GPT-4o, Claude, Gemma, and Mistral models. Based on initial results, we decided to run experiments using GPT-4o.

into a human-readable narrative. Be sure to explicitly mention all values from the explanation in your response.

We found that the NARRATOR was able to generate reasonably complete and accurate narratives with these base prompts along with the the other three inputs. For the remaining experiments we only used base prompt 1, as it has the fewest tokens, scored highly on accuracy and completeness in our initial experiments, and does not include any information to suggest a specific style.

However, without exemplars of what a good narrative looks like for a specific application, the NARRATOR is not able to customize the narrative style and length. For example, for an explanation from our House 1 dataset, the generated narrative using just the base prompt looks like *“The model predicts the house price based on several key features. The above ground living area, which is 2090 square feet, contributes significantly to increasing the price by 16382.07 units.”* For this dataset, based on the hand-written exemplar narratives, we were seeking a narrative style more like *“The relatively larger above-ground living space in this house increased its predicted price by about \$16,000.”*

As described in Section 8.3, we selected five hand-written narratives for each exemplar dataset. We next consider if we can add these to the prompt to better customize narrative style.

**Hand-Written Few-Shot.** In this experiment, we add a small number (denoted as  $H$ ) of exemplar hand-written narratives to the prompt. These narratives were randomly selected for each prompt from the hand-written narratives we have for the exemplar dataset.

**Bootstrapped Few-Shot.** In the previous experiment, we are limited to the hand-written narratives available to us. Additionally, these exemplars may not be complete or accurate. We therefore used DSPy’s `BootstrapFewShot` to create more exemplar narratives that score highly according to our GRADER. To do this we give the bootstrapper:

- our hand-written narratives as a starting point;
- a requirement that exemplars it produces achieve a perfect score on accuracy, completeness, and fluency and at least 3.5 on conciseness as measured by our GRADER<sup>4</sup>.

Once we have the bootstrapped exemplars, we add them, along with the the hand-written exemplars, to the prompt shown in Fig. 8.1. Thus the prompt now has  $H$  hand-written and  $B$  bootstrapped exemplars. With this addition, our prompt now has the base prompt, the explanation, the explanation format, the context, and  $H + B$  exemplar narratives.

## 8.6 Results

We now evaluate the quality of narratives generated by our NARRATOR, and investigate how the exemplars provided influence these results.

**Experimental Setup.** Using the NARRATOR we created narratives for explanations in the exemplar datasets. We used a variety of settings for the narrator: 3 base prompts and 11 few-shot (exemplar) settings. In total, we had 169 explanations (across 9 datasets) and

---

<sup>4</sup>We lowered the requirement slightly for conciseness as this significantly reduced bootstrapping time.

Table 8.4: Narrative quality scores for narratives generated by our NARRATOR, as evaluated by our GRADER, averaged across all datasets. Metrics are reported in terms of their weighted (0-4) values. Values greater than 3.5 are in blue, values less than 3 are in red, and the highest scoring row for each metric is in bold.  $H$ =number of hand-written few-shot exemplars,  $B$ =number of bootstrapped few-shot exemplars.

Base Prompt	$H$	$B$	Accuracy	Completeness	Fluency	Conciseness	Total grade
Base Prompt 1	0	0	<b>4.000 ± 0.00</b>	<b>4.000 ± 0.00</b>	2.467 ± 0.76	0.850 ± 0.94	11.317 ± 1.23
Base Prompt 2	0	0	3.911 ± 0.27	<b>4.000 ± 0.00</b>	2.467 ± 0.66	0.872 ± 0.89	11.250 ± 0.97
Base Prompt 3	0	0	<b>4.000 ± 0.00</b>	<b>4.000 ± 0.00</b>	2.222 ± 0.82	1.056 ± 1.07	11.278 ± 1.44
Base Prompt 1	1	0	3.289 ± 1.35	3.511 ± 0.56	3.622 ± 0.47	3.749 ± 0.14	14.171 ± 1.67
Base Prompt 1	3	0	3.111 ± 1.57	3.422 ± 0.53	3.689 ± 0.41	3.708 ± 0.19	13.930 ± 1.82
Base Prompt 1	5	0	3.111 ± 1.57	3.289 ± 0.56	3.644 ± 0.49	3.669 ± 0.35	13.713 ± 2.08
Base Prompt 1	1	1	3.800 ± 0.40	3.800 ± 0.23	3.650 ± 0.34	3.702 ± 0.32	14.952 ± 0.58
Base Prompt 1	1	3	3.800 ± 0.40	3.800 ± 0.23	3.700 ± 0.26	3.734 ± 0.28	<b>15.034 ± 0.51</b>
Base Prompt 1	3	1	3.323 ± 1.34	3.538 ± 0.49	3.846 ± 0.19	3.843 ± 0.16	14.551 ± 1.72
Base Prompt 1	3	3	3.262 ± 1.33	3.631 ± 0.58	3.846 ± 0.23	3.890 ± 0.11	14.629 ± 1.73
Base Prompt 1	5	1	3.289 ± 1.35	3.333 ± 0.63	<b>3.933 ± 0.14</b>	3.983 ± 0.03	14.538 ± 1.63
Base Prompt 1	5	3	3.289 ± 1.35	3.467 ± 0.66	3.889 ± 0.27	<b>3.988 ± 0.02</b>	14.632 ± 1.65
Base Prompt 1	5	5	3.378 ± 1.37	3.422 ± 0.64	3.911 ± 0.15	3.977 ± 0.03	14.688 ± 1.66

applied 13 techniques, resulting in 2,197 total narratives<sup>5</sup>. We scored each narrative on each of the four metrics described in Section 8.4. For the conciseness metric, we set the maximum input length to 90% of the longest feature description from the exemplar narratives. We also used these narratives (5 per dataset) to grade fluency. The mean metric grades from our evaluation by base-prompt/few-shot setting are shown in Table 8.4.

**Adding few-shot examples (exemplars) greatly improves narrative style at a small cost to correctness.** In Fig. 8.6, we visualize the impact that the number of exemplars has on each metric. We see a clear trend of fluency and conciseness grades increasing with more exemplars, while accuracy and completeness grades decrease. These findings are not surprising — without an exemplar, the NARRATOR has nothing to match to improve fluency and conciseness. On the other hand, more exemplars introduces more complexity that may increase the chance of the NARRATOR hallucinating, and increase the chance of confusing the NARRATOR with a potentially inaccurate exemplar narrative. The fact that we see high accuracy and completeness scores without any exemplars demonstrates that the LLM is able to complete the task with careful prompting out-of-the-box; adding exemplars mainly serves to adjust the style and structure of the narrative.

**We see the overall highest quality narratives with a small number of hand-written and bootstrapped exemplars.** Our highest total grade across all four metrics was achieved with one hand-written few-shot exemplar and three bootstrapped exemplars. Having one of each style of few-shot exemplar achieved a close second-highest score.

**Bootstrapped exemplars contribute more to generating accurate and complete**

<sup>5</sup>The total LLM costs when using GPT-4o for generating and evaluating all narratives was about US\$8.44, or .38 cents per narrative to generate and evaluate on all metrics. The estimated number of input tokens required to transform one explanation ranges from about 150 to 1000 depending on explanation length and prompting technique.

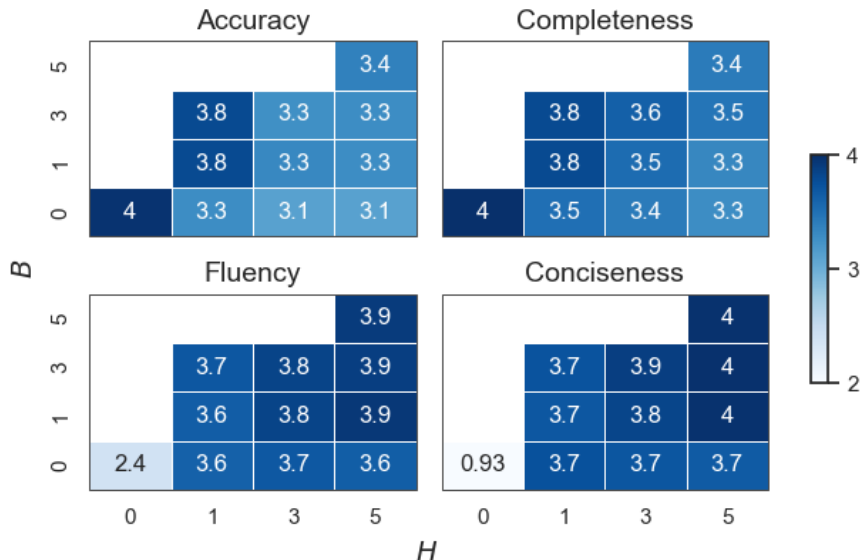


Figure 8.6: Correlation between mean metric grades (color) over all datasets by number of hand-written few-shot exemplars ( $H$ , x-axis) and number of bootstrapped few-shot exemplars ( $B$ , y-axis).

**narratives.** We also see a subtle trend in Table 8.4 and Fig. 8.6 of bootstrapped exemplars improving accuracy and completeness over purely hand-written exemplars. This may be because bootstrapped exemplars must achieve perfect accuracy and completeness scores to be passed to the NARRATOR, therefore removing the chance of passing in exemplar narratives that are either inaccurate or considered inaccurate by the GRADER.

For further investigation, we report the mean metric grades by dataset for the two highest-performing few-shot settings ( $H = 1, B = 1/3$ ) in Table 8.5. We see unexpectedly low scores for Mush 2 and PDF datasets. Inspecting the narratives generated more closely suggests a few possible issues.

**Some narrative styles require additional information to be generated and graded.** The low accuracy for the PDF 2 dataset may stem from the exemplar narratives for this dataset using relative terms like “the larger file size”; the GRADER does not get information about the distribution of these features, so it may be labeling these terms as inaccurate. This suggests that if narratives with such comparative words are desired, additional information should be passed in.

**Further investigation into the GRADER quality on diverse datasets is required.** The low scores on the Mush 2 dataset and the lower completeness score on the PDF 1 dataset appear to stem from an error with the GRADER rather than the NARRATOR. For example, for the Mush 2 dataset, the narrative *The lack of odor and broad gill size suggest the mushroom is more likely to be poisonous* from the explanation (*odor, none, 0.15*), (*gill-size, broad, 0.06*) was incorrectly given a 0 on accuracy. It is possible that the LLM is “playing it safe”, given the high-stakes application of mushroom toxicity, as a result of AI safety work (this problem may not affect Mush 1, as its narratives include safety-conscious additions like “be careful” or “confirm with external sources”). However, this is speculation that requires



Table 8.5: Narrative quality scores generated by our NARRATOR in our two highest-scoring conditions ( $H = 1, B = 1, 3$ ), averaged by dataset. Values greater than 3.5 are in blue, values less than 3 are in red.

Dataset	Accuracy	Completeness	Fluency	Conciseness	Total score
House 1	3.733 ± 0.40	4.000 ± 0.00	3.800 ± 0.30	3.719 ± 0.24	15.252 ± 0.58
House 2	4.000 ± 0.00	4.000 ± 0.00	3.911 ± 0.11	3.836 ± 0.16	15.748 ± 0.24
House 3	4.000 ± 0.00	3.689 ± 0.27	3.933 ± 0.10	3.869 ± 0.14	15.491 ± 0.35
Mush 1	3.556 ± 0.42	3.600 ± 0.00	3.511 ± 0.15	4.000 ± 0.00	14.667 ± 0.33
Mush 2	1.760 ± 0.88	2.640 ± 0.36	3.920 ± 0.18	3.989 ± 0.03	12.309 ± 0.67
PDF 1	4.000 ± 0.00	2.400 ± 0.00	4.000 ± 0.00	3.977 ± 0.03	14.377 ± 0.03
PDF 2	0.000 ± 0.00	3.040 ± 0.22	3.840 ± 0.22	3.949 ± 0.02	10.829 ± 0.46
Student 1	4.000 ± 0.00	3.600 ± 0.28	3.960 ± 0.09	4.000 ± 0.00	15.560 ± 0.36
Student 2	3.840 ± 0.36	3.920 ± 0.18	4.000 ± 0.00	3.880 ± 0.15	15.640 ± 0.47

further investigation.

In the next section, we discuss our key takeaways from these results further.

## 8.7 Discussion and Future Work

This work represents a first step into transforming ML explanations into narratives, and suggests several future directions. We determined that powerful modern LLMs are capable of creating high-quality narrative versions of SHAP explanations. Balancing hand-written and bootstrapped exemplars, while keeping the number of these low, appears to create the best balance between correctness (accuracy and completeness) and style (fluency and conciseness). Adding additional exemplars can improve style at the cost of correctness — users in different domains may value these metrics differently, and wish to modify the number of exemplars accordingly.

However, certain subtle issues — such as including comparative terms like “larger” or working within complex domains — confuse our GRADER into scoring narratives incorrectly. Without a highly-accurate grader, it may be difficult to generate narratives with enough trust for high-risk domains. Therefore, further investigation is needed.

### 8.7.1 System Integration and Open-Source Implementation

We integrated the findings from this paper in Pyreal, a library for transforming explanations for readability. We implemented the NARRATOR in the form of the `NarrativeTransformer`<sup>6</sup> object in this library, an object that transforms explanations into narratives given some hyperparameters including style samples.

Pyreal can be used to both generate the initial explanation and then transform it into narratives. Fig. 8.7 shows a sample of the code to generate a SHAP feature contribution

<sup>6</sup>Instructions for transforming explanations using Pyreal can be found here: <https://dtaill.gitbook.io/pyreal/user-guides/using-applications-realapps/narrative-explanations-using-llms>

```

1 from pyreal.transformers import NarrativeTransformer
2 from pyreal import RealApp
3
4 context = "The model predicts house prices"
5
6 # Explanation, narrative pairs
7 examples = {"feature_contributions":
8             [("size, 120, -120", "The model...")]
9            }
10
11 narrative = NarrativeTransformer(
12             num_features=3,
13             context_description=context,
14             training_examples=examples)
15
16 app = RealApp(model, transformers=[narrative])
17 app.produce_feature_contributions(X, X_train)

```

Figure 8.7: Open-source library implementation of narrative generation process. The output of the final line is a dictionary with keys as row IDs in X and values as narratives of that model prediction on that row.

explanation and transform it to narrative format.

## 8.7.2 Lessons from Adjusting the GRADER

In past work [90], we suggested that the accuracy metric be graded on a 3-point scale of *inaccurate*, *accurate but misleading*, or *accurate*. However, discussion and experimentation revealed that “misleading” in this context is a highly subjective specifier, which makes it difficult for both human and LLM graders to give consistent grades. We therefore settled on a boolean 0 or 1 grade, which was sufficient for our purposes.

We originally intended for the fluency metric to broadly describe the degree to which the narrative sounds natural or human-written, fine-tuned from a general-purpose crowd-generated exemplar dataset of narratives on a wide variety of datasets. However, we determined from discussion with diverse ML users that preferred narrative style is highly subjective and context-dependent, and therefore the fluency metric would be far more useful if based on a small, application-specific exemplar dataset.

## 8.7.3 Future Work

In this work we focused on SHAP contributions; however, by adjusting the input explanation format and some details in the metric rubrics, this work could easily be updated to support other kinds of explanations.

While in this paper we focus on automated grading, a natural next step would be to evaluate how effective the narratives are for real-world decision making.

It is essential to emphasize that this work represents only a first step toward enabling explanations that offer deeper insights and greater clarity. Our ultimate goal is to develop a system that not only generates understandable narratives but also enhances users’ ability

to make informed decisions based on those narratives. This goal requires future research focused on providing richer context and deeper insights in the explanations, allowing users to engage more critically with the information presented. One example of such insights includes rationalizing model behavior; for instance, an explanation narrative of “This house is predicted to sell for more because it was built in the year 1930” can be rationalized as “In this market, older houses hold a premium because of their better building materials and unique history.” In cases where the model uncovered unexpected patterns in data, such rationalizations can be valuable for knowledge discovery and decision-making.

Additionally, we encountered some challenges where our system could not generate and correctly grade comparative narratives (such as referring to a file as “larger” or “smaller”) because we did not pass in the necessary context information. To address these issues, we plan to explore methods for clearly defining data context. For instance, we could pass statistical benchmarks such as mean feature values or more detailed feature distributions to the LLMs. Developing explanations that correctly use comparative terms may improve narrative fluency, and help users understand why certain characteristics, such as file size, are indicative of potential outcomes, ultimately enriching the narrative quality.

We determined that finetuning approaches are not required to develop an effective NARRATOR with a sufficiently powerful base LLM. However, finetuning may be beneficial to allow smaller architectures to perform the task as well as the larger models we worked with. As a first step to trying this, we applied the same final prompt (with 3 labeled few-shot exemplars) to a smaller, 7B parameter local model (Mistral-7B-v0.1) to determine how well it could perform out-of-the-box. The model effectively matched style, with average fluency and conciseness scores of 3.91 and 3.81, respectively. However, the small NARRATOR did not always parse the input explanation correctly, resulting in lower accuracy and completeness scores of 0.98 and 1.64 respectively. Since smaller, local models are more practical for many real-world applications, we plan to refine prompts and explore finetuning to improve these results.



# Chapter 9

## Discussion and Future Work

The work in this thesis revealed many lessons about the challenges involved in developing usable ML for use in real-world decision problems. In this chapter, we summarize these lessons, and discuss further extensions to this work.

We begin by summarizing the key lessons from this thesis. We then dive into more depth on each of these lessons. We then discuss some ethical considerations raised by this thesis. Finally, we go through a few possible extensions of this work.

Here, we summarize the key lessons that we believe will be valuable to future researchers and developers working on usable ML.

**Lesson 1: The first step to making ML usable is clearly identifying the decision-makers, answering the question *usable for whom?*** Different users have very different skills, preferences, expertise, and constraints. Identifying who will be using ML for decision making, as well as who is available to help this group, affects how ML tools are built and deployed.

**Lesson 2: There are many properties of decision problems that affect how decision-makers will use and interact with ML.** Categorizing these properties into a formal taxonomy may speed up future application work.

**Lesson 3: Rich insights can be found through applying systems work to real-world applications, beyond those learned through isolated lab studies.** This process can take years, and therefore should be started early by researchers entering the field of usable ML.

**Lesson 4: Progress is led by building systems to address known challenges — the remaining, socio-technical challenges will surface once effective systems are in place.** Effective, configurable systems like Pyreal and Sibyl that create tools that avoid rudimentary usability challenges allows researchers to avoid repeating work. Instead, more time can be spent observing decision making with ML in real applications, and gleaning lessons from that process.

**Lesson 5: ML usability is an important concept to consider that goes beyond**

**ML explainability.** Not all applications require decision-makers to understand how the model works, and most decision-makers require richer or more curated information than just explanations.

**Lesson 6: Type 2 decision problems come with unique challenges relevant to usable ML beyond those present in Type 1 decision problems.** The existing human-only workflows, complexity of evaluation, and high-risk nature of these decision problems must be addressed carefully.

In the next few sections, we will discuss each of these lessons in more detail.

## 9.1 Lesson 1: Identify the decision-makers and users of ML first

Subtle differences in the decision-maker audience for ML tools can lead to significantly different requirements. It is therefore essential to be aware of who will be using these tools, and for what purpose. For example, child welfare screeners, integrating a risk score prediction as one of several sources of information to decide what decision to make on a highly sensitive and high-stakes decision treat ML models as data filters and feature flags. In contrast, wind turbine data analysts see usable ML more as a way to filter out key entities (turbines), and appreciate a focus on summarizing model predictions.

We also suggest learning whether anyone holds the increasingly ubiquitous bridge role — bridging the gap between decision-makers and ML developers — to take advantage of the assistance they can bring to communication and collaboration. In our child welfare application, social scientists played a valuable role, translating early discussions of possible explanations so that they made sense to child welfare screeners. In our wind turbine application, this role was played by data analysts. In both cases, the ML developers were able to spend less time learning the nuances of the domain, and more time deploying and evaluating ML tools, thanks to the diverse expertise of these bridges.

Feedback is a gift — one that takes time and effort from busy people. It can also be misdirected. Feedback should be sought early and often to avoid going down the wrong path, but solicited in a way that minimizes effort and maximizes quality. We learned that asking decision-makers what kind of information they want is often an unhelpful exercise, as people unfamiliar with ML may not understand the scope of what is possible, and may have a hard time imagining ML augmentations they have no experience with. On the other hand, asking for open-ended feedback on ML developer-provided suggestions also often does not work — it is easy for decision-makers to feel that any information they provide will be useful, causing these feedback sessions to lead to generic “yes we like it” responses. We believe there are three primary reasons for this. First, decision-makers lack motivation to turn down the potential for more information when they know they can always simply not use. Second, it is difficult to imagine using information that you have never had access to before. Finally, it takes time and effort to synthesize open-ended feedback, and decision-makers are often busy with their own jobs.

Instead, we recommend first gleaning feedback indirectly through careful observation of processes, and then directly through asking decision-makers to actively complete tasks using ML tools. This active use brings challenges to the surface in a more natural way.

## 9.2 Lesson 2: Many properties of real-world decision problems affect how ML is used

The real-world applications in this thesis revealed a wide range of axes that complex, Type 2 decision problems vary on beyond the people and process summaries introduced in Table 1.2. In this section, we summarize this taxonomy for further categorizing Type 2 decision problems, and the lessons learned from the applications in this thesis for usable ML in these scenarios.

We will begin by revisiting some of the categorizations introduced in Table 1.2, formalizing the lessons learned.

1. **Risk associated with individual decisions:** High risk decisions result in more usability challenges, as decision-makers look at the ML outputs with more caution. Higher risk decisions — such as child welfare screening — may have more additional required information in augmentations, though ML developers creating ML tools must still be careful not to present too much information. Lower risk decisions may not require augmentations and ML tools at all.
2. **Time required to make a decision:** The amount of information users are willing and able to process scales naturally with the time available to make decisions. In the child welfare application, decision-makers had significantly less time to make decisions than in the wind turbine monitoring application (minutes compared to hours or even days), and therefore the expected augmentations needed to be significantly more precise with the information presented.
3. **ML/data expertise of decision-makers:** Decision-makers with less ML/data expertise not only benefit from explanations and augmentations presented using interpretable features, but also subjectively prefer them; in contrast decision-makers with high ML/data expertise, such as those in our Pyreal user studies are more likely to prefer explanations that use more typically “model-ready” engineered features.
4. **Domain expertise of decision-makers:** Decision-makers with more expertise in the decision-making domain (for example, child welfare screening or real estate valuation) display higher standards for ML tools, and therefore require more carefully configured and customized tools.
5. **ML Integration:** In Type 2 decision problems, the ML model is usually used by human decision-makers, who make the final decision. However, there are two major categories of ways this can happen — the model can make a *direct suggestion of action* or provide *proxy information*. For example, for the child welfare screening application, the former scenario would mean that the ML output would take the form of a

Boolean “screen-in/screen-out” prediction; in the latter, the model returns a risk score representing the likelihood of abuse. Models that provide proxy information may be more common in complex applications as the final decision is too complex to process with ML, and require more detailed ML tools to allow decision-makers to translate the proxy information into a decision.

We now introduce several new axes that we discovered carried heavy importance when designing ML tools across diverse applications.

1. **Intrinsic decision-maker trust in ML:** Some decision-makers, such as the wind turbine data analysts, have a natural appreciation and trust in computational tools such as ML — this does not mean that trust will not be a usability challenges in these domains, but that less caution is needed when iterating on and evaluating ML tools. In the child welfare application, our decision-makers had a (justified) concern with using ML tools, which increased the chance that a half-baked or confusing prototype tool would turn them off using the ML outputs entirely.
2. **Ethical/personal significance:** Even across applications with high risk, the nature of that risk varies. In the wind turbine application, there was significantly less humanitarian and ethical concern than in child welfare screening — decisions were made about turbines rather than humans. This difference resulted in differences in appropriate augmentations — for example, augmentations like nearest-neighbors style comparisons of entities may be less appropriate when those entities are humans.
3. **Complexity of input data:** In some applications, the ML model considers all or most of the data available to the decision-makers — the wind turbine monitoring application was closer to this scenario. In others, there are large amounts or kinds of data not considered at all by the model — in the child welfare screening application, the hotline narrative was an essential piece of information not at all considered by the model. The more complex the input data, and the more data not considered by the model, the more effort decision-makers have to put in to manually combine insights from different sources of data. In our simulated ML tool scenario, the child welfare screeners regularly cross-referenced the narrative and the information in the ML tool to ensure everything was consistent and covered by the model.
4. **Primary purpose from ML:** For some decisions problems, ML is integrated primarily with the goal of improving decisions, potentially in applications where outcomes were deemed to be unsatisfactory. For example, ML may help highlight missed information or reduce bias, as were goals in the child welfare screening. In other applications, such as some ML for conservation work [110], the ML model serves mostly to reduce decision-makers effort required, by removing the need for some of the manual data processing steps. Decision-makers may be more receptive and trusting of model in the latter case.
5. **Ease of ML evaluation:** In some applications, it is possible to effectively quantify the quality of outcomes and the performance of the ML outputs through well-defined key



performance metrics (KPIs). In other applications, this outcome is much more nuanced and difficult to quantify, and the performance of the ML model is less meaningful. The easier the model is to evaluate — assuming the model achieves high performance — the less important it may be to build decision-maker trust in the model.

Identifying where a decision problem sits on these axes before developing ML tools can improve the efficiency and effectiveness of the ML.

### **9.3 Lesson 3: Research in real-world, long-term applications uncovers rich insights**

Conducting academic research within real-world, long-term applications uncovers more and deeper insights than lab-based research alone. We advise future researchers in usable ML to decide on one or two real-world decision problems early on, and continuously apply findings to these problems.

We uncovered many insights related to making ML usable through our work in real-world applications. In our child welfare application, we learned from the existing decision process that screeners focus on risk and protective factors, often visualizing these side-by-side to weigh their respective influences on the case. This was an excellent analog to feature contributions, labeled as risk and protective factors accordingly; presenting explanations in this way made it easier for screeners to integrate the new information into their process. Understanding these existing processes in child welfare screening also revealed that making comparisons between cases and children in order to aid with decision-making is not done, despite this being technically possible without any computational component, as past work has already determined this to not be a good approach. We were able to integrate that lesson and avoid wasting time evaluating nearest-neighbors-based interfaces.

In our wind turbine monitoring application, we learned that our collaborators were often as or more interested in wide-scale trends as they were in individual turbines, and were especially interested in understanding where failures were predicted. This understanding of the process led us to develop interfaces that put extra focus on global explanations and prediction summaries.

Evaluating usable ML through collaborative efforts is the best way to get genuine information about its impact. These evaluations happen both through simulated studies, like the ones we ran in this thesis, and through in-deployment studies. The balance of these depends on the risk associated with the decision problem, as well as the availability of decision-makers.

### **9.4 Lesson 4: Progress in usable ML is led by building systems to address known challenges — deeper challenges are discovered after**

Systems like the ones developed in this thesis make the real-world applications described in the previous section practical. Certain, generalizable requirements of usability — like

ensuring terminology is consistent with the domain, ensuring interactions are smooth, and ensuring the cognitive load remains reasonable — will be required across more applications, and if not met, will prevent tools from being used at all. Once these general challenges are addressed, deeper socio-technical challenges will become apparent through studies in real-world applications and can then be addressed separately.

For example, in the child welfare application, concerns about how the model worked and the information it used — such as using features about whether families receive food benefits — as well as deeper findings about how information should be presented — such as avoiding redundant information such as child being an infant and not having a criminal history — were only brought to light once explanations were presented in an interpretable format.

## 9.5 Lesson 5: Consider usability, not just explainability

In this thesis, we discuss ML *usability* rather than *explainability*, even though many of the augmentations we developed were built on explainable ML literature. Our findings investigating real-world decision problems suggested that usability is a more important property for decision-making than explainability. While explainability is often a key component of usability, it is often insufficient, and can even be unnecessary.

For some decision problems, explainability is not required, even when humans actively use ML outputs. If the decision problem is straightforward and well-validated, decision-makers may get everything they need just from the ML outputs. They may still need other kinds of augmentations, as discussed below, but not specifically explainability. For example, consider a public relations employee at a company. This employee uses a sentiment analysis model to filter out especially positive and negative social media posts about the company, and then analyzes these to understand trends in customer opinion. Especially for short posts, the employee may not need any explanation to understand the prediction; the model is very useful as a filter, but the decision-maker is able to immediately understand what makes a post positive or negative themselves, possibly better than the model.

In many other domains, explainability is not enough for ML outputs to be effectively used for decision-making, especially when the decision problem is complex and risky. For example, explanations cannot be used to ensure ML models lack bias, as biases may be hidden across proxy features that a human would not recognize. Fairness metrics that evaluate ML outputs and performance across groups are more effective at this task. Explanations are less useful than extensive evaluations for understanding broad ML model performance. The explainable ML literature also focuses on explaining the model’s logic, but understanding the input data itself may be equally or more important. For example, in our child welfare application, child welfare supervisors found historic data trends, and how they associated with ML outputs, to be more useful for validating the model than traditional ML explanations were. In many cases, a *justification* — or basic information that helps a model validate its decision — is much more helpful than a full explanation of the model’s logic.

Explanations may also be misleading. For example, in our child welfare study, counterfactual explanations incorrectly implied real-world causality. Additionally, when features are heavily correlated, some explanation algorithms like SHAP may arbitrarily label one as important and another as unimportant, which can lead to incorrect conclusions about how

features correlate with the ML output.

With the advent of large language models (LLMs), the meaning of explainable ML has shifted. Many large models cannot be explained well using traditional explainable ML algorithms, simply because there are too many parameters and the complexity of the underlying logic is too high. Still, these models can be highly usable, even providing their own form for explanations through self-rationalization.

Usability encompasses the overall ability for humans to factor ML outputs into decision making. Any time a decision-maker uses ML outputs for their work, they need usability. This includes explanations, but also many other factors.

For example, ML *usefulness* describes the extent to which the ML outputs actually provide useful information. For both the child welfare and wind turbine case studies, the modeling teams worked hard to experiment with different outputs to identify which correlated best with real-world outcomes. A useful model must perform sufficiently well, and decision-makers must understand that performance. They must know the performance is sufficient for them to trust and use the ML outputs, and they may also need to understand the shortcomings of the model’s performance to understand when it should be ignored. Basic ML performance metrics like  $R^2$  scores, *AUC*, and *MSE* are often not sufficient for non-technical decision-makers (or even technical decision-makers) to fully understand performance. Relatedly, decision-makers may need metrics describing a model’s prediction uncertainty, which can also help reveal when to follow the model’s guidance and when to defer to human intuition.

Overall, our case studies confirmed that explainability is just one component of ML usability, and deploying ML for use in Type 2 decision problems requires careful consideration of the full range of usability requirements.

## 9.6 Lesson 6: The parameters of Type 2 decision problems come with unique challenges

In this section, we revisit the definition of Type 2 decision problems we gave in the introduction, adding a deeper discussion of the challenges and requirements of such problems based on lessons from this thesis. We refer the reader to Figure 1.1 in the first chapter for a reminder of the properties discussed.

The complex impact and long impact timeframe for Type 2 decision problems work together to make evaluating ML tools especially difficult in these cases. Evaluating the impact of ML tools requires going through multiple rounds of evaluation, including both simulated and real-world evaluations in deployment. Therefore, in-deployment evaluations may be the only way to fully understand the impact and usability of ML tools; however, especially when risks are high, these can pose safety issues. Multiple metrics based on real-world KPIs must be tracked, capturing the full range of impact the model may have on the full range of affected individuals. The traditional performance metrics referenced in the ML literature become less sufficient as the number of potential sources of impact increases. In addition, computing these metrics can be difficult when factoring in the impact timeframe, adding time during which confounding variables can complicate evaluations.

These evaluation challenges mean it is especially important to develop systems that reduce the time, effort, and errors involved in developing usable ML tools. This prevents time from being wasted developing and tuning systems, so that all available time and effort can be focused instead on evaluation and fine-tuning. These challenges also make it especially important to take time to fully understand the people and processes involved in the decision problem, which will help shed light on what KPIs should be tracked.

The higher risk associated with Type 2 decision problems makes it especially difficult and undesirable to try to remove the human decision-maker. For example, one concern brought up during the child welfare case study was the added pain of families being investigated by child welfare based only on an algorithm, without human involvement — in such domains, even if the model could perform on par with humans, humans may still play a vital role. As such, in high-stakes domains, ensuring we have systems in place to support the development of usable ML tools will remain essential. Additionally, the high risks associated with Type 2 decision problems makes the other components of usable ML discussed in the previous section especially important — for example, the ability to verify models through uncertainty measures, performance metrics, fairness metrics, and other more empirical measures.

The existence of equivalent human-only decision processes was especially valuable for the work in this thesis. These were vital resources for us as we thought through real-world applications for our tools. Our ability to understand the people and processes involved offered a jump-start for developing ML tools, as we could begin by mirroring these existing processes. Sibyl’s emphasis on diverse configuration and interaction options is meant to support the diversity in existing processes, and allow ML tools to be adjusted to fit these workflows.

More work is required to fully meet the requirements necessary for application to Type 2 decision problems; some possible avenues are discussed in later sections of this chapter. The evaluations in our case studies did not entirely capture the complex and delayed impact of ML tools; longer-term deployments are required. The work in this thesis offers a first step towards this goal.

## 9.7 Ethical considerations posed by this work

The work in this thesis poses several important questions related to ethics in usable ML. For the most part, we do not have answers to these questions. Rather, here we pose the questions we believe should be considered by future researchers working in usable ML, and discuss how they were relevant to the work in this thesis.

**When, if ever, is it acceptable to hide elements of explanations, or modify elements of explanations, such that they improve the decision-makers’ ability to understand and use the explanations?** In both our child welfare and wind turbine applications, some elements of the model’s logic were confusing, even though there may be a good reason this model uses this logic. For example, child welfare screeners were confused to see that the model considered both the fact that a child is an infant and that they had never gone to juvenile detention, because including both factors is redundant. Wind turbine analysts were confused as to why different aggregations of the same features could have significantly different impacts on the model. Sometimes, re-training the model to behave in

a more interpretable manner can be effective; in other cases, this is not possible or would significantly reduce model performance. If showing accurate explanations reduces users' usage of high-performing models, it may be that modifying what parts of explanations we show improves decision-making and therefore real-world outcomes. To what extent is this an acceptable strategy?

**In the process of developing ML tools, at what stage or level of validation do we consider deployment?** This question involves balancing two opposing thoughts: 1) deploying ML tools that are not fully validated could cause decision-makers to misuse the ML outputs and lead to worse outcomes. On the other hand, 2) in high-risk decision problems, if there is good reason to believe these interfaces would greatly improve decision-making, how long can you ethically withhold them to run validations? This is not a problem unique to usable ML — medicine development is a classic analog. These questions become especially important in Type 2 decision problems, where fully validating ML outputs can take a long time, but even minor decisions that could have been improved with explanations can have major effects on individuals. For example, in the child welfare application, supervisors expressed concerns around running an A/B study where only half the screeners would get the ML tool, as this was seen as withholding potentially valuable information.

**Does explainability play a role in respect for decision-makers?** Can withholding explanations constitute a potential form of disrespect towards the decision-makers, even in situations where we believe providing these explanations could reduce decision-making quality? Is hiding information that was deemed to be misused patronizing, or is showing confusing information the more disrespectful option? Much literature looks into whether explanations are useful for decision making, but there is an argument that if the decision-makers feel they need them, this is reason enough to provide them.

We consider these to be questions to be especially important when addressing Type 2 decision problems, and encourage future researchers to consider them carefully.

## 9.8 Future Extensions

In this section, we introduce a few recommended next steps for future researchers looking to advance the field of usable ML.

### 9.8.1 Formal Deployment Evaluation

Due to resource and time constraints, this thesis does not include evaluations in-deployment. Such evaluations can be difficult to run in academic environments for several reasons.

First, deployments come with extensive infrastructure requirements, a hurdle that can be difficult to cross from within an academic setting. Deployments need to be set up and run on infrastructure that the original ML developers may not be allowed to directly interact with.

Second, a commitment from some party to maintain the deployed ML tools is required — this was the roadblock that arose in our child welfare application. This requires either finding a new individual that understands the system and is able to commit to maintenance, or building the system using lightweight or familiar frameworks such that it can be maintained

by the existing team. In the child welfare application, our original system was built in React, a framework not familiar to the county’s IT team. Had we identified this challenge earlier, we may have instead created a Tableau front-end, which would have been easier to deploy.

Both these challenges further motivate the development of lightweight usable ML systems, which make it easier for a wider group of people to set up and maintain interfaces on a wide variety of platforms.

Finally, the evaluations themselves must be done in a lightweight manner. Academic-style lab experiments in rigorous, controlled environments are difficult to run within real decision problems; such studies require extensive time and effort from industry collaborators outside of their normal work. Therefore, evaluations in-deployment — as are more common in industry — must be run, including techniques like A/B testing and shadow deployment. This leads to a less scientifically rigorous but more practical and useful evaluation framework.

In the future, we hope to take advantage of Sibyl’s lightweight design to deploy interfaces and evaluate their influence on real-world outcomes.

## 9.8.2 Adding Data Modalities

In this thesis, we focus on tabular data. This is a popular data modality in many real-world deployments, and allowed our algorithms and systems to stay focused and achieve full comprehensibility within this scope. Additionally, feature engineering is popular for tabular data, making it relevant for systems like Pyreal. Future work can extend Pyreal and Sibyl to support other data modalities, such as images, vision, and text.

One possibility is to specifically address applications where raw data like images or time series are featurized, and transforming explanations to be represented back to the original, raw format. We already took one step towards this as part of the VBridge project [41], which presented feature contributions in terms of the original time series steps. The equivalent of this approach for images would be to show how much each part of an image correlates with each created feature, and use this information to distribute feature contributions back into the original image. Doing so may create more interpretable versions of existing saliency maps, as they can include more information as to why each pixel in an image was important based on the feature that it contributed to.

# Chapter 10

## Conclusion

Ensuring ML models, explanations, and other augmenting information are usable is an essential part of offering real-world benefit to Type 2 applications. In our investigation into using ML for child welfare screening, we discovered several key challenges. Decision-makers struggled to understand explanations using complex, uninterpretable features, and required usable ML tools to be specially configured to match their existing language and workflows. We also saw immense benefit in evaluating these tools within the framework of a real-world decision problem.

Motivated by these shortcomings, we developed taxonomies and systems to aid the usability of machine learning, and evaluated their impact in real-world applications. Our taxonomy for interpretable features helps lay the groundwork for producing more usable and interpretable ML explanations and augmentations. Pyreal, our system and corresponding Python library implementation, builds on this taxonomy by producing explanations that use these interpretable features. Our studies demonstrated that Pyreal significantly reduces the time taken and errors made when generating interpretable explanations, which are deemed more useful by users than the traditional outputs from existing explanation libraries.

We then developed Sibyl, our full-stack system for developing and configuring usable ML interfaces for diverse domains. We used Sibyl to tune a usable ML interface for wind turbine monitoring, a process that was much quicker and more efficient thanks to the configurability of our system. Our work revealed that remaining usability challenges exist in explanations, and that further research should narrow the interfaces down to a maximally useful subset.

Finally, we developed Explingo, our system for further making ML usable by automatically producing and evaluating natural-language or narrative versions of ML explanations.

The work in this thesis revealed that making ML truly usable is a highly nontrivial process, and further work remains to bridge this gap. It also revealed several key lessons to improving ML usability. The requirements of real-world domains diverge greatly from the metrics optimized for in research settings; therefore, long-term, real-world projects are required to make ML usable and fully evaluate the impact of ML models on decision-making.





# Appendix A

## Glossary

In this chapter, we compile all terms used throughout this thesis.

### A.1 People

**receiver** Person who uses ML outputs and ML tools to make decisions and complete their work. Examples from this thesis include child welfare screeners, wind turbine engineers, homebuyers, and real estate agents.

**bridge** Person who interfaces between receivers and ML developers. Examples from this thesis include product managers, social scientists, and wind turbine data analysts.

**ML developer** Person who develops, tunes, and deploys ML models and [ML tools](#) for real-world applications. Experts in applied ML.

**ML contributor** Person who specializes in fields such as ML, AI, explainable ML, or human-computer interaction, and develop general-use ML systems.

**affected individual** Person who is affected by decisions made using [ML outputs](#).

**user** Person who uses a tool or system.

**participant** Person who participates in a user study or other human evaluation.

### A.2 Processes

**ML application** A specific instance of an [ML output](#) or [ML tool](#) being applied to a real-world decision problem.

**decision problem** A task to be completed that involves making a decision — in the scope of this thesis, we specifically focus on decisions made using [ML outputs](#).

**decision process** The full process required to address a [decision problem](#), including considering [ML outputs](#), [ML tools](#), and other external information and incorporating this information in a final decision.

**ML model** A construct that takes in input data and returns some predicted output based on patterns learned from training data.

**ML output** A predicted value from an [ML model](#).

**ML augmentation** Any additional information that is presented alongside [ML outputs](#) to aid with decision-making, such as explanations, data visualizations, uncertainty quantifications, performance metrics, etc.

**ML explanation** A type of ML augmentation that explains or justifies why an [ML model](#) predicted some outputs, or how the model works in general. Local explanation explain specific outputs on specific inputs, while global explanation explain the model's logic in general.

### A.3 Systems and Artifacts

**considerations for usable ML** A set of factors related to the people and processes involved in a [decision problem](#) that should be considered when developing usable [ML tools](#).

**system for usable ML** Software or frameworks used to develop usable [ML tools](#).

**ML tool** A set of user interfaces or other programs that allow users to interact with [ML outputs](#).

**interface** One specific page in an [ML tool](#) that visualizes an [ML augmentation](#).

**explanation algorithm** An algorithm that generates an ML explanation.

**data transform** A function that transforms data between feature spaces.

**explanation transform** A function that transforms explanations between feature spaces.

### A.4 Adjectives

**usable** A property of [ML outputs](#), [ML augmentations](#), or [ML tools](#) such that they are used to address a [decision problem](#) and result in an improved outcome to that problem.

**interpretable** A property of features, explanations, or augmentations such that they can be easily understood by the intended audience.

**configurable** A property of ML systems such as Sibyl, such that they can easily be adjusted to match the diverse requirements of ML applications.

**comprehensive** A property of ML systems such as Sibyl, such that they support creating ML tools with the full range of interactions required by the intended audience.

**lightweight** A property of ML systems such as Sibyl, such that they can be set up quickly and easily without overhead effort.

**extensible** A property of ML systems such as Sibyl, such that they can be easily extended to include new functionality.



# Appendix B

## Pyreal Additional Contents

### B.1 Code for Generating Interpretable Explanation without Pyreal

Figure B.1 shows the equivalent code required to create the same outputs as the code in Figures 5.4 and 5.3. Note that this code only generates one kind of explanation. To generate another, lines 32-50 would need to be modified for the new explanation, accomplishing the equivalent of line 8 in Figure 5.3.

```

1 from sklearn.preprocessing import OneHotEncoder
2 import shap
3 import reverse_geocoder
4
5 X_train = load_training_data() # external function
6 X_to_explain = load_data_to_be_explained()
7
8 def transform_to_x_model(X):
9     x_to_encode = X[["ocean_proximity"]]
10    ohe = OneHotEncoder().fit(x_to_encode)
11    encoded_columns = ohe.get_feature_names_out()
12    ocean_encoded = ohe.transform(x_to_encode)
13    return pd.concat([X.drop("ocean_proximity", axis="columns"),
14                     ocean_encoded], axis=1)
15
16 X_train_model = transform_to_x_model(X_train)
17 X_model = transform_to_x_model(X_to_explain)
18
19 X_interpret = X_to_explain.copy()
20 coordinates = list(
21     X_interpret[["latitude", "longitude"]].itertuples(index=False, name=None))
22 results = reverse_geocoder.search(coordinates)
23 neighborhoods = [result["name"] for result in results]
24 X_interpret["neighborhood"] = neighborhoods
25 X_interpret = X_interpret.drop(columns=["latitude", "longitude"])
26
27 model = train_model(X_train_model) # external function
28 explainer = shap.Explainer(model, X_train_model)
29 shap_exp = explainer(X_model)
30 columns = X_model.columns
31 exp_df = pd.DataFrame(shap_exp.values, columns=columns)
32 encoded_features = [item for item in encoded_columns
33                    if item.startswith("ocean_proximity_")]
34 exp_df = exp_df.drop(encoded_features, axis="columns")
35 exp_df["ocean_proximity"] = exp_df[encoded_features].sum(axis=1)
36 exp_df["neighborhood"] = exp_df["latitude"] + exp_df["longitude"]
37 exp_df = exp_df.drop(columns=["latitude", "longitude"])
38
39 feature_descs = {"med_income": "Median income", ...}
40 exp1 = exp_df.rename(feature_descs, axis="columns")

```

Figure B.1

# Appendix C

## Sibyl Additional Contents

### C.1 Screenshots of Sibylapp Interfaces

Figures C.1 – C.8 show screenshots from various Sibylapp interfaces, with our demo application of house price prediction. These interfaces can also be found and interacted with at <https://sibylapp.streamlit.app/>.

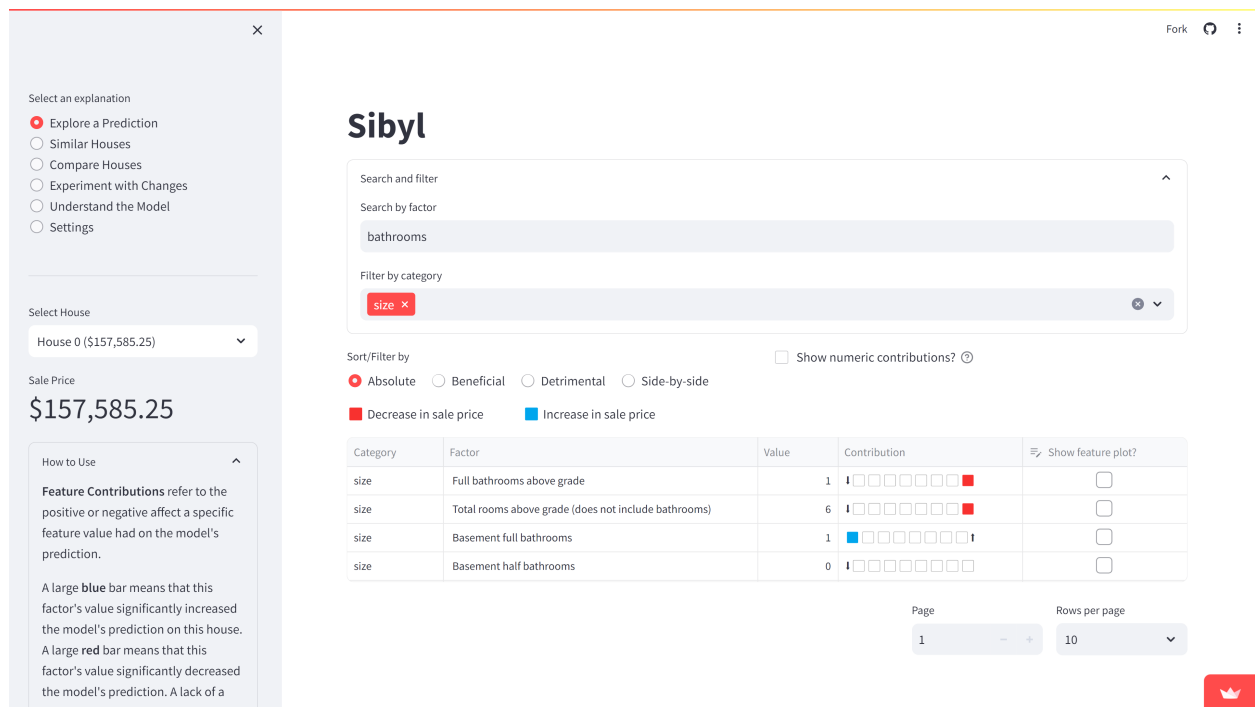


Figure C.1: Overall structure of a Sibylapp tool, including the following components. From top to bottom in the left toolbar: the configured set of interfaces available, the entity selector, prediction for the selected entity, brief instructions on how to use the selected interface. From top to bottom in the main interface space: A search-and-filter box that allows filtering features by name and category, sort options specific to the type of explanation, and the explanation itself.

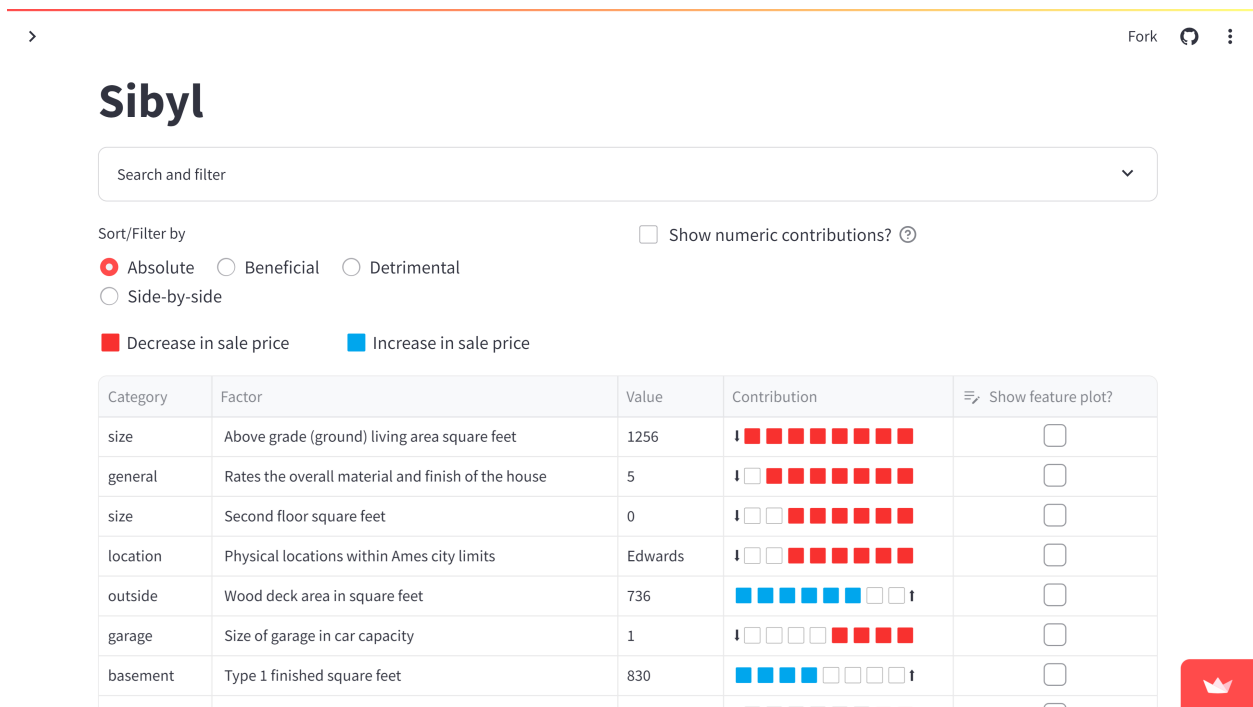


Figure C.2: The main explanation interface contents for the Explore a Prediction interface. This interface shows the positive or negative contributions of each feature to the final model prediction. The right-most column in the table allows users to see more detailed information about how the model uses a specific feature, as shown in Figure C.8



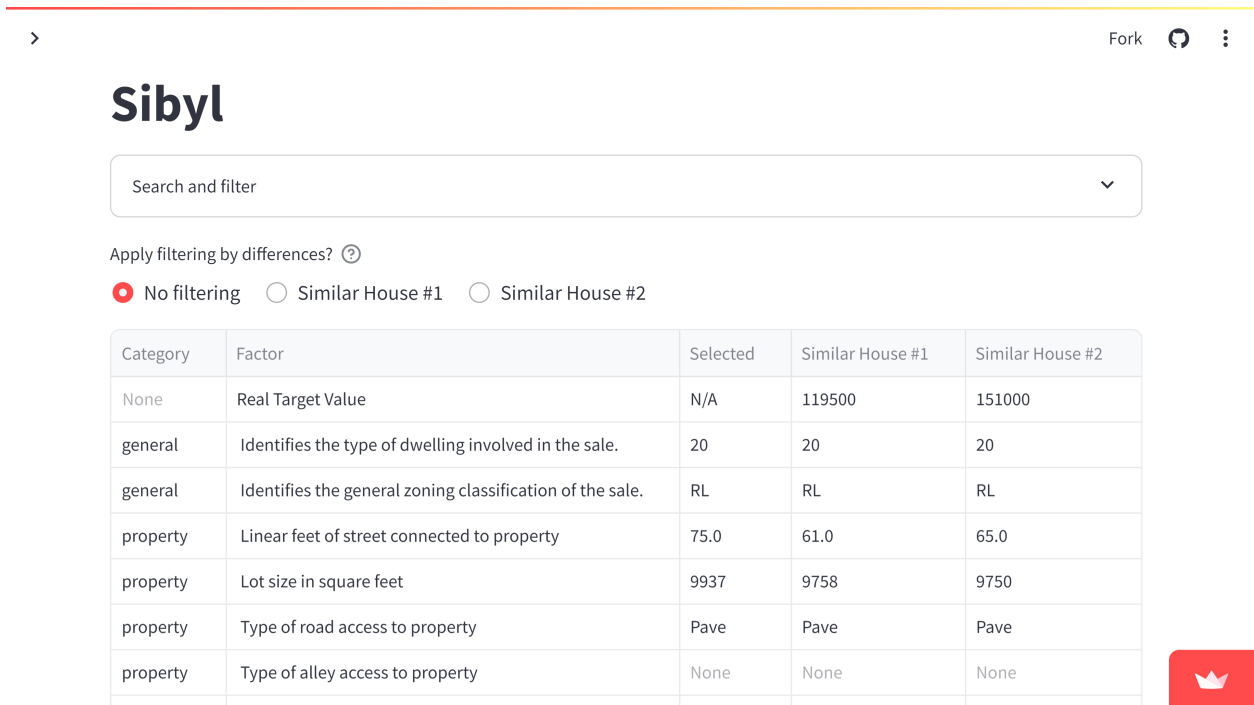


Figure C.3: The main explanation interface contents for the Similar Entities (Houses) interface. The *Selected* column shows feature values for the currently selected entity, while the Similar House columns show the feature values for the two nearest neighbors in the training set. Users can choose to see only the rows that differ between the select and similar houses.

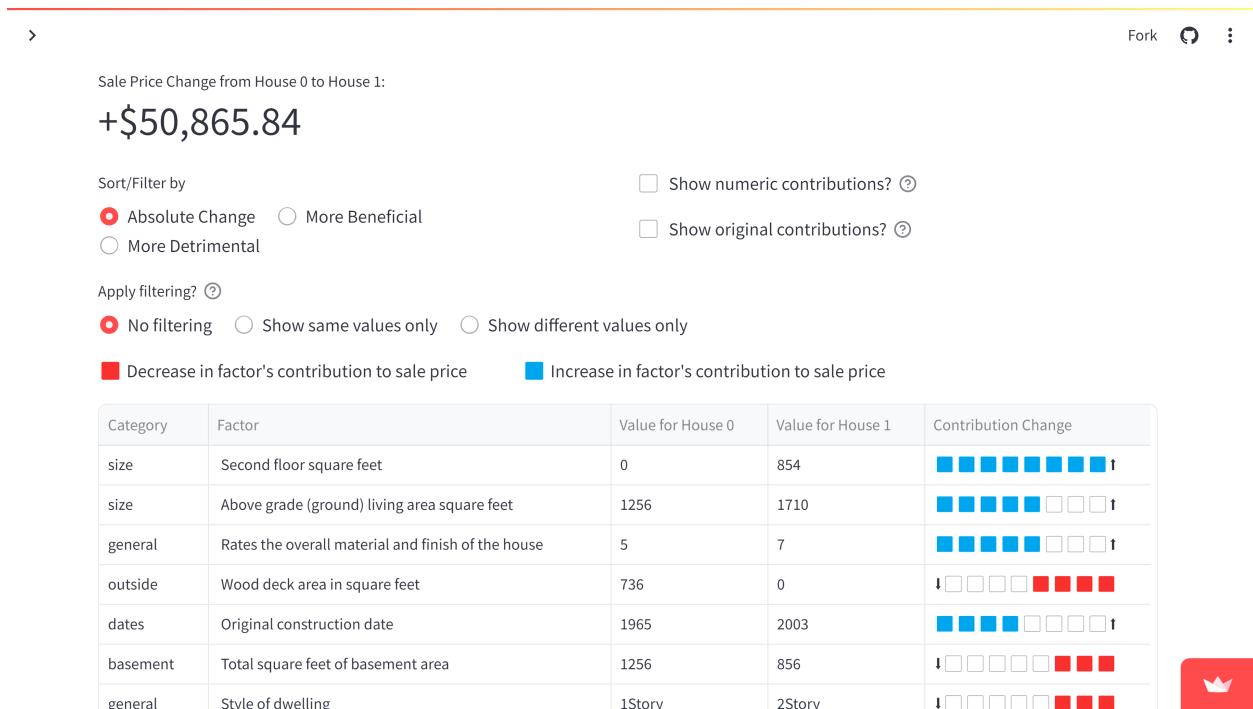


Figure C.4: The main explanation interface contents for the Compare Entities interface. Users select two houses or observations to compare from the left sidebar (not shown), and see the difference in prediction and feature contributions between these two inputs. Here, we can see that House 1 is predicted to sell for around \$50,000 more than House 0, with the primary cause of the increase coming from its second floor, large size, and high quality materials.

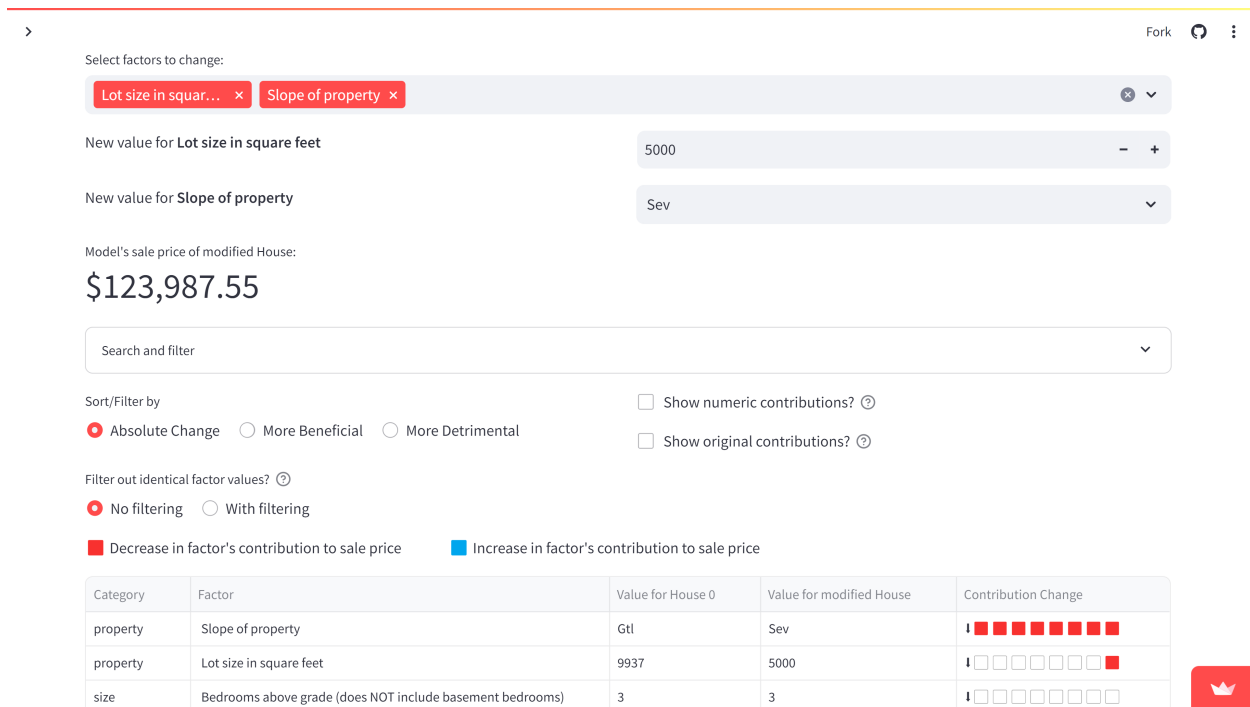


Figure C.5: The main explanation interface contents for the Experiment with Changes interface. Users can select up to five features to modify, and see how the model prediction would be different in this hypothetical scenario. In this example, we can see that if the house was on a property with a severe slope, rather than the gentle slope it is actually on, its predict price would decrease significantly. On the other hand, reducing the lot size by about 5,000 sq. ft. has a relatively small effect.

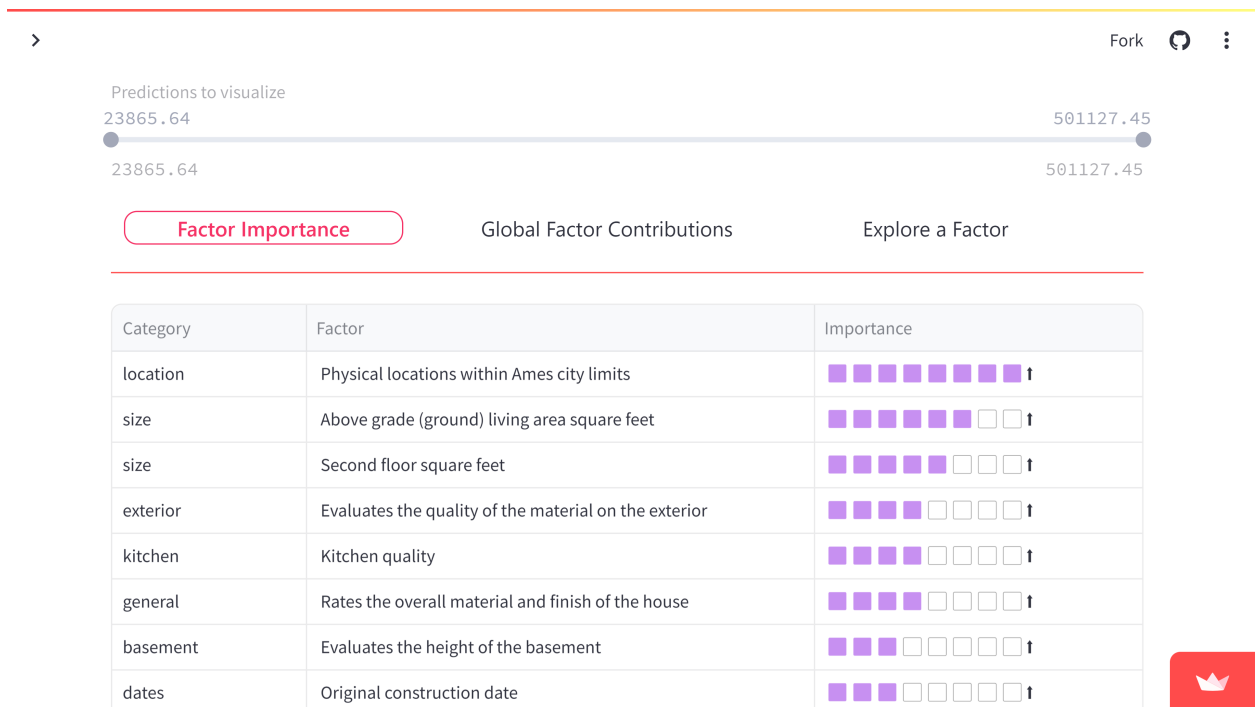


Figure C.6: The main explanation interface contents for the Understand the Model – Feature Importance interface. Each bar shows the overall importance of each feature to the model, sorted in descending order by importance.

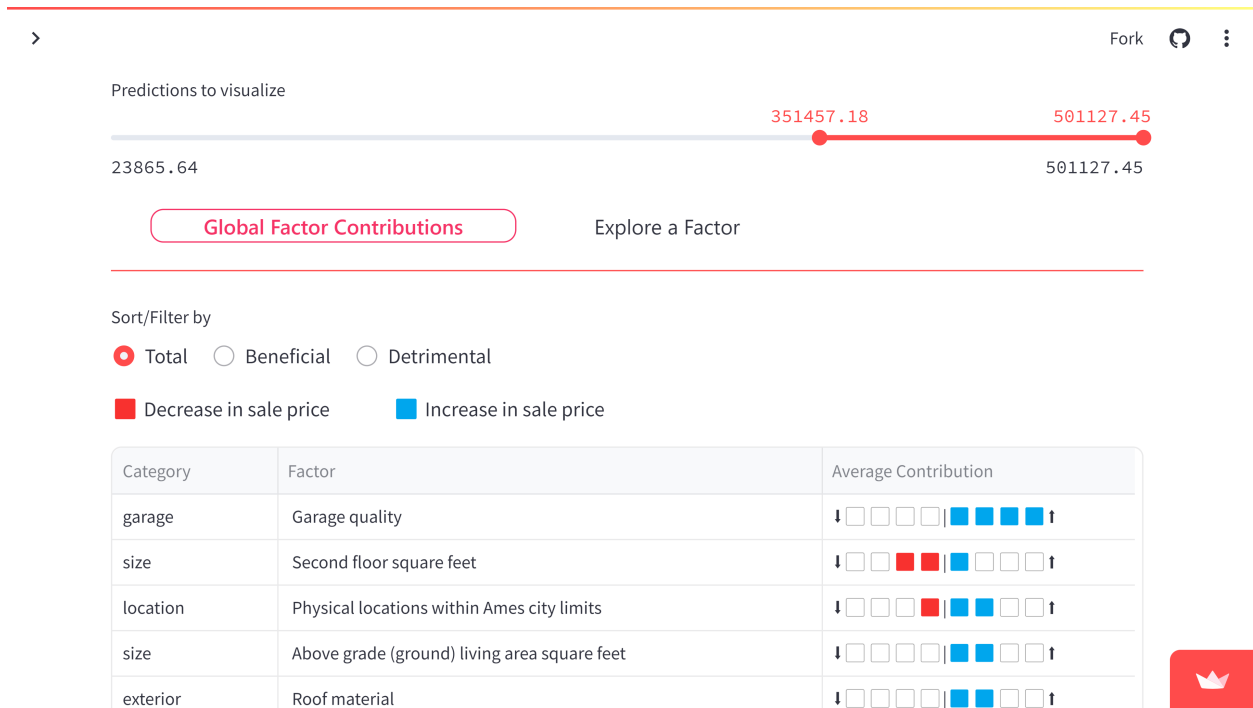


Figure C.7: The main explanation interface contents for the Understand the Model – Global Contributions interface. At the top of this interface, users can select only a specific range of predictions (or set of predictions for classification models) to see contributions for. For each feature, across the full dataset, the positive and negative contributions are visualized separately. For example, we can see here that, among more expensive houses, the garage quality feature often significantly increases the model prediction, but never decreases it. On the other hand, the Second Floor Square Feet feature may increase or decrease house price predictions among more expensive houses, depending on its exact value.

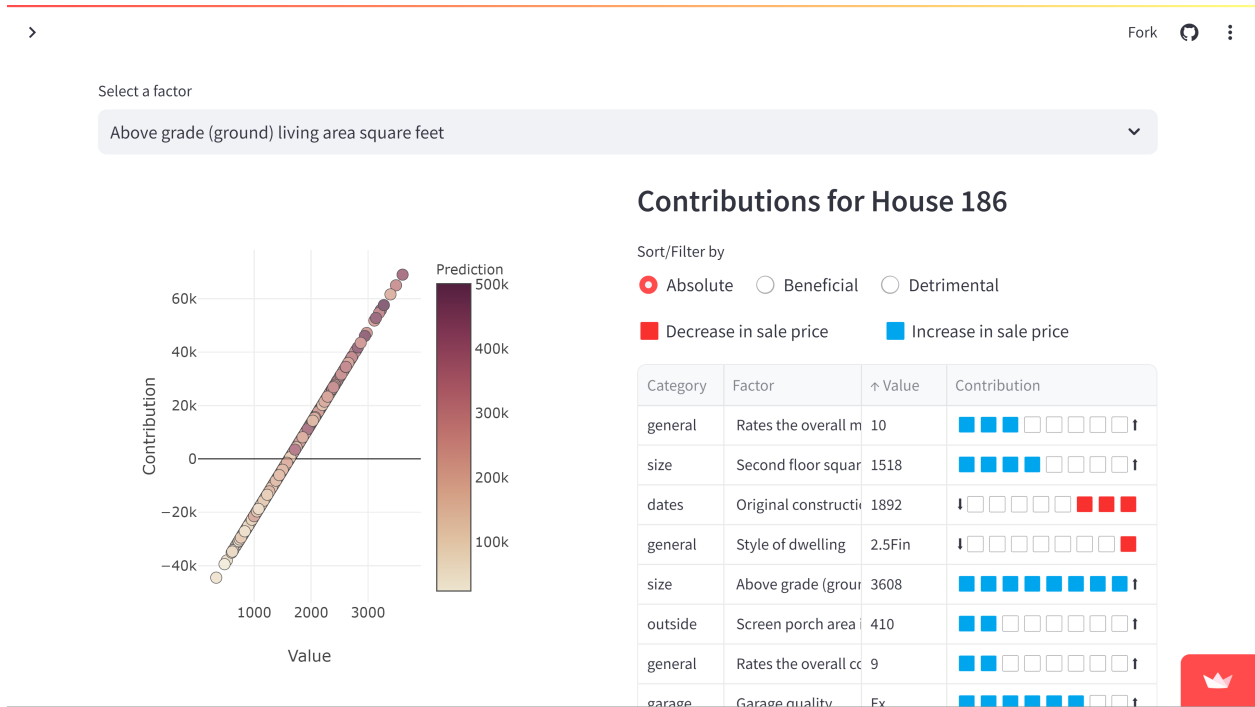


Figure C.8: The main explanation interface contents for the Understand the Model – Explore a Feature interface. This interface shows users how different features contribute to the model overall, across all values in the training dataset. Each point in the plot on the left shows an observation (house) in the training data. The points x-axis represents the feature value, the y-axis represents its feature contribution to the model prediction for that observation, and the color represents the model’s prediction for that observation. In this example, we can see that higher living area values have higher contributions to the model predictions; living area sizes below around 1,700 decrease the model’s prediction, and size above this point increase the model’s prediction. Because the model for this example in a Linear Regression model, we see a clear linear trend.

Users can also select points in this plot to see the full feature contribution table for this entry on the right.

# References

- [1] O. Springer and J. Miler, “The Role of a Software Product Manager in Various Business Environments,” in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*, Sep. 2018, pp. 985–994.
- [2] A. D. Selbst and J. Powles, “Meaningful information and the right to explanation,” *International Data Privacy Law*, vol. 7, no. 4, pp. 233–242, Nov. 2017, ISSN: 2044-3994. DOI: [10.1093/idpl/ix022](https://doi.org/10.1093/idpl/ix022).
- [3] M. Gillies, R. Fiebrink, A. Tanaka, *et al.*, “Human-Centred Machine Learning,” en, in *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, San Jose California USA: ACM, May 2016, pp. 3558–3565, ISBN: 978-1-4503-4082-3. DOI: [10.1145/2851581.2856492](https://doi.org/10.1145/2851581.2856492).
- [4] M. Nyre-Yu, E. Morris, B. Moss, C. Smutz, and M. Smith, “Considerations for Deploying xAI Tools in the Wild: Lessons Learned from xAI Deployment in a Cybersecurity Operations Setting.,” en, in *Proposed for presentation at the ACM SIG Knowledge Discovery and Data Mining Workshop on Responsible AI held August 14-18, 2021 in Singapore, Singapore.*, US DOE, May 2021. DOI: [10.2172/1869535](https://doi.org/10.2172/1869535).
- [5] H.-F. Cheng, R. Wang, Z. Zhang, F. O’Connell, T. Gray, F. M. Harper, and H. Zhu, “Explaining Decision-Making Algorithms through UI: Strategies to Help Non-Expert Stakeholders,” en, in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, Glasgow Scotland Uk: ACM, May 2019, pp. 1–12, ISBN: 978-1-4503-5970-2. DOI: [10.1145/3290605.3300789](https://doi.org/10.1145/3290605.3300789).
- [6] M. Riedl, “Human-centered artificial intelligence and machine learning,” en, DOI: [10.1002/hbe2.117](https://doi.org/10.1002/hbe2.117).
- [7] G. Ramos, J. Suh, S. Ghorashi, C. Meek, R. Banks, S. Amershi, R. Fiebrink, A. Smith-Renner, and G. Bansal, “Emerging Perspectives in Human-Centered Machine Learning,” en, in *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, Glasgow Scotland Uk: ACM, May 2019, pp. 1–8, ISBN: 978-1-4503-5971-9. DOI: [10.1145/3290607.3299014](https://doi.org/10.1145/3290607.3299014).
- [8] A. Preece, D. Harborne, D. Braines, R. Tomsett, and S. Chakraborty, “Stakeholders in Explainable AI,” en, in *AAAI FSS-18: Artificial Intelligence in Government and Public Sector*, Arlington, Virginia, 2018, p. 6.

- [9] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, *et al.*, “Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI,” en, *Information Fusion*, vol. 58, pp. 82–115, Jun. 2020, ISSN: 15662535. DOI: [10.1016/j.inffus.2019.12.012](https://doi.org/10.1016/j.inffus.2019.12.012).
- [10] S. R. Hong, J. Hullman, and E. Bertini, “Human Factors in Model Interpretability: Industry Practices, Challenges, and Needs,” en, *Proceedings of the ACM on Human-Computer Interaction*, vol. 4, no. CSCW1, pp. 1–26, May 2020, ISSN: 2573-0142. DOI: [10.1145/3392878](https://doi.org/10.1145/3392878).
- [11] U. Ehsan, S. Passi, Q. V. Liao, L. Chan, I.-H. Lee, M. Muller, and M. O. Riedl, “The Who in XAI: How AI Background Shapes Perceptions of AI Explanations,” en, in *Proceedings of the CHI Conference on Human Factors in Computing Systems*, arXiv:2107.13509 [cs], May 2024, pp. 1–32. DOI: [10.1145/3613904.3642474](https://doi.org/10.1145/3613904.3642474).
- [12] A. Zytek, D. Liu, R. Vaithianathan, and K. Veeramachaneni, “Sibyl: Understanding and Addressing the Usability Challenges of Machine Learning In High-Stakes Decision Making,” in *IEEE Transactions on Visualization and Computer Graphics*, Conference Name: IEEE Transactions on Visualization and Computer Graphics, 2021, pp. 1–1. DOI: [10.1109/TVCG.2021.3114864](https://doi.org/10.1109/TVCG.2021.3114864).
- [13] F. Doshi-Velez and B. Kim, *Towards A Rigorous Science of Interpretable Machine Learning*, en, arXiv:1702.08608 [cs, stat], Mar. 2017.
- [14] R. Poyiadzi, X. Renard, T. Laugel, R. Santos-Rodriguez, and M. Detryniecki, *Understanding surrogate explanations: The interplay between complexity, fidelity and coverage*, en, arXiv:2107.04309 [cs], Jul. 2021.
- [15] M. T. Ribeiro, S. Singh, and C. Guestrin, ““Why Should I Trust You?”: Explaining the Predictions of Any Classifier,” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Feb. 2016, arXiv: 1602.04938.
- [16] S. M. Lundberg and S.-I. Lee, “A Unified Approach to Interpreting Model Predictions,” English, in *Advances in Neural Information Processing Systems*, vol. 31, Long Beach, California: Curran Associates Inc., 2017, p. 10.
- [17] A. Fisher, C. Rudin, and F. Dominici, “All Models are Wrong, but Many are Useful: Learning a Variable’s Importance by Studying an Entire Class of Prediction Models Simultaneously,” *Journal of Machine Learning Research*, vol. 20, pp. 1–81, Dec. 2019, arXiv: 1801.01489.
- [18] S. Wachter, B. Mittelstadt, and C. Russell, “Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR,” en, *SSRN Electronic Journal*, 2017, ISSN: 1556-5068. DOI: [10.2139/ssrn.3063289](https://doi.org/10.2139/ssrn.3063289).
- [19] M. T. Ribeiro, S. Singh, and C. Guestrin, “Anchors: High Precision Model-Agnostic Explanations,” en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, p. 9, Apr. 2018. DOI: [10.1609/aaai.v32i1.11491](https://doi.org/10.1609/aaai.v32i1.11491).



- [20] U. Bhatt, A. Xiang, S. Sharma, A. Weller, A. Taly, Y. Jia, J. Ghosh, R. Puri, J. M. F. Moura, and P. Eckersley, “Explainable machine learning in deployment,” in *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, ser. FAT\* ’20, New York, NY, USA: Association for Computing Machinery, Jan. 2020, pp. 648–657, ISBN: 978-1-4503-6936-7. DOI: [10.1145/3351095.3375624](https://doi.org/10.1145/3351095.3375624).
- [21] H. Jiang and E. Senge, “On Two XAI Cultures: A Case Study of Non-technical Explanations in Deployed AI System,” in *Human Centered AI (HCAI) workshop at NeurIPS 2021*, arXiv:2112.01016 [cs], Dec. 2021.
- [22] S. Tonekaboni, S. Joshi, M. Mccradden, and A. Goldenberg, “What Clinicians Want: Contextualizing Explainable Machine Learning for Clinical End Use,” *Proceedings of Machine Learning Research*, May 2019.
- [23] S. M. Lundberg, B. Nair, M. S. Vavilala, *et al.*, “Explainable machine-learning predictions for the prevention of hypoxaemia during surgery,” *Nature Biomedical Engineering*, vol. 2, no. 10, pp. 749–760, Oct. 2018, ISSN: 2157-846X. DOI: [10.1038/s41551-018-0304-0](https://doi.org/10.1038/s41551-018-0304-0).
- [24] B. C. Kwon, M.-J. Choi, J. T. Kim, E. Choi, Y. B. Kim, S. Kwon, J. Sun, and J. Choo, “RetainVis: Visual Analytics with Interpretable and Interactive Recurrent Neural Networks on Electronic Medical Records,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 299–309, Jan. 2019, arXiv: 1805.10724, ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: [10.1109/TVCG.2018.2865027](https://doi.org/10.1109/TVCG.2018.2865027).
- [25] D. Wang, Q. Yang, A. Abdul, and B. Y. Lim, “Designing Theory-Driven User-Centric Explainable AI,” *en*, in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI ’19*, Glasgow, Scotland Uk: ACM Press, 2019, pp. 1–15, ISBN: 978-1-4503-5970-2. DOI: [10.1145/3290605.3300831](https://doi.org/10.1145/3290605.3300831).
- [26] J. Wexler, M. Pushkarna, T. Bolukbasi, M. Wattenberg, F. Viegas, and J. Wilson, “The What-If Tool: Interactive Probing of Machine Learning Models,” *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2019, arXiv: 1907.04135, ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: [10.1109/TVCG.2019.2934619](https://doi.org/10.1109/TVCG.2019.2934619).
- [27] H. Nori, S. Jenkins, P. Koch, and R. Caruana, “InterpretML: A unified framework for machine learning interpretability,” *arXiv preprint arXiv:1909.09223*, 2019.
- [28] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert, “Manifold: A Model-Agnostic Framework for Interpretation and Diagnosis of Machine Learning Models,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 364–373, Jan. 2019, arXiv: 1808.00196, ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: [10.1109/TVCG.2018.2864499](https://doi.org/10.1109/TVCG.2018.2864499).
- [29] P. Hall, N. Gill, M. Kurka, W. Phan, and A. Bartz, “Machine Learning Interpretability with H2O Driverless AI,” *en*, Documentation, Apr. 2019.
- [30] V. Arya, R. K. E. Bellamy, P.-Y. Chen, *et al.*, “AI Explainability 360: An Extensible Toolkit for Understanding Data and Machine Learning Models,” *en*, *Journal of Machine Learning Research*, vol. 21, p. 6, 2020.

- [31] R. K. E. Bellamy, K. Dey, M. Hind, *et al.*, “AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias,” en, *IBM Journal of Research and Development*, vol. 63, no. 4/5, 4:1–4:15, Jul. 2019, ISSN: 0018-8646, 0018-8646. DOI: [10.1147/JRD.2019.2942287](https://doi.org/10.1147/JRD.2019.2942287).
- [32] H. Strobelt, S. Gehrmann, H. Pfister, and A. M. Rush, “LSTMVis: A Tool for Visual Analysis of Hidden State Dynamics in Recurrent Neural Networks,” in *InfoVis*, arXiv: 1606.07461, vol. 24, Phoenix, Arizona, USA: IEEE, Oct. 2017, pp. 667–676.
- [33] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau, “ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models,” English, in *IEEE Conference on Visual Analytics Science and Technology (VAST)*, arXiv: 1704.01942, vol. 24, Phoenix, Arizona, USA: IEEE, Apr. 2017, p. 10.
- [34] R. Vaithianathan, E. Putnam-Hornstein, N. Jiang, P. Nand, and T. Maloney, “Developing Predictive Models to Support Child Maltreatment Hotline Screening Decisions: Allegheny County Methodology and Implementation,” en, p. 60, 2017.
- [35] A. Zytek, W.-E. Wang, S. Koukoura, and K. Veeramachaneni, “Lessons from Usable ML Deployments Applied to Wind Turbine Monitoring,” in *Neurips 2023: XAI in Action: Past, Present, and Future Applications*, Oct. 2023.
- [36] A. Zytek, “Towards Usable Machine Learning,” en, Ph.D. dissertation, MIT, Feb. 2021.
- [37] L. Xu, M. Skoularidou, A. Cuesta-Infante, and K. Veeramachaneni, “Modeling tabular data using conditional GAN,” in *Advances in neural information processing systems*, 2019.
- [38] M. Robnik-Šikonja and M. Bohanec, “Perturbation-Based Explanations of Prediction Models,” en, in *Human and Machine Learning: Visible, Explainable, Trustworthy and Transparent*, ser. Human-Computer Interaction Series, J. Zhou and F. Chen, Eds., Cham: Springer International Publishing, 2018, pp. 159–175, ISBN: 978-3-319-90403-0. DOI: [10.1007/978-3-319-90403-0\\_9](https://doi.org/10.1007/978-3-319-90403-0_9).
- [39] C. Rudin, “Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead,” en, *Nature Machine Intelligence*, vol. 1, no. 5, pp. 206–215, May 2019, ISSN: 2522-5839. DOI: [10.1038/s42256-019-0048-x](https://doi.org/10.1038/s42256-019-0048-x).
- [40] I. Arnaldo and K. Veeramachaneni, “The Holy Grail of "Systems for Machine Learning": Teaming humans and machine learning for detecting cyber threats,” en, *ACM SIGKDD Explorations Newsletter*, vol. 21, no. 2, pp. 39–47, Nov. 2019, ISSN: 1931-0145, 1931-0153. DOI: [10.1145/3373464.3373472](https://doi.org/10.1145/3373464.3373472).
- [41] F. Cheng, D. Liu, F. Du, Y. Lin, A. Zytek, H. Li, H. Qu, and K. Veeramachaneni, “VBridge: Connecting the Dots Between Features and Data to Explain Healthcare Models,” *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2021, arXiv: 2108.02550, ISSN: 1077-2626, 1941-0506, 2160-9306. DOI: [10.1109/TVCG.2021.3114836](https://doi.org/10.1109/TVCG.2021.3114836).

- [42] D. Liu, S. Alnegheimish, A. Zyttek, and K. Veeramachaneni, “MTV: Visual Analytics for Detecting, Investigating, and Annotating Anomalies in Multivariate Time Series,” in *Proceedings of the ACM on Human-Computer Interaction*, arXiv: 2112.05734, vol. 6, Dec. 2021, pp. 1–30.
- [43] K. Veeramachaneni, U.-M. O’Reilly, and C. Taylor, “Towards Feature Engineering at Scale for Data from Massive Open Online Courses,” *arXiv:1407.5238 [cs]*, Jul. 2014, arXiv: 1407.5238.
- [44] D. Plohmann, K. Yakdan, M. Klatt, E. Gerhards-Padilla, and J. Bader, “A Comprehensive Measurement Study of Domain Generating Malware,” English, in *25th USENIX Security Symposium*, vol. 25, Austin, TX, 2016, p. 17.
- [45] S. Khalid, T. Khalil, and S. Nasreen, “A survey of feature selection and feature extraction techniques in machine learning,” in *2014 Science and Information Conference*, Aug. 2014, pp. 372–378. DOI: [10.1109/SAI.2014.6918213](https://doi.org/10.1109/SAI.2014.6918213).
- [46] A. Zheng and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*, en. "O’Reilly Media, Inc.", Mar. 2018, Google-Books-ID: sthSDwAAQBAJ, ISBN: 978-1-4919-5319-8.
- [47] J. Cheng and M. S. Bernstein, “Flock: Hybrid Crowd-Machine Learning Classifiers,” in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, ser. CSCW ’15, New York, NY, USA: Association for Computing Machinery, Feb. 2015, pp. 600–611, ISBN: 978-1-4503-2922-4. DOI: [10.1145/2675133.2675214](https://doi.org/10.1145/2675133.2675214).
- [48] W. Shi, G. Huang, S. Song, Z. Wang, T. Lin, and C. Wu, “Self-Supervised Discovering of Interpretable Features for Reinforcement Learning,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: [10.1109/TPAMI.2020.3037898](https://doi.org/10.1109/TPAMI.2020.3037898).
- [49] M. Choi, L. M. Aiello, K. Z. Varga, and D. Quercia, “Ten Social Dimensions of Conversations and Relationships,” in *Proceedings of The Web Conference 2020*, ser. WWW ’20, New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1514–1525, ISBN: 978-1-4503-7023-3. DOI: [10.1145/3366423.3380224](https://doi.org/10.1145/3366423.3380224).
- [50] Z. A. Daniels and D. Metaxas, “ScenarioNet: An Interpretable Data-Driven Model for Scene Understanding,” en, *IJCAI Workshop on Explainable Artificial Intelligence (XAI) 2018*, Jul. 2018.
- [51] B. Mathew, S. Sikdar, F. Lemmerich, and M. Strohmaier, “The POLAR Framework: Polar Opposites Enable Interpretability of Pre-Trained Word Embeddings,” in *Proceedings of The Web Conference 2020*, ser. WWW ’20, New York, NY, USA: Association for Computing Machinery, Apr. 2020, pp. 1548–1558, ISBN: 978-1-4503-7023-3. DOI: [10.1145/3366423.3380227](https://doi.org/10.1145/3366423.3380227).

- [52] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. Turaga, “Learning Feature Engineering for Classification,” en, in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization, Aug. 2017, pp. 2529–2535, ISBN: 978-0-9992411-0-3. DOI: [10.24963/ijcai.2017/352](https://doi.org/10.24963/ijcai.2017/352).
- [53] J. Duan, Z. Zeng, A. Oprea, and S. Vasudevan, “Automated Generation and Selection of Interpretable Features for Enterprise Security,” in *2018 IEEE International Conference on Big Data (Big Data)*, Dec. 2018, pp. 1258–1265. DOI: [10.1109/BigData.2018.8621986](https://doi.org/10.1109/BigData.2018.8621986).
- [54] C. Guan, X. Wang, Q. Zhang, R. Chen, D. He, and X. Xie, “Towards a Deep and Unified Understanding of Deep Neural Models in NLP,” en, in *International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, May 2019, pp. 2454–2463.
- [55] Q. Zhang, X. Wang, R. Cao, Y. N. Wu, F. Shi, and S.-C. Zhu, “Extracting an Explanatory Graph to Interpret a CNN,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020, Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, ISSN: 1939-3539. DOI: [10.1109/TPAMI.2020.2992207](https://doi.org/10.1109/TPAMI.2020.2992207).
- [56] W. Jitkrittum, Z. Szabo, K. Chwialkowski, and A. Gretton, “Distinguishing distributions with interpretable features,” in *ICML 2016 Workshop on Data-Efficient Machine Learning*, Conference Name: International Conference on Machine Learning (ICML): Data-Efficient Machine Learning workshop Meeting Name: International Conference on Machine Learning (ICML): Data-Efficient Machine Learning workshop Place: New York, USA Publisher: International Conference on Machine Learning (ICML): Data-Efficient Machine Learning workshop, vol. 2016, Jun. 2016.
- [57] K. Sokol and P. Flach, “Glass-Box: Explaining AI Decisions With Counterfactual Statements Through Conversation With a Voice-enabled Virtual Assistant,” en, in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, Stockholm, Sweden: International Joint Conferences on Artificial Intelligence Organization, Jul. 2018, pp. 5868–5870, ISBN: 978-0-9992411-2-7. DOI: [10.24963/ijcai.2018/865](https://doi.org/10.24963/ijcai.2018/865).
- [58] L. Beltzung, A. Lindley, O. Dinica, N. Hermann, and R. Lindner, “Real-Time Detection of Fake-Shops through Machine Learning,” in *2020 IEEE International Conference on Big Data (Big Data)*, Dec. 2020, pp. 2254–2263. DOI: [10.1109/BigData50022.2020.9378204](https://doi.org/10.1109/BigData50022.2020.9378204).
- [59] X. Chen, Q. Lin, C. Luo, *et al.*, “Neural Feature Search: A Neural Architecture for Automated Feature Engineering,” in *2019 IEEE International Conference on Data Mining (ICDM)*, ISSN: 2374-8486, Nov. 2019, pp. 71–80. DOI: [10.1109/ICDM.2019.00017](https://doi.org/10.1109/ICDM.2019.00017).
- [60] M. Doron, I. Segev, and D. Shahaf, “Discovering Unexpected Local Nonlinear Interactions in Scientific Black-box Models,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD ’19, New

- York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 425–435, ISBN: 978-1-4503-6201-6. DOI: [10.1145/3292500.3330886](https://doi.org/10.1145/3292500.3330886).
- [61] F. Harder, M. Bauer, and M. Park, “Interpretable and Differentially Private Predictions,” en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 4083–4090, Apr. 2020, Number: 04, ISSN: 2374-3468. DOI: [10.1609/aaai.v34i04.5827](https://doi.org/10.1609/aaai.v34i04.5827).
- [62] Y. Sheng, S. Tata, J. B. Wendt, J. Xie, Q. Zhao, and M. Najork, “Anatomy of a Privacy-Safe Large-Scale Information Extraction System Over Email,” in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD ’18, New York, NY, USA: Association for Computing Machinery, Jul. 2018, pp. 734–743, ISBN: 978-1-4503-5552-0. DOI: [10.1145/3219819.3219901](https://doi.org/10.1145/3219819.3219901).
- [63] C. Molnar, G. Casalicchio, and B. Bischl, “Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges,” *arXiv:2010.09337 [cs, stat]*, Oct. 2020, arXiv: 2010.09337.
- [64] A. Yadav, A. Rahmattalabi, E. Kamar, P. Vayanos, M. Tambe, and V. L. Noronha, “Explanation Systems for Influence Maximization Algorithms,” English, in *3rd International Workshop on Social Influence Analysis (SocInf 2017)*, 2017, p. 12.
- [65] T. Miller, “Explanation in Artificial Intelligence: Insights from the Social Sciences,” *arXiv:1706.07269 [cs]*, Aug. 2018, arXiv: 1706.07269.
- [66] U. Khurana, H. Samulowitz, and D. Turaga, “Feature Engineering for Predictive Modeling Using Reinforcement Learning,” en, *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018, Number: 1, ISSN: 2374-3468.
- [67] Z. Zhang, J. Singh, U. Gadiraju, and A. Anand, “Dissonance Between Human and Machine Understanding,” en, *Proceedings of the ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–23, Nov. 2019, ISSN: 2573-0142. DOI: [10.1145/3359158](https://doi.org/10.1145/3359158).
- [68] J. DeBoer, A. D. Ho, G. S. Stump, and L. Breslow, “Changing “Course”: Reconceptualizing Educational Variables for Massive Open Online Courses,” *Educational Researcher*, vol. 43, no. 2, pp. 74–84, Mar. 2014, Publisher: American Educational Research Association, ISSN: 0013-189X. DOI: [10.3102/0013189X14523038](https://doi.org/10.3102/0013189X14523038).
- [69] D. Dua and C. Graff, *UCI machine learning repository*, Publisher: University of California, Irvine, School of Information and Computer Sciences, 2017.
- [70] R. Kelley Pace and Ronald Barry, “Sparse Spatial Autoregressions,” *Statistics and Probability Letters*, vol. 33, no. 3, pp. 291–297, May 1997.
- [71] L. Breiman, “Random forests,” *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.
- [72] D. De Cock, “Ames, Iowa: Alternative to the Boston Housing Data as an End of Semester Regression Project,” en, *Journal of Statistics Education*, vol. 19, no. 3, Nov. 2011, ISSN: 1069-1898. DOI: [10.1080/10691898.2011.11889627](https://doi.org/10.1080/10691898.2011.11889627).
- [73] P. Cortez and A. Silva, “Using data mining to predict secondary school student performance,” *EUROSIS*, Jan. 2008.

- [74] *Iranian churn*, UCI Machine Learning Repository, UCI Machine Learning Repository, 2020. DOI: <https://doi.org/10.24432/C5JW3Z>.
- [75] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, “Scikit-learn: Machine Learning in Python,” en, *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
- [76] A. Zytek, W.-E. Wang, D. Liu, L. Berti-Equille, and K. Veeramachaneni, *Pyreal: A Framework for Interpretable ML Explanations*, arXiv:2312.13084 [cs], Dec. 2023.
- [77] A. Zytek, I. Arnaldo, D. Liu, and K. Veeramachaneni, “The Need for Interpretable Features: Motivation and Taxonomy,” en, *SIGKDD Explorations*, vol. 24, no. 1, Jun. 2022.
- [78] R. Poyiadzi, K. Sokol, R. Santos-Rodriguez, T. De Bie, and P. Flach, “FACE: Feasible and Actionable Counterfactual Explanations,” in *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, arXiv:1909.09369 [cs, stat], Feb. 2020, pp. 344–350. DOI: [10.1145/3375627.3375850](https://doi.org/10.1145/3375627.3375850).
- [79] S. Hadash, M. C. Willemsen, C. Snijders, and W. A. IJsselsteijn, “Improving understandability of feature contributions in model-agnostic explainable AI tools,” en, in *CHI Conference on Human Factors in Computing Systems*, New Orleans LA USA: ACM, Apr. 2022, pp. 1–9, ISBN: 978-1-4503-9157-3. DOI: [10.1145/3491102.3517650](https://doi.org/10.1145/3491102.3517650).
- [80] T. S. Filho, H. Song, M. Perello-Nieto, R. Santos-Rodriguez, M. Kull, and P. Flach, “Classifier Calibration: A survey on how to assess and improve predicted class probabilities,” en, *Machine Learning*, vol. 112, no. 9, pp. 3211–3260, Sep. 2023, arXiv:2112.10327 [cs, stat], ISSN: 0885-6125, 1573-0565. DOI: [10.1007/s10994-023-06336-7](https://doi.org/10.1007/s10994-023-06336-7).
- [81] F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker, “Gamut: A Design Probe to Understand How Data Scientists Understand Machine Learning Models,” en, in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*, Glasgow, Scotland Uk: ACM Press, 2019, pp. 1–13, ISBN: 978-1-4503-5970-2. DOI: [10.1145/3290605.3300809](https://doi.org/10.1145/3290605.3300809).
- [82] *Streamlit: A faster way to build and share data apps*, <https://streamlit.io/>, Jan. 2021.
- [83] J. M. Kanter, O. Gillespie, and K. Veeramachaneni, “Label, Segment, Featurize: A Cross Domain Framework for Prediction Engineering,” en, in *2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, Montreal, QC, Canada: IEEE, Oct. 2016, pp. 430–439, ISBN: 978-1-5090-5206-6. DOI: [10.1109/DSAA.2016.54](https://doi.org/10.1109/DSAA.2016.54).
- [84] W.-E. Wang, “Why did the prediction change? Explaining changes in predictions as time progresses,” MIT Dept. of EECS, 2024.
- [85] F. R. Hartwell, “Zephyr: A Data-Centric Framework for Predictive Maintenance of Wind Turbines,” MIT Dept. of EECS, 2023.
- [86] T. Chen and C. Guestrin, “XGBoost: A scalable tree boosting system,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, ser. KDD '16, Number of pages: 10 Place: San Francisco, California, USA tex.acmid: 2939785, New York, NY, USA: ACM, 2016, pp. 785–794, ISBN: 978-1-4503-4232-2. DOI: [10.1145/2939672.2939785](https://doi.org/10.1145/2939672.2939785).

- [87] J. Zhou, A. H. Gandomi, F. Chen, and A. Holzinger, “Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics,” en, *Electronics*, vol. 10, no. 5, p. 593, Jan. 2021, Number: 5 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2079-9292. DOI: [10.3390/electronics10050593](https://doi.org/10.3390/electronics10050593).
- [88] D. S. Watson and J. O’Hara, “Explaining Predictive Uncertainty with Information Theoretic Shapley Values,” in *37th Conference on Neural Information Processing Systems*, vol. 37, 2023.
- [89] H. Lakkaraju, D. Slack, Y. Chen, C. Tan, and S. Singh, *Rethinking Explainability as a Dialogue: A Practitioner’s Perspective*, en, arXiv:2202.01875 [cs], Feb. 2022.
- [90] A. Zytek, S. Pidò, and K. Veeramachaneni, *LLMs for XAI: Future Directions for Explaining Explanations*, en, arXiv:2405.06064 [cs], May 2024.
- [91] A. Bhattacharjee, R. Moraffah, J. Garland, and H. Liu, “Towards LLM-guided Causal Explainability for Black-box Text Classifiers,” in *AAAI ReLM 2024*, arXiv:2309.13340 [cs], AAAI, Jan. 2024.
- [92] N. Kroeger, D. Ley, S. Krishna, C. Agarwal, and H. Lakkaraju, “Are Large Language Models Post Hoc Explainers?” In *NeurIPS XAIA 2023*, arXiv:2310.05797 [cs], arXiv, Oct. 2023.
- [93] O. Khattab, A. Singhvi, P. Maheshwari, et al., *DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines*, en, arXiv:2310.03714 [cs], Oct. 2023.
- [94] V. B. Nguyen, J. Schlötterer, and C. Seifert, “From Black Boxes to Conversations: Incorporating XAI in a Conversational Agent,” en, Book Title: Explainable Artificial Intelligence Series Title: Communications in Computer and Information Science, vol. 1903, Cham: Springer Nature Switzerland, 2023, pp. 71–96, ISBN: 978-3-031-44069-4 978-3-031-44070-0. DOI: [10.1007/978-3-031-44070-0\\_4](https://doi.org/10.1007/978-3-031-44070-0_4).
- [95] H. Shen, C.-Y. Huang, T. Wu, and T.-H. K. Huang, “ConvXAI : Delivering Heterogeneous AI Explanations via Conversations to Support Human-AI Scientific Writing,” en, *Computer Supported Cooperative Work and Social Computing*, pp. 384–387, Oct. 2023, Conference Name: CSCW ’23: Computer Supported Cooperative Work and Social Computing ISBN: 9798400701290 Place: Minneapolis MN USA Publisher: ACM. DOI: [10.1145/3584931.3607492](https://doi.org/10.1145/3584931.3607492).
- [96] D. Slack, S. Krishna, H. Lakkaraju, and S. Singh, “Explaining machine learning models with interactive natural language conversations using TalkToModel,” en, *Nature Machine Intelligence*, vol. 5, no. 8, pp. 873–883, Aug. 2023, Number: 8 Publisher: Nature Publishing Group, ISSN: 2522-5839. DOI: [10.1038/s42256-023-00692-8](https://doi.org/10.1038/s42256-023-00692-8).
- [97] J. Burton, N. Al Moubayed, and A. Enshaei, “Natural Language Explanations for Machine Learning Classification Decisions,” en, in *2023 International Joint Conference on Neural Networks (IJCNN)*, Gold Coast, Australia: IEEE, Jun. 2023, pp. 1–9, ISBN: 978-1-66548-867-9. DOI: [10.1109/IJCNN54540.2023.10191637](https://doi.org/10.1109/IJCNN54540.2023.10191637).
- [98] Z. Guo, M. Yan, J. Qi, J. Zhou, Z. He, Z. Lin, G. Zheng, and X. Wang, *Adapting Prompt for Few-shot Table-to-Text Generation*, arXiv:2302.12468 [cs], Aug. 2023.

- [99] Z. Guo, R. Jin, C. Liu, *et al.*, *Evaluating Large Language Models: A Comprehensive Survey*, en, arXiv:2310.19736 [cs], Nov. 2023.
- [100] Y. Bai, J. Ying, Y. Cao, *et al.*, “Benchmarking Foundation Models with Language-Model-as-an-Examiner,” en, in *NeurIPS 2023 Datasets and Benchmarks*, arXiv:2306.04181 [cs], arXiv, Nov. 2023.
- [101] Y. Chen, R. Wang, H. Jiang, S. Shi, and R. Xu, *Exploring the Use of Large Language Models for Reference-Free Text Quality Evaluation: An Empirical Study*, arXiv:2304.00723 [cs], Sep. 2023. DOI: [10.48550/arXiv.2304.00723](https://doi.org/10.48550/arXiv.2304.00723).
- [102] Y. Ji, Y. Gong, Y. Peng, C. Ni, P. Sun, D. Pan, B. Ma, and X. Li, *Exploring ChatGPT’s Ability to Rank Content: A Preliminary Study on Consistency with Human Preferences*, en, arXiv:2303.07610 [cs], Mar. 2023.
- [103] J. Fu, S.-K. Ng, Z. Jiang, and P. Liu, *GPTScore: Evaluate as You Desire*, en, arXiv:2302.04166 [cs], Feb. 2023.
- [104] J. Wang, Y. Liang, F. Meng, Z. Sun, H. Shi, Z. Li, J. Xu, J. Qu, and J. Zhou, “Is ChatGPT a Good NLG Evaluator? A Preliminary Study,” en, in *NewSumm@EMNLP*, arXiv:2303.04048 [cs], arXiv, Oct. 2023.
- [105] Y. Liu, D. Iter, Y. Xu, S. Wang, R. Xu, and C. Zhu, *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment*, en, arXiv:2303.16634 [cs], May 2023.
- [106] UCI Machine Learning Repository, *Mushroom*, 1981.
- [107] M. Issakhani, P. Victor, A. Tekeoglu, and A. Lashkari, “PDF Malware Detection based on Stacking Learning,” en, in *Proceedings of the 8th International Conference on Information Systems Security and Privacy*, Online Streaming, — Select a Country —: SCITEPRESS - Science and Technology Publications, 2022, pp. 562–570, ISBN: 978-989-758-553-1. DOI: [10.5220/0010908400003120](https://doi.org/10.5220/0010908400003120).
- [108] S. Banerjee and A. Lavie, “METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments,” in *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, J. Goldstein, A. Lavie, C.-Y. Lin, and C. Voss, Eds., Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 65–72.
- [109] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*, en, arXiv:2201.11903 [cs], Jan. 2023.
- [110] D. Tuia, B. Kellenberger, S. Beery, *et al.*, “Perspectives in machine learning for wildlife conservation,” en, *Nature Communications*, vol. 13, no. 1, p. 792, Feb. 2022, Publisher: Nature Publishing Group, ISSN: 2041-1723. DOI: [10.1038/s41467-022-27980-y](https://doi.org/10.1038/s41467-022-27980-y).