An Analysis of the Ingredients for Learning Interpretable Symbolic Regression Models with Human-in-the-loop and Genetic Programming

GIORGIA NADIZAR, University of Trieste, Trieste, Italy and Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
LUIGI ROVITO, ANDREA DE LORENZO, and ERIC MEDVET, University of Trieste, Trieste, Italy
MARCO VIRGOLIN, Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Interpretability is a critical aspect to ensure a fair and responsible use of machine learning (ML) in high-stakes applications. Genetic programming (GP) has been used to obtain interpretable ML models because it operates at the level of functional building blocks: if these building blocks are interpretable, there is a chance that their composition (i.e., the entire ML model) is also interpretable. However, the degree to which a model is interpretable depends on the observer. Motivated by this, we study a recently-introduced human-in-the-loop system that allows the user to steer GP's generation process to their preferences, which shall be online-learned by an artificial neural network (ANN). We focus on the generation of ML models as analytical functions (i.e., symbolic regression) as this is a key problem in interpretable ML, and propose a two-fold contribution. First, we devise more general representations for the ML models for the ANN to learn upon, to enable the application of the system to a wider range of problems. Second, we delve into a deeper analysis of the system's components. To this end, we propose an incremental experimental evaluation, aimed at (1) studying the effectiveness by which an ANN can capture the perceived interpretability for simulated users, (2) investigating how the GP's outcome is affected across different simulated user feedback profiles, and (3) determining whether humans participants would prefer models that were generated with or without their involvement. Our results pose clarity on pros and cons of using a human-in-the-loop approach to discover interpretable ML models with GP.

CCS Concepts: • **Computing methodologies** → **Active learning settings**; **Genetic programming**; • **Computer systems organization** → **Neural networks**; • **Applied computing** → **Multi-criterion optimization and decision-making**; • **Human-centered computing** → **User studies;**

Additional Key Words and Phrases: Explainable artificial intelligence, interpretable machine learning, active learning, neural networks, genetic programming, deep learning, evolutionary computation, evolutionary algorithms, explainable evolutionary computation

G. Nadizar and L. Rovito share first-co-authorship.

Authors' addresses: G. Nadizar, Department of Mathematics, Informatics and Geosciences, University of Trieste, Via Alfonso Valerio 12 34127, Trieste, Italy and Department of Evolutionary Intelligence, Centrum Wiskunde & Informatica, Science Park 123 1098 XG, Amsterdam, The Netherlands; e-mail: giorgia.nadizar@phd.units.it; L. Rovito, Department of Mathematics, Informatics and Geosciences, University of Trieste, Via Alfonso Valerio 12 34127, Trieste, Italy; e-mail: luigi.rovito@phd.units.it; A. De Lorenzo and E. Medvet, Department of Engineering and Architecture, University of Trieste, Via Alfonso Valerio 6 34127, Trieste, Italy; e-mails: andrea.delorenzo@units.it, emedvet@units.it; M. Virgolin, Department of Evolutionary Intelligence, Centrum Wiskunde & Informatica, Science Park 123 1098 XG, Amsterdam, The Netherlands; e-mail: marco.virgolin@cwi.nl.

ACM 2688-3007/2024/02-ART5
https://doi.org/10.1145/3643688

ACM Reference Format:
Giorgia Nadizar, Luigi Rovito, Andrea De Lorenzo, Eric Medvet, and Marco Virgolin. 2024. An Analysis of the Ingredients for Learning Interpretable Symbolic Regression Models with Human-in-the-loop and Genetic Programming. *ACM Trans. Evol. Learn.* 4, 1, Article 5 (February 2024), 30 pages. https://doi.org/10.1145/3643688

1 INTRODUCTION

Recent years have seen that an irresponsible use of machine learning models can pose considerable risks [Jobin et al. 2019]. To deal with this problem, the field of **Explainable AI** (**XAI**) studies methods to explain the predictions of models and how they can be possibly interpreted, and methods to generate models that are inherently amenable for human-interpretation (i.e., *interpretable* models) [Adadi and Berrada 2018; Guidotti et al. 2018a]. The importance of research concerning XAI is mainly justified by the presence of highly complex models (i.e., *black-box* models) in high-stakes real-world applications, in which users need to in-depth interact with these models to analyze the obtained predictions and understand why those types of predictions have been performed. The current consensus is that methods to explain black-box models have certain limitations (depending on their respective underlying assumptions), and should therefore be used with care [Molnar 2020; Molnar et al. 2020]. Indeed, if the data allows for an interpretable model to be found that achieves high accuracy, then, arguably, the interpretable model should be preferred over a black-box counterpart [Rudin 2019].

Genetic Programming (**GP**) is an **Evolutionary Computation** (**EC**) algorithm that can be used to seek interpretable models as, for instance, formulae, decision trees, rule sets, or computer programs [Koza 1994]. GP operates by initializing a *population* of (typically) random models composed of basic operations (or instructions), and *evolving* the population by stochastic recombination or mutation of the models followed by survival of the fittest. If the basic operations at play are clearly understandable and, at the same time, GP discovers an accurate model, then there is a chance that the discovered model is interpretable as well, i.e., it contains a limited number of operations that are composed in an understandable manner. Yet, leaving the discovery of interpretable models to chance alone is unlikely to yield acceptable results. Therefore, to steer GP towards discovering interpretable models, we need to define an objective function that represents a suitable proxy of interpretability. Unfortunately, interpretability is intrinsically subjective, i.e., it strongly depends on the application at hand and on the background of the user [Benk and Ferrario 2020; Lipton 2018]. Moreover, different applications may require a different tradeoff between model accuracy and model interpretability, depending on the stakes at play [Freitas 2014; Hatherley 2020].

Virgolin et al. [2021] proposed to address the problem concerning the subjective nature of interpretability by means of a GP-based human-in-the-loop system. The system, called ***model learning with personalized interpretability estimation*** (**ML-PIE**), uses GP in a multi-objective fashion. One objective is the model's accuracy, while the other is a proxy of interpretability that is implemented with a neural network. The neural network is trained upon feedback from the user that is conveyed via a graphical interface while GP's evolution is taking place. More specifically, the user is repeatedly prompted with a pair of models (among the ones discovered by the ongoing GP process) and, for each pair, is asked to tell which of the two models is more interpretable according to their personal definition of interpretability. The authors found that, over time, the network learns to approximate the preference of the user, and that the user tends to prefer models found with ML-PIE over those found with non-personalized baselines. Figure 1 provides an overview of ML-PIE, its main components and phases—we provide later a detailed description of the system.

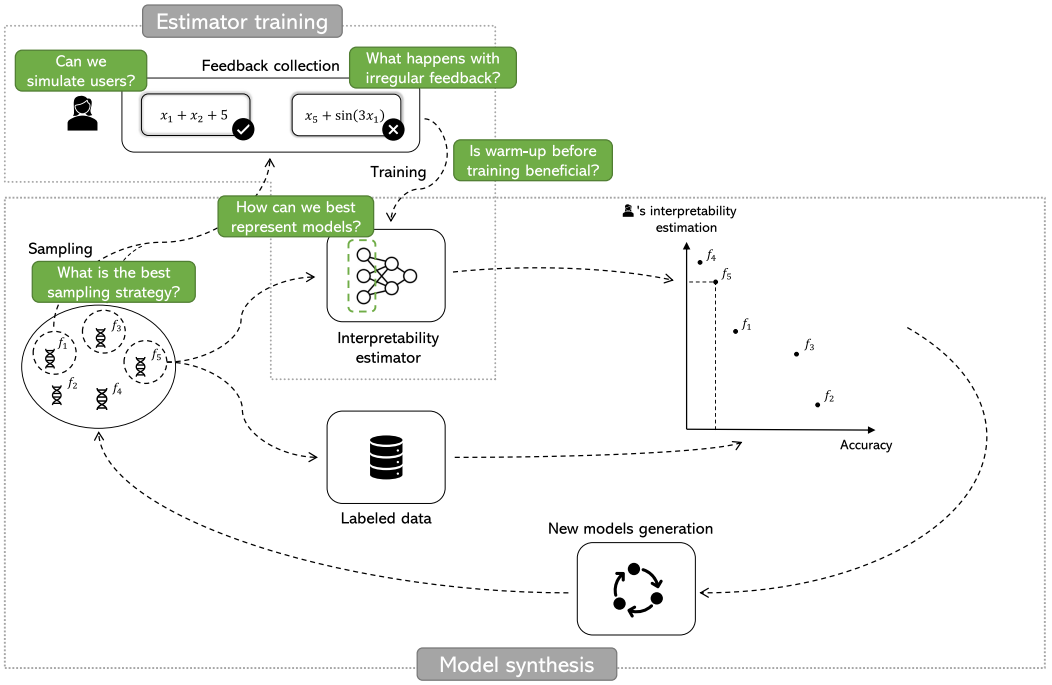

Fig. 1. Schematic view of ML-PIE. In the green rectangles, we highlight the research questions we tackled in this study.

While promising, the proposal of Virgolin et al. [2021] had a number of limitations. Firstly, the system (more specifically, the neural network) was specifically configured to work for symbolic regression models. For example, some of the features used by the system are the number of basic arithmetic operations ($+, -, \times, \div$). It is unclear whether ML-PIE would still work well with more general features, which can be used to represent models of other nature (e.g., decision trees and rules). Secondly, several aspects were not addressed in [Virgolin et al. 2021], such as whether the reliability of the *active learning* [Ren et al. 2021; Settles 2009] strategy proposed by the authors scales as the representation used to estimate the interpretability of the models becomes more complex (i.e., more fine-grained); or whether updating the network as GP's search progresses has a positive or negative impact on the outcome of the search, possibly depending on the frequency with which the user provides feedback. In other words, we argue that it remains largely unclear how to best insert the human in the loop when using GP for interpretable machine learning, and what are possible limitations thereof. In this article, we try to expand our understanding of GP with human-in-the-loop for interpretable machine learning.

Namely, we address the aforementioned shortcomings by (a) proposing different representations (features) to capture the interpretability of the models evolved by GP, and by (b) experimentally assessing the building blocks of ML-PIE, both with real and simulated users. In Figure 1, we further detail the specific components of the ML-PIE we investigate and highlight what aspects we target, in the form of questions.

We structure our experimental evaluation in three phases, as follows:

(1) First, we assess the expressiveness of the model representations and how well a neural network can learn upon them against simulated users feedback. We highlight a clear trade-off between expressiveness and learning capability, and show that uncertainty-based active

learning (as proposed in [Virgolin et al. 2021]) does *not* scale to larger representations unless the amount of feedback significantly grows.

(2) Then, we incorporate the neural network within GP to act as proxy for the user's interpretability, and proceed to study the search dynamics in different scenarios, again mimicking user feedback according to some ground-truths, i.e., simulating users. We discover that the search is indeed influenced by different notions of interpretability, which yield diverse outcomes, yet it proves robust with respect to inconsistent users who provide feedback in an irregular manner.

(3) Last, we involve actual human participants to use our system for finding suitable models for two real-world datasets, and let them judge, in a blind manner, whether the models generated with their participation are preferable over some predefined ones. We find that with limited user involvement, i.e., with a scarce amount of feedback collected, the users tend to prefer predefined models over the ones tailored with our system, hence we note a possible need for prolonging the feedback collection process.

Based on the outcome of the experimental phase, we summarize the main contributions of our work as follows:

(i) We design and implement a system that is able to address any kind of machine learnable problem, in contrast to the original work that is applicable only for mathematical formulae. To the best of our knowledge, this is the first attempt to build a general problem-agnostic framework for discovering ML models that are interpretable according to a personalized notion of interpretability. This is of paramount importance since in many real-world applications there is the need to solve many heterogeneous problems that often require interpretable models, where the interpretability is arguably subjective with respect to the user that has to interact with the discovered model;

(ii) We propose, implement, and compare experimentally different and novel ways of encoding a tree for being evaluated by a neural network, only accounting for the syntactical structure (i.e., function set and terminal set) of it, making this approach general and problem-agnostic. Each proposed encoding type represents a tradeoff between expressiveness and complexity, and the choice of the encoding type depends upon the type of interpretability definition to learn and the availability of feedback. To the best of our knowledge, this is the first attempt to present a method that can map the syntactical representation of a GP tree to a numerical vector that can be easily processed by a vanilla feed-forward neural network to estimate the interpretability of the tree, regardless of the semantic that is related to the operators in the function set. We demonstrate that by incorporating this type of general interpretability estimator in the proposed framework we are able to learn and approximate an arbitrary general and non-linear interpretability definition, provided that enough training data (i.e., user feedback) is processed. We believe that this is of high relevance for the field of XAI since it enables us to learn arbitrary interpretability notions by employing general problem-agnostic numerical representations of the evolved models.

This article proceeds as follows. In Section 2, we review the current literature on evolutionary algorithms applied to discover interpretable models. In Section 3, we formulate the problem at hand; in Section 4, we give an overview of the framework based on [Virgolin et al. 2021]. In Section 5, we describe the application of *ML-PIE* in the context of **Symbolic Regression** (**SR**), and we present the tree encoding strategies, how the estimator is optimized, and the feedback collection methods. In Section 6, we discuss the experiments conducted both with simulated human intervention and with actual human participants. Finally, in Section 7, we draw the conclusions

and summarize our contributions in the context of supervised learning with personalized interpretability.

2 RELATED WORK

Novel approaches for discovering how to explain and interpret ML model predictions have gained great interest in the scientific literature [Doran et al. 2017; Goebel et al. 2018; Hagras 2018; Hoffman et al. 2018; Holzinger 2018]. The most relevant strategies that can be adopted to make models explainable consist in directly synthesizing interpretable models by means of *ad hoc* learning algorithms or, alternatively, applying specifically designed techniques (e.g., reverse engineering, feature attribution methods, counterfactual explanations) on black-box models to obtain the needed explanations [Adadi and Berrada 2018; Guidotti et al. 2018a; Ribeiro et al. 2016; Vilone and Longo 2020; Xu et al. 2019]. The latter approach can be model-agnostic, but has the clear limitation that it cannot provide a complete understanding of how a model operates for any potential input [Lipton 2018; Rudin 2019]. Here, we focus on the former approach, i.e., attempting to synthesize inherently interpretable models.

In general, decision trees, rule-based systems, and linear models are considered to be better candidates to obtain interpretable models than more complex models such as deep neural networks [Das and Rad 2020; Došilović et al. 2018; Guidotti et al. 2018b; Huysmans et al. 2011; Sagi and Rokach 2020; Samek and Müller 2019]. However, even potentially simple models, such as the aforementioned ones, may require justification, i.e., the simplicity of the model does not imply its interpretability anyway [Kovalerchuk et al. 2021; Lipton 2018]. Decision tree models may be simplified by leveraging size reduction and pruning strategies [Breslow and Aha 1997; Izza et al. 2020], or by minimizing a loss function that represents a tradeoff between detection accuracy and model complexity [Lakkaraju et al. 2016; Su et al. 2015]. Linear models may be simplified by reducing the number of features [Poursabzi-Sangdeh et al. 2021; Tibshirani 1996; Ustun and Rudin 2016; Zou and Hastie 2005]. In general, several types of ML models may, hopefully, be improved by removing irrelevant and redundant features (e.g., linear models, regression trees, and decision trees). Furthermore, neural network-based models may be improved as well by removing redundant hidden layers and neurons.

In the context of interpretable machine learning, evolutionary algorithms, such as **Genetic Algorithms** (**GA**) and GP, have gained great interest because of their capabilities to learn potentially interpretable models (EC for XAI) [Bacardit et al. 2022; Fernandez et al. 2019; Mei et al. 2022; Sharma et al. 2020]. Especially, GP has been widely adopted to seek intrinsically interpretable models such as decision trees [Ferigo et al. 2023] and classification rules based on **Learning Classifier Systems** (**LCSs**) algorithms [Urbanowicz and Moore 2009]. A common strategy that can be implemented to obtain interpretable trees consists in limiting the number of tree components. This can be done by minimizing the number of nodes, along with the other objectives, and, additionally, by limiting the function set to simple functions only [Brotto Rebuli et al. 2023; Cano et al. 2013; Ekart and Nemeth 2001; Lensen 2021; Lensen et al. 2020; Murphy et al. 2021; Rovito et al. 2022; Smits and Kotanchek 2005; Virgolin et al. 2020a].

Model components can also be weighted according to a given pre-defined weighting scheme such that the weighted sum of these components may provide an estimation of the model interpretability, with the least interpretable models that undergo a penalization during the evolution [Hein et al. 2018; Medvet et al. 2015]. Moreover, formula simplification can be employed, perhaps as a post-processing step, to simplify the learned models for enhancing their interpretability and readability [Javed et al. 2022].

Recently, much research work has been done in the field of interpretable EC applied to **Reinforcement Learning** (**RL**) [Kaelbling et al. 1996; Videau et al. 2022; Wilson et al. 2018]

scenarios. Custode and Iacca [2023] combined **Grammatical Evolution** (**GE**) [O'Neill and Ryan 2001] with Q-learning [Watkins and Dayan 1992] to evolve interpretable decision trees for RL problems by learning a decomposition of the state-space. This approach was also extended by the authors to continuous action spaces [Custode and Iacca 2021]. Since interpretable models usually struggle when dealing with raw data and high dimensionality, Custode and Iacca [2022a] presented a framework based on end-to-end pipelines composed of multiple interpretable models co-optimized by means of evolutionary algorithms. With this system, the authors were able to decompose the decision-making process in an RL environment and compute high-level features from raw data that are easily understandable. Machine learning was also employed to generate policies for containing pandemics [Kompella et al. 2020; Miikkulainen et al. 2021; Trott et al. 2021]. In this setting, Custode and Iacca [2022b] proposed a combination of RL and EC techniques to generate interpretable policies for containing pandemics, with a major focus on the COVID-19 one.

Another strategy that was proposed consists in a data-driven approach that showed how to derive from human feedback an interpretability formula that can be leveraged to estimate the interpretability of formulae synthesized within a GP evolutionary process [Virgolin et al. 2020b]. This was later adopted by [Custode and Iacca 2023]. The major difference between [Virgolin et al. 2020b] and our proposed approach consists in the fact that, while in [Virgolin et al. 2020b] a formula is directly learned by training a linear model on data that was collected through a mathematical domain-constrained survey, in this work we propose a framework that has the potential to learn an arbitrary interpretability notion of an arbitrary user, whose domain of expertise is not necessarily tied to a mathematical one.

The framework presented by Virgolin et al. [2021] tried to address the fact that interpretability is subjective to the observer. The authors implemented an SR system in which formulae are evaluated according to an interpretability estimator trained with user feedback. This system consists in a bi-objective GP process where the first objective evaluates the detection accuracy of the models while the second objective evaluates the interpretability through a neural network trained concurrently during the evolutionary process. Pairs of formulae are sampled using an active learning criterion based on the estimator uncertainty and presented to the users [Settles 2009; Yang and Loog 2016]. The users select the best formula for each pair according to their subjective interpretability concept. Such feedback is used to train the interpretability estimator that, in this way, learns to approximate a personalized **Proxy of Human Interpretability** (**PHI**).

There exist several other works that attempted to build an AI-based system with human-in-the-loop during training [Christiano et al. 2017; Mahoor et al. 2017; Secretan et al. 2011]. Murphy et al. [2021] state that incorporating human feedback during a model synthesis process may help to discover interpretable models. There also exist works in which active learning is leveraged within GP [Bartoli et al. 2017; De Freitas et al. 2010; Isele and Bizer 2013] to directly incorporate the collected labels into the objective function.

In this work, we analyze more in depth the system presented by Virgolin et al. [2021], especially since that work was designed for SR problems, while GP can in principle be used to evolve many other types of model. We provide a generalization of the neural network-based interpretability estimator that now can be adapted to any type of evolutionary problems with an arbitrary function set. Therefore, the *ML-PIE* framework of Virgolin et al. [2021] is not constrained to exclusively work with SR problems anymore. We show that by employing an estimator that relies on general automatically-crafted features it is possible to learn an approximation of arbitrary PHI with few training samples. Even though this approach is applicable to arbitrary types of problems, we conducted the experiments on SR because of the wide variety of literature that highlights the importance of this problem in GP.

3 PERSONALIZED MODEL LEARNING: PROBLEM STATEMENT

We deal with a supervised learning scenario where, given a dataset $D = \left\{(\boldsymbol{x}^{(i)}, y^{(i)})\right\}_i$ of observations and corresponding labels, the goal is to find a model $f : X \to Y$ that is *accurate*, i.e., $f(\boldsymbol{x}^{(i)})$ correctly predicts the label $y^{(i)}$ for each i, and is also *interpretable*.

We assume that the accuracy of a model can be measured with a function $q_f : \mathcal{F}_{X\to Y} \to \mathbb{R}$, where $\mathcal{F} = \{f, f : X \to Y\}$ is the *space of models* (or hypothesis space). In practice, a number of well-defined accuracy measures exists, such as the mean squared error or the coefficient of determination for regression (i.e., when $Y = \mathbb{R}$), and the percentage of correct classifications for classification (i.e., when Y is a finite set without intrinsic ordering). Regarding interpretability, we assume that a measure $\hat{\psi}_f : \mathcal{F}_{X\to Y} \to \mathbb{R}$ exists that captures the *personal* notion of interpretability of the user.

We consider the case in which the user wants to solve a supervised learning problem by providing the dataset D and the desired measure of accuracy q_f, but not their interpretability measure $\hat{\psi}_f$ (e.g., because the user is *unable* to formalize $\hat{\psi}_f$). We assume, however, that the user is available for providing feedback (annotations) about the interpretability of models that are generated by the machine learning process.

4 PERSONALIZED MODEL LEARNING WITH ML-PIE

4.1 Overview

We propose to solve the interpretable model learning problem stated in the previous section as a bi-objective search in the space $\mathcal{F}_{X\to Y}$ of models, where accuracy and interpretability are the two objectives. Since the *true* interpretability measure $\hat{\psi}_f$ is not provided, the corresponding objective is pursued using an *estimator*[1] ψ_f of interpretability that is learned during the search of the model using the user's feedback. This approach is an extension of the one pursued by Virgolin et al. [2021] and, just like in the original work, we are going to refer to it as ML-PIE.

Internally, ML-PIE resorts to a *model search algorithm* for searching $\mathcal{F}_{X\to Y}$ and to an *active learning algorithm* for training the estimator. Both algorithms are iterative in nature. The training process of the estimator is realized via *human-in-the-loop*: the user is prompted with selected models among the ones being evaluated by the search algorithm, and asked to provide feedback about the models' interpretability. ML-PIE collects the user's feedback concurrently with the model search process, yet asynchronously: that is, the rate at which models are generated is independent from the rate at which the user provides feedback on (part of) them.

4.2 Concurrent Model Search and Active Learning

In ML-PIE, the model search and the active learning algorithms proceed concurrently (yet asynchronously). The motivation for a concurrent approach, as opposed to, e.g., an off-line one where user feedback is collected *prior* to model search, is that the user will be presented with the models that the model search algorithm is discovering. This means that ψ_f is trained upon data that is *in-distribution* for the model search algorithm, since that data represents the actual distribution of models that are being discovered. Conversely, in an off-line approach there would be no guarantee that the user feedback could cover the same models (in terms of distribution) being discovered by the model search algorithm.

At initialization, ML-PIE starts with a randomly generated initial estimator ψ_f. Note that ψ_f can also be initially set to be an estimator that was *pre-trained*, e.g., on previous data that was

[1] Although both the model f that evolution is searching for and the estimator of interpretability ψ_f can be called model or estimator, from here on we refer with *model* to the first, and with *estimator* to the second.

annotated by one or multiple users [Virgolin et al. 2020b], or on a proxy of interpretability from the literature (e.g., [Vladislavleva et al. 2009]): we discuss a few alternatives for initializing ψ_f in Section 5.3.1. Then, at each iteration, the model search algorithm builds some new candidate models and evaluates them using q_f and the current ψ_f. ML-PIE adds (syntactically unique) models discovered by the model search algorithm to an initially empty set $F \subset \mathcal{F}_{X \to Y}$ of models. F is re-set at every iteration of the model search algorithm, to contain only models that are relevant to the current status of the search. Meanwhile, the active learning algorithm is in charge of iteratively refining ψ_f. At each iteration, the active learning algorithm builds a *query*, i.e., it selects a pair f_1, f_2 of candidate models from F according to a priority rule (see Section 5.4), and submits the two models for user assessment. The user is shown the two models on a graphical interface and is instructed to choose the one that they believe to be more interpretable. When the user answers the query, the active learning algorithm updates ψ_f by using the feedback as signal. For example, if the user judges f_1 to be more interpretable than f_2, then the active learning algorithm will update ψ_f to increase the difference $\psi_f(f_1) - \psi_f(f_2)$ if $\psi_f(f_1) > \psi_f(f_2)$ (i.e., the model is given a *positive reward* signal), or it will update it to decrease the difference $\psi_f(f_2) - \psi_f(f_1)$ otherwise (i.e., the model is given a *negative reward* signal). Once ψ_f has been updated, the active learning algorithm builds a new query, thus starting a new iteration of the feedback collection phase.

We remark that feedback collection is modeled as a simple choice between two models to make the annotation task easier for the user, and the approach general to many types of model spaces $\mathcal{F}_{X \to Y}$. Indeed, asking the user to rank more than two models at a time would reasonably require considerably more effort. Similarly, it may be hard for the user to provide an estimation for a single model at a time, e.g., in a form of a score from one to ten. About this last point, designing a scoring function that the user is requested to consider may also be a hard task for the designer of the system.

The feedback collection process (equivalently, the active learning algorithm) continues to run until the model search algorithm terminates. The model search algorithm terminates when a certain budget is exhausted, such as a maximum number of iterations (in GP, generations), or a maximum time is reached.

In summary, ML-PIE executes concurrently two iterative algorithms. The model search algorithm updates one or more models at each iteration, guided by a fixed q_f and a moving ψ_f; the pace of iterations is the fastest possible on the machine ML-PIE is executing on. The active learning algorithm updates ψ_f at each iteration, guided by the user; the pace of iterations is determined by rate at which the user provides feedback.

4.3 Applicability

ML-PIE, as described so far, is a rather general approach that can be applied to many different use cases. For example, regarding the space of models $\mathcal{F}_{X \to Y}$, the only requirements are that (a) models in $\mathcal{F}_{X \to Y}$ are compatible with the estimator being updated by the active learning algorithm ψ_f, and that (b) pairs of models can be judged by the user in terms of relative interpretability. In practice, the first requirement can be met if a proper encoding of the models in $\mathcal{F}_{X \to Y}$ is used—we discuss a concrete case in Section 5. The second requirement can be met if the models can be visualized: hence, the user's feedback can be collected by means of a **graphical user interface** (**GUI**).

For what concerns the model search algorithm, i.e., the one searching in $\mathcal{F}_{X \to Y}$, we require it to be bi-objective, although this constraint can, if necessary, be relaxed using linearization or lexicographic order among objectives.

Lastly, a practical requirement is that the speed at which the model search algorithm progresses must be approximately compatible with the speed at which the user provides feedback. In other

words, the model search algorithm should run for enough time for the user to provide a reasonable amount of feedback (we experiment in detail on this in Section 6.4.2).

5 ML-PIE FOR SYMBOLIC REGRESSION

We now consider a concrete application of ML-PIE to the case study of SR, showing how the general framework described so far can be adapted to this specific problem of interpretable machine learning.

SR is a form of supervised learning in which, given data on d independent numerical variables $(x_1, \ldots, x_d) = \boldsymbol{x}$ and on the dependent numerical variable y, one wish to find the model $f : \mathbb{R}^d \to \mathbb{R}$ that best explains y from $\boldsymbol{x}$ while, *crucially*, f can be written as an *analytical expression* (or, simply, a formula). In other words, f must be realized by composition of basic and interpretable functional building blocks such as $+, \times, -, \div, \exp, \ln, x_1, x_2, 0.5, -1, \pi$, and so on. These building blocks are decided by the user of the system and are an input for the SR problem, which is NP-hard [Virgolin and Pissis 2022]. With respect to the general formulation of Section 3, here $X = \mathbb{R}^d$, $Y = \mathbb{R}$, and the space of models $\mathcal{F}_{X \to Y}$ is the space $\mathcal{S}_d$ of formulae for d variables. Consequently, the interpretability estimator $\psi_f : \mathcal{S}_d \to \mathbb{R}$ takes a formula as input.

We consider GP to act as model search algorithm since it has been shown to be an effective approach for SR [La Cava et al. 2021]. We set GP to work in a bi-objective manner, with accuracy and interpretability as objectives. At termination, GP outputs a collection of formulae, with different tradeoff levels among each other between the considered objectives. This allows the user to choose the model that best fits their needs. We provide a more detailed description of the model search via GP in Section 5.1.

Concerning the active learning algorithm for the interpretability estimator ψ_f, we investigate several design options. All of them share the following general structure. We use an encoding $h : \mathcal{S}_d \to \mathbb{R}^m$ to map a model, i.e., a formula, to a numeric vector whose m components are features of the formula. These features must be decided beforehand, we discuss this in Section 5.2. Next, we use an **artificial neural network** (**ANN**) to obtain an interpretability estimate $\psi_f(f)$ for a formula f by giving the feature vector of f as the input for the ANN, i.e., $\psi_f(f) := \text{ANN}_{\boldsymbol{\theta}}(h(f))$, where $\boldsymbol{\theta} \in \mathbb{R}^p$ represents the parameters of the ANN. Over time, by using an online learning approach, the parameters $\boldsymbol{\theta}$ must be optimized to make the ANN consistent with the user's feedback; provided that the choice of the encoding function h is adequate, i.e., the features of the formulae are informative. We discuss (and experiment with) a few design options for the encoding in Section 5.2 and for the ANN optimization in Section 5.3. Besides updating ψ_f by optimizing $\boldsymbol{\theta}$, the active learning algorithm is also in charge of building the queries, i.e., of selecting which pairs of formulae should be shown to the user. We discuss design options concerning query building in Section 5.4. From a practical point of view, we collect the user's feedback via a GUI, depicted in Figure 2: ML-PIE shows two rendered formulae and the user can tell which one is the most interpretable with a single click on the GUI.

We remark that, although we only focus on and experiment with the case of SR, our approach remains fully general, provided the few requisites mentioned in Section 4.3 are satisfied. For instance, we could tackle RL problems where symbolic policies are sought with the very same methodology proposed for SR, by simply introducing an appropriate fitness evaluation in simulated environments. Moreover, we could also generalize our scheme to the search of other types of models, i.e., not only symbolic formulae, as further detailed in Section 5.2.

5.1 Bi-objective Model Search

We use the NSGA-II algorithm [Deb et al. 2002] as bi-objective model search algorithm, with the implementation provided in [Blank and Deb 2020]. More in detail, we resort to a GP version of NSGA-II, encoding the formulae as trees [O'Neill 2009]. We use NSGA-II because it is, for a

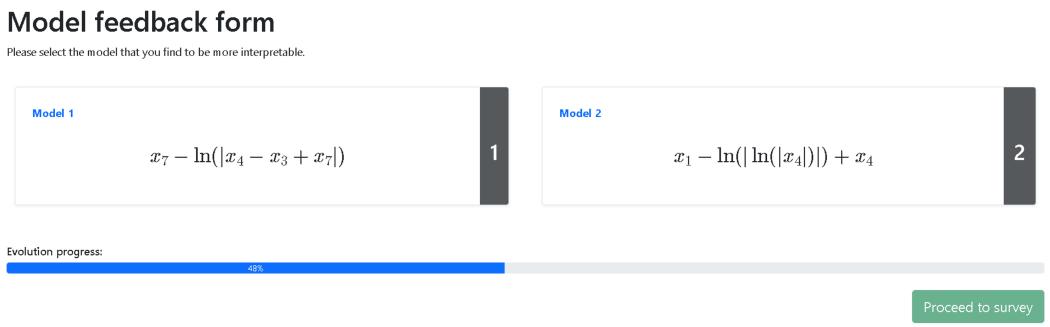

Fig. 2. A screenshot of the user interface during the feedback collection process.

multi-objective evolutionary setting, a very popular choice. However, we remark that the specific optimization algorithm is not the key point of this work, rather the important part is how we compute the interpretability estimation. As a matter of fact, this system may be implemented as well by replacing NSGA-II with another multi-objective optimization algorithm. We believe that NSGA-II is the most convenient choice considering the scope of this work because of its popularity and proven speed and effectiveness for multi-objective discrete optimization tasks.

For the initialization of the population, we use the popular *ramped half-and-half* initialization method [Luke and Panait 2001], with a maximum depth of four.

Every iteration (or *generation*), offspring are created from the current population. Firstly, promising parents are selected using tournament selection of size 2. Next, offspring are generated from the parents in number equal to the population size, using subtree crossover and subtree mutation. In detail, firstly we create 90 % of the offspring via crossover of randomly-chosen parents (two parents produce two offspring), while the remaining 10 % of the offspring are clones of randomly-chosen parents. Next, 60 % of the offspring, chosen at random, undergo subtree mutation. During this process, we discard (and repeat the creation of) any offspring with depth greater than $l_{\max} = 4$ (as too large formulae are hardly interpretable), and that are identical to an existing member of the population or of the offspring generated so far. We discard identical trees because diversity preservation can greatly improve the performance of NSGA-II when used for GP [Liu et al. 2022]. Specifically, we retain two trees to be identical if they are semantically equivalent (e.g., $x_1 + x_2$ is the same as $x_2 + x_1$). We do this by simply comparing the corresponding predictions, meaning that if the predictions are the same, then we assume that the trees are identical, even if they are syntactically different.

After the offspring have been created, NSGA-II applies a final selection, based on non-domination and crowding distance, upon the union between the population (which was given at the beginning of the generation) and the offspring. This final selection round also employs a form of elitism, performed according to the working principle of NSGA-II, and results in the population for the next generation.

We employ the following primitives as nodes for the trees representing the formulae: (1) the variables $x_1, \ldots, x_d$ of the problem at hand, also known as the *features* of the dataset, (2) random constants sampled uniformly at random between -5 and 5, and (3) the mathematical operators $+$, $-$, $\times$, $\div^*$, $(\cdot)^3$, $\ln^*$, max (where * denotes the protected version[2] to avoid mathematical operations performed on out-of-domain values). We set the population size to 200, and we iterate the algorithm for 50 generations. These settings are the result of several preliminary experiments, and correspond to a reasonable compromise between performance and execution time. In particular,

[2] $\div^*(a, b) = \text{sign}(b)\ \frac{a}{|b| \text{ if } |b| > 10^{-9} \text{ else } 10^{-9}}$; $\ln^*(a) = \ln(|a| + 10^{-9})$

we aim at enabling the algorithm to discover accurate models, while keeping the model search within reasonable time for the involved user. With the chosen settings, each model synthesis lasts approximately 10 min.

As mentioned before, NSGA-II attempts to optimize accuracy and interpretability simultaneously. Regarding the former, we use the coefficient of determination, i.e., the R^2 score, to be maximized—which corresponds to maximizing the model accuracy. We include *linear scaling* [Keijzer 2003, 2004] to adapt the fitting of the synthesized formulae, given its proven effectiveness on real-world datasets [Virgolin et al. 2019]. Regarding interpretability, we rely on the outcome of the estimator ψ_f, which should ideally capture the perceived interpretability of a formula for the user in question. Since the estimations of interpretability change over time as ψ_f is updated by the user's feedback, the interpretability of the formulae is re-computed at every generation. Finally, the outcome NSGA-II is a set $S^\star \subset \mathcal{S}_d$ of formulae that achieve a Pareto tradeoff between the two objectives, i.e., each formula corresponds to some level of compromise between accuracy and interpretability.

5.2 Model Encoding

We consider three options to realize the encoding function $h : \mathcal{S}_d \to \mathbb{R}^m$, which differ in terms of size, complexity, and expressiveness. In our setting of SR, as formulae are represented with trees, h also operates on trees. We remark that the tree representation can be generalized to other problems besides SR, as, e.g., regular expressions or decision trees, which proves the generality of our approach.

5.2.1 Counts Encoding. With this option, a tree is encoded as a vector where each component corresponds to a possible primitive, and the value of that component is the number of times the primitive occurs in the tree. In particular, $|\mathcal{M}|$ components are devoted to counting the occurrences of each mathematical operator, with $\mathcal{M} = \{+, -, \times, \div^*, (\cdot)^3, \ln^*, \max\}$ for SR, d components are employed for counting the occurrences of each of the features of the problem, and one component counts how many numerical constants appear in the tree. The encoding vector further includes three components that provide additional information about shape and size of the tree, namely: (i) the ratio between the number of depth levels and the number of nodes; (ii) the ratio between the maximal number of operands that an operator in the given tree can have, and the maximal breadth (i.e., the maximum number of nodes at a same depth level); (iii) the percentage of leaf nodes.

The size of this encoding is thus, $m = |\mathcal{M}| + d + 1 + 3$. This encoding can be advantageous because it scales with $\mathcal{M}$ and d, i.e., it is independent of the maximal number of nodes in the tree. However, this encoding does not distinguish between primitives appearing at different positions in the tree, which might impact interpretability.

5.2.2 One-hot Encoding. Different from the counts encoding, the one-hot encoding takes into account the exact position of each element in the tree. More specifically, each node is one-hot encoded as a vector of size $|\mathcal{M}| + d + 1$, where all components are set to zero with the exception of the one corresponding to the element in the node (the last +1 being the vector element reserved to all the constants), which is set to one. Then, $h(f)$ is the concatenation of the encoding of all possible nodes as the tree is traversed from the root in a breadth-first fashion. To ensure encoding coherence, we assume all nodes to have a number of operands equal to the maximal possible (maximum arity α): in case a node is missing, i.e., it is an intron, the corresponding encoding is $\mathbf{0} \in \mathbb{R}^{|\mathcal{M}|+d+1}$.

Clearly, the one-hot encoding is more expressive than the counts encoding, as it can capture a wide variety of properties of the tree that can be related to interpretability. However, the one-hot encoding scales poorly compared to the counts encoding, since it depends on $|\mathcal{M}|$, d, as well as on

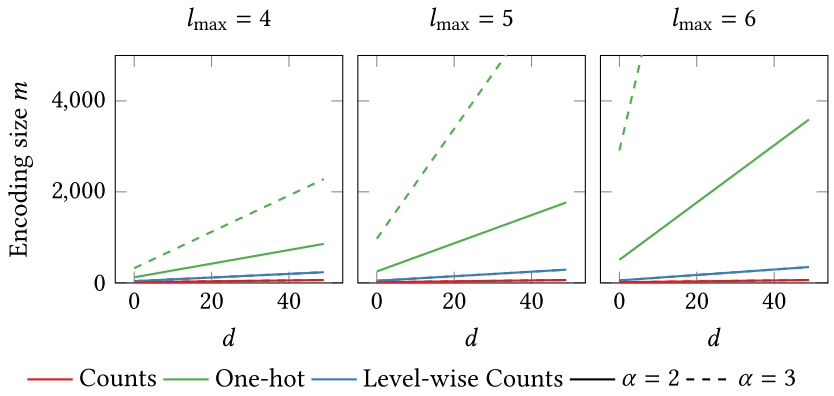

Fig. 3. Encoding sizes (in different colors) w.r.t. the amount of problem features d (on the x-axis), the maximum tree depth $l_{\max}$ (in each subplot), and the maximum node arity α (solid vs. dashed line). We keep the size of the function set $\mathcal{M}$ fixed to 7, as in Section 5.1.

the maximum possible number of nodes contained in a tree of maximal depth $l_{\max}$. Namely, the encoding size is $m = \frac{\alpha^{l_{\max}}-1}{\alpha-1}(|\mathcal{M}| + d + 1)$.

5.2.3 Level-wise Counts Encoding. This last encoding represents a compromise between the counts encoding and the one-hot encoding. This encoding counts the occurrences of operators, problem features, and constants separately for each depth level $l \leq l_{\max}$ in the tree. In addition, as for the counts encoding, we provide three descriptive components for the shape and size of tree.

The size of the level-wise counts encoding is $m = l_{\max}(|\mathcal{M}| + d + 1)+3$, which makes it still fairly compact with respect to the maximal number of levels $l_{\max}$, the size of the function set $|\mathcal{M}|$, and the amount of problem features d. However, differently from the one-hot encoding, information of specific node positions is now missing.

Figure 3 shows how the size m of the three encodings scales for different combinations of $|\mathcal{M}|$, d, α, and $l_{\max}$.

5.3 ANN Optimization

We use an ANN as interpretability estimator. The ANN takes in input the encoded formula and returns a score of interpretability, i.e., $\psi_f(f) = \text{ANN}_{\boldsymbol{\theta}}(h(f))$. We make use of dropout [Srivastava et al. 2014] at prediction time [Moore et al. 2016] to make ANNs provide a measure of prediction uncertainty, which enables to perform uncertainty-based active learning. Other ways of obtaining an uncertainty measure from an ANN could be used, e.g., bootstrapping or conformal prediction [Psaros et al. 2023]: we chose dropout because it well integrates with the rest of our framework.

Regarding the architecture of the ANN, we employ two hidden layers, each with 100 neurons with ReLU activations [Nair and Hinton 2010]. The output layer consists of a single node with tanh activation. Moreover, we add dropout to each hidden layer [Srivastava et al. 2014] with a probability of 0.25. Dropout is enabled both at training and evaluation time. During training, the neural network is applied (forward pass) to the encoding of the given formula *once* in order to obtain the signal needed to train the ANN's parameters (explained below). Instead, when the ANN is used for inference, we actually perform $k = 10$ predictions (forward passes) for the same formula, to account for the stochastic nature of dropout. We then take the mean of the k predictions as interpretability estimate, and the standard deviation as uncertainty (see Section 5.4).

We rely on a binary signal from a pair of formulae for optimizing the parameters $\boldsymbol{\theta}$ of the ANN, given the nature of the feedback requested from the user (preferred formula between two options).

In detail, a training data point for optimizing $\boldsymbol{\theta}$ is essentially a triplet that contains two formulae, f_1 and f_2, and a binary label $p \in \{-1, 1\}$. If the user deems f_1 to be more interpretable than f_2, then $p = -1$, otherwise $p = 1$. Given this data, we compute the interpretability scores $\psi_f(f_1)$ and $\psi_f(f_2)$ (one forward pass for each), and we compare them to p. Then, we use the Wasserstein loss function [Gulrajani et al. 2017] to train the ANN:

$$\mathcal{L}\left(\psi_f\right) = p\left(\psi_f\left(f_1\right) - \psi_f\left(f_2\right)\right), \tag{1}$$

which was used in the same scenario in [Virgolin et al. 2021]. The loss will be positive if the user and the estimator disagree on the relative interpretabilities of f_1 and f_2, whereas it will be negative in case of agreement. Note that this loss trains a *ranking* ANN rather than a regressor ANN. In other words, the ANN does not learn to estimate scores that have a specific meaning per se, rather, these scores are only meaningful in relative terms, i.e., to rank different models. As shown by Virgolin et al. [2021], this training effective and requires less annotation effort than estimating specific scores (i.e., a regression approach). Moreover, the employed model search algorithm, NSGA-II, relies for the most part on rankings rather than specific values. The only exception of this is the computation of the crowding distance, which is anyhow normalized, at each generation, using the minimum and maximum value for each objective encountered.

Finally, we use Adam as optimizer [Zhang 2018], with a learning rate of 10^{-3} and a weight decay of 10^{-5}.

We believe that using an ANN as interpretability estimator for an online learning strategy is a natural choice because, by using a given loss, we are able to incrementally update the interpretability estimator model (ANN weights). This happens because we can take advantage of gradients, which can be incrementally computed as feedback is received. Specifically, our feedback consists in telling, given two formulae, which one is the best. To capture this aspect, we use a Wasserstein-like loss. Moreover, we also want a way to measure prediction uncertainty. If we used another type of model such as a decision tree, then the problem of training the estimator by using user feedback would turn into a binary classification problem, in which we cannot neither leverage incremental gradients nor compute uncertainty anymore.

5.3.1 ANN Warm-up. As stated in Section 4, ML-PIE starts the search for a formula f using an initial interpretability estimator ψ_f^0. In practice, the initial estimator may have a substantial impact on what models are eventually discovered, as it drives the model search algorithm in a particular direction.

We, therefore, consider the effect of an optional *warm-up phase* prior to the actual start of ML-PIE. In other words, we consider the option of providing ML-PIE with a ψ_f^0 which is the result of an optimization performed on a dataset of triplets $(f_1^{(i)}, f_2^{(i)}, p^{(i)})_i$, pre-collected and independent from the problem at hand—but compatible with it in terms of $\mathcal{M}$ and d. We elaborate below on choices to obtain the warm-up signal p. The warm-up phase needs not be too short (in terms of how many training triplets are given as data), as that will be ineffective, nor too long, as that will has the potential to lead to premature convergence to a sub-optimal notion of interpretability (with respect to the one of the user). With preliminary experiments, we found that using 20 triplets leads to reasonable results.

We consider the following two options to realize the warm-up.

ϕ-driven warm up. We rely on the ϕ interpretability estimator proposed by Virgolin et al. [2020b] for formulae, which we use to label 20 pairs of randomly generated formulae that use $\mathcal{M}$ and d variables. For each pair f_1, f_2 of randomly generated formulae, p is set to -1 if $\phi(f_1) \leq \phi f_2$ or to 1 otherwise. The ϕ estimator takes into account the number of components, the number of

operations, the amount of non-arithmetic operations, and the number of consecutive compositions of non-arithmetic operations in a formula. We remark that the encodings proposed in Section 5.2 use features that are more general than those used by ϕ, which is tailored for SR.

Data-driven warm up. As an alternative and potentially more general approach, we experiment also with a data-driven warm-up. In this case, we wish to use a dataset of "well-formed" and "elegant" models for the problem at hand, retrieved, e.g., from the internet. For the case study of SR, we particularly consider a subset of the dataset of the 100 equations from the Feynman lectures on physics [Feynman et al. 1965; La Cava et al. 2021]. Specifically, we build 20 triplets f_1, f_2, p by extracting f_1 from the dataset, and building f_2 out of a random mutation to f_1 that makes the formula larger in terms of number of nodes for the respective tree. We make the assumption that, by its very construction, any f_2 is less interpretable than its respective f_1, and set the label p accordingly. Finally, to obtain a balanced dataset, we swap f_1 with f_2 and change the sign of the respective p for 50 % of the triplets.

We remark that this data-driven approach can be extended to other types of models than SR. For example, this approach could be used for the automatic inference of regular expressions, using a pool of human-written cases (notably, regular expressions can be represented as trees and evolved with GP [Bartoli et al. 2016]).

5.4 Query Building

The user is presented with a query of formulae $(f_1, f_2) \in F$ (F is the set of unique formulae available in the population of the current generation), where f_1 and f_2 have been picked by the active learning algorithm. We consider two approaches to realize active learning.

5.4.1 Random Sampling. We use *random sampling* without re-insertion, i.e., random f_1 and f_2 are extracted from F and the user will not see a same formula twice. While simple and fast to execute, this baseline approach is limited approach in that it makes no attempt at maximizing the information gain for training the ANN. However, random sampling is can still be effective for very large input spaces (here, when the encoding has very high dimensionality).

5.4.2 Uncertainty Sampling. The second strategy is to use (the ANN's) uncertainty sampling, which is a common strategy in active learning and it was found to be effective in [Virgolin et al. 2021]. To realize this, we assign to each formula a level of uncertainty using the ANN with dropout at prediction time as explained in Section 5.3. Namely, for any given formula in the set F of formulae seen during the current NSGA-II's generation, $k = 10$ predictions are obtained from the ANN. These predictions are different from one another due to the ANN's dropout. Then, the standard deviation of those predictions is taken as uncertainty. Finally, we pick the two formulae with the largest uncertainty among those in F.

Clearly, uncertainty sampling is more computationally heavy than random sampling, as it requires to evaluate the uncertainty for all the formulae in F for each query. In fact, this computation cannot be done beforehand, as the uncertainty might vary from query to query due to the update of ψ_f performed in between them, which results in different uncertainties for the same formula.

6 EXPERIMENTS AND DISCUSSION

We divide the experimental phase in three main chapters, respectively, aimed to (1) simulate how well the estimator ψ_f can be trained to mimic the user's preference (regardless of model search), (2) assess the impact of different user profiles (feedback speed) on the outcome of GP's model search, and (3) deploy the most promising configuration of ML-PIE from the previous experiments on actual users and assess its effectiveness.

In the first experiment, we consider different ways in which the user's true notion of interpretability can be simulated, i.e., we consider different PHIs. We describe the PHIs we consider in the next subsection. We do this across the different encodings h, warm-up strategies, and active learning sampling techniques. For the second experiment, we run ML-PIE with simulated users that act according to the PHIs, with different profiles in terms of response pattern. The third experiment is realized by asking 42 B.Sc., M.Sc., and Ph.D. students of courses of engineering, computer science, and AI, to participate and give feedback on two real-world datasets (explained below, in Section 6.2).

All experimental phases are driven by keeping in mind that the proposed system should involve actual human participants. We have to assume that human participants are not highly responsive and probably they are not willing to provide much feedback. Overall, it is safe to assume that a human participant will probably not spend several hours providing feedback to train the interpretability estimator. Despite that, we want to analyze our system and try to make it perform well even if the amount of feedback is low. To this end, experiments with the estimator are not conducted with many training pairs in order to provide a realistic point of view. Moreover, experiments with GP are conducted with a low number of generations and population size to ensure that the whole optimization process does not take too much time, otherwise there is a significantly high chance that the user gets bored.

Our code is implemented in Python 3.9.12 and is available at the following repository.[3] We leverage Scikit-learn [Pedregosa et al. 2011] and PyTorch [Paszke et al. 2019] for the pre-processing of the datasets and the neural network training, respectively, and GenePro [Virgolin 2022] and PyMOO [Blank and Deb 2020] for the GP evolution. We remark that, unless otherwise specified, we set all the parameters to the values used originally used by Virgolin et al. [2021] for ML-PIE.

6.1 Considered Proxies of Human Interpretability

We resort to simulated users for the first and second experiments, which are aimed at understanding and refining key options of ML-PIE, prior to the third experiment with actual users. These simulated users respond according to a given PHI. Doing so allows us to provide a large amount of feedback data in a deterministic and coherent manner. The PHIs we consider are the following:

- ℓ*-PHI* evaluates formulae solely based on their size, i.e., the number of nodes in the tree representation. In other words, the smaller the formula the more interpretable it is considered to be, regardless of the primitives composing it.
- ϕ*-PHI* adheres to the interpretability estimator ϕ proposed in [Virgolin et al. 2020b], based on the size of the formula, the number of operations, the number of non-arithmetic operations, and the number of consecutive compositions of non-arithmetic operations.
- *W-PHI* assigns a different weight to each possible primitive (each function in $\mathcal{M}$, each problem feature, and constants) and then computes the sum of weights across the primitives that appear in the tree to evaluate the interpretability of the respective formula.
- *LW-PHI* extends W-PHI, in a level-wise fashion, meaning that primitives at different depth level are assigned a different weight.
- *NW-PHI* further extends W-PHI and LW-PHI, in that the weighting is now dependent on primitives as well as their specific position it the tree. This is the most general PHI we consider.

For W-PHI, LW-PHI, and NW-PHI, we simulate different users by sampling the weights using

$$w = -1 \times |\rho|, \text{ with } \rho \sim \mathcal{N}\left(0, \sigma^2\right). \tag{2}$$

[3] https://github.com/lurovi/ML-PIE

The negative one multiplication represents the fact that each included component lowers a formula's interpretability, while σ^2 is chosen differently for each type primitive. We particularly choose smaller values of σ^2 for simple operators (such as $+, -, \times, \div$), variables and constants (which are clearly understandable), as these are arguably easy to understand, and larger values for more complex operators (such as ln and max), since these may be reasonably perceived as harder to understand.

Note that, by construction, our PHIs are always negative. Moreover, we remark that the proposed PHIs do not necessarily capture some *true* notion of interpretability, e.g., using the model size has been often criticized in the interpretability literature, but they just act as proxies to simulate a possible user behavior.

6.2 Datasets Description

We run our GP-based experiments on two ML datasets for supervised regression tasks:

- *Boston housing* (Boston): dataset for discovering models that can predict prices of houses in different areas of Boston [Harrison Jr and Rubinfeld 1978]. This dataset is used, e.g., for assessing fairness in AI, since it includes a feature regarding race. It contains 506 examples and 13 features;
- *Heating load* (Heating): dataset for discovering models that can predict heating load requirements of buildings. It contains 768 examples and 8 features [Tsanas 2012; Tsanas and Xifara 2012].

For each dataset, we perform three different 7:3 random splits in training and test set. Features are normalized by using robust scaling. For each split, we run 10 different GP processes.

6.3 Experiment 1: Training the Interpretability Estimator

We train the ANN using up to 150 training pairs, sampled according to each of the two active learning approaches, from a training set consisting of 500 randomly generated trees. We choose 150 feedback rounds as a sufficiently large number for the expected number of feedback that users are willing to give. Each training pair is labeled according to a given PHI that simulates a possible interpretability notion of a generic user. For each feedback, we compute the Spearman footrule [Diaconis and Graham 1977; Spearman 1906] (a measure of how much two rankings mismatch) on a validation set consisting of 300 randomly generated trees. In this phase, we use six distinct variables $x_1, \ldots, x_6$ in the randomly-generated formulae. We repeat each experiment 10 times with different seeds, and we perform this type of experiment using every combination of the following parameters: (1) encoding h (Section 5.2); (2) PHI (Section 6.1); (3) active learning method (Section 5.4); (4) warm-up strategy (Section 5.3.1).

Figure 4 shows the outcome of this experiment. Our findings are as follows:

- *Encoding*. Counts encoding seems to work generally best, across different PHIs and active learning methods, although the differences are not statistically significant. This encoding is the one with the lowest dimensionality and best scalability (recall Section 5.2). Counts encoding is sufficient to learn a simple PHI like ℓ-PHI, while it appears to be incapable to learn more complex PHIs, like NW-PHI (note the convergence of the footrule). Contrary to counts encoding, one-hot encoding is the largest encoding we consider. We can observe that, on the simple ℓ-PHI, one-hot encoding can exhibit overfitting (for the solid line, i.e., random sampling strategy). Meanwhile, one-hot encoding can improve the footrule beyond counts encoding's capability, on complex PHIs like NW-PHI. However, a large amount of feedback is needed, prior to this becoming possible. Considering the low amount of training that can

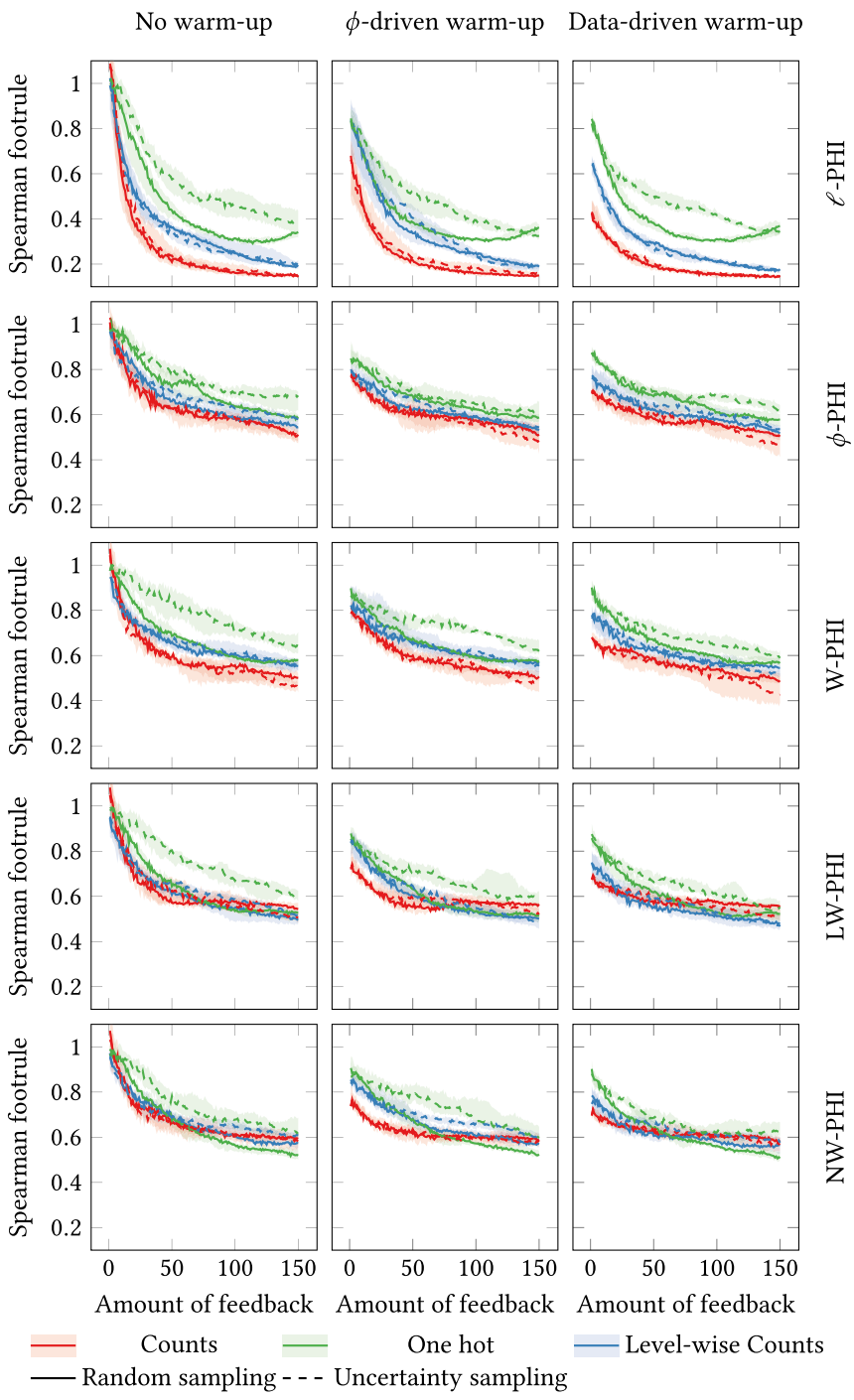

Fig. 4. Median and interquartile range of Spearman footrule with increasing amount of feedback for different formulae encodings (color), sampling strategies (linetype), warm-up procedures (one per column), and different PHIs (one per row).

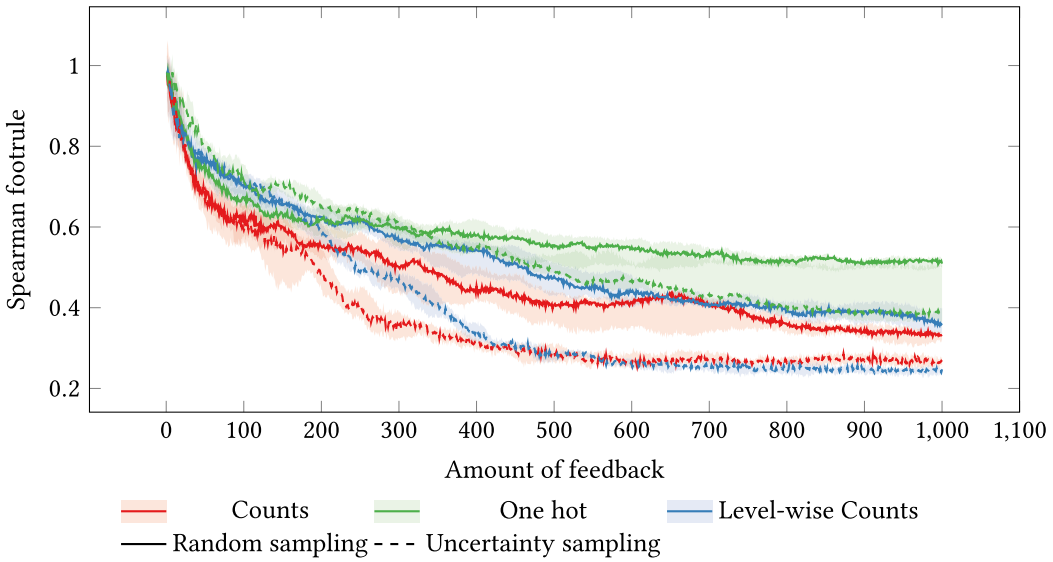

Fig. 5. Median and interquartile range of spearman footrule with increasing amount of feedback for different formulae encodings (color) and sampling strategies (linetype) with no warm-up and ϕ-PHI.

be expected in a human-in-the-loop setting, a small encoding is likely to be the best suited, compared to a larger one.

— *PHI.* As expected, ℓ-PHI is the ground-truth that is most easily captured by the estimator, irrespective of encoding choice, since it is very simple. We can see that the footrule decreases even with more complex ground-truths such as ϕ-PHI and NW-PHI. This means that, potentially, a more general encoding allows the estimator to approximate more complex PHIs, provided that the estimator is trained with enough feedback;

— *Active learning.* The results suggest that random sampling is generally equal to, or better than, uncertainty-based sampling. Only in some cases, e.g., W-PHI with Feynman warm-up and counts encoding, uncertainty-based sampling shows a moderate improvement over random sampling. We delve deeper into this later in this section.

— *Warm-up.* In almost all configurations, and most evidently for counts encoding, warm-up does help lowering the footrule when no feedback has been given yet. After around 20–25 feedback rounds, however, having no warm-up catches up with having it. At the same time, using warm-up does not generally cause premature convergence to a suboptimal estimator, compared to not using warm-up. In general, there is no observable substantial difference between the two proposed warm-up strategies.

In order to provide a more detailed explanation of some of our previous observations, we conduct additional experiments in which we measure the average uncertainty and the footrule on the validation set when the estimator is trained with much more feedback. Figure 5 shows the performance of the ANN when trained with up to 1000 training pairs. We adopt ϕ-PHI to label the feedback and we do not perform warm-up in this specific experiment. In this case, the uncertainty sampling becomes, on average, more effective than random sampling after 200–400 feedback, across encoding choice. This result validates our hypothesis that uncertainty-based active learning *can*, in fact, beat random sampling. However, especially depending on how complex the encoding is, too much feedback may be required before that is the case.

Table 1. p-values from Bonferroni-corrected Wilcoxon Paired Tests of the Distribution of Footrules (10 Runs) at Five Feedback Rounds (with Random Sampling)

PHI	ϕ-driven vs. no warm-up	Data-driven vs. no warm-up
ℓ	0.097	0.004
ϕ	0.004	0.008
W	0.008	0.008
LW	0.004	0.027
NW	0.004	0.020

Given the aforementioned observations, we decide to adopt, for the following experiments: counts encoding, random sampling, and the ϕ-based warm-up. Regarding the latter aspect, this choice is based on statistical testing: Table 1 shows whether using a warm-up strategy over no warm-up is significantly better when only a small number of five feedback is given. We thus choose ϕ warm-up because a user might be very unresponsive, and ϕ warm-up achieves smaller p-values compared to the data-driven warm-up across the different PHIs (except for ℓ that, however, is very simplistic).

Take-home message. We provide different general novel ways of encoding a tree. Each encoding type has pros and cons as regards expressiveness and complexity. However, according to our experiments, a general numerical representation of a GP tree is able to capture arbitrary interpretability notions. Hence, the choice of the specific encoding type highly depends on the availability of feedback and the interpretability proxy to approximate. Specifically, a simple encoding type is efficient and it is able to quickly learn an interpretability definition, but the estimation may be quite inaccurate if the interpretability definition is highly complex. On the other hand, a complex encoding type is time-consuming to compute, but it is potentially able to learn any kind of interpretability definition, regardless of its complexity. However, even for simple interpretability proxies, a complex encoding type requires the availability of a large amount of user feedback.

6.4 Experiment 2: Integrating the Interpretability Estimator within GP

In this section, we delve into the analysis of the full ML-PIE process, i.e., with multi-objective GP running while the user (in this section, still simulated) provides feedback to train the estimator of interpretability. As mentioned at the end of the previous section, and motivated by that section's results, we adopt: counts encoding, random sampling, ϕ warm-up.

In this experiment, we attempt to answer the following questions:

(1) Do different user profiles induce different outcomes in terms of model accuracy? In other words, is GP's search substantially influenced by by the user's personal notion of interpretability?
(2) Is the proposed approach robust with respect to different user behaviors in terms of engagement and response rate?

To this extent, we perform a two fold experimental evaluation, as described below.

6.4.1 User Impact on Discovered Tradeoffs between Accuracy and Interpretability. We instantiate ML-PIE using a simulated user who responds at regular intervals of 5 s, according to one of the PHIs. As PHIs, we consider a smaller but reasonable subset, comprising ℓ-PHI, ϕ-PHI, and NW-PHI. We consider ℓ-PHI and ϕ-PHI as they refer to common interpretability notions [Virgolin et al. 2020b], but we also include NW-PHI, which is one of the most general notions we introduced. We repeat the model search 10 times for each combination of dataset, train-test split, and PHI, for a

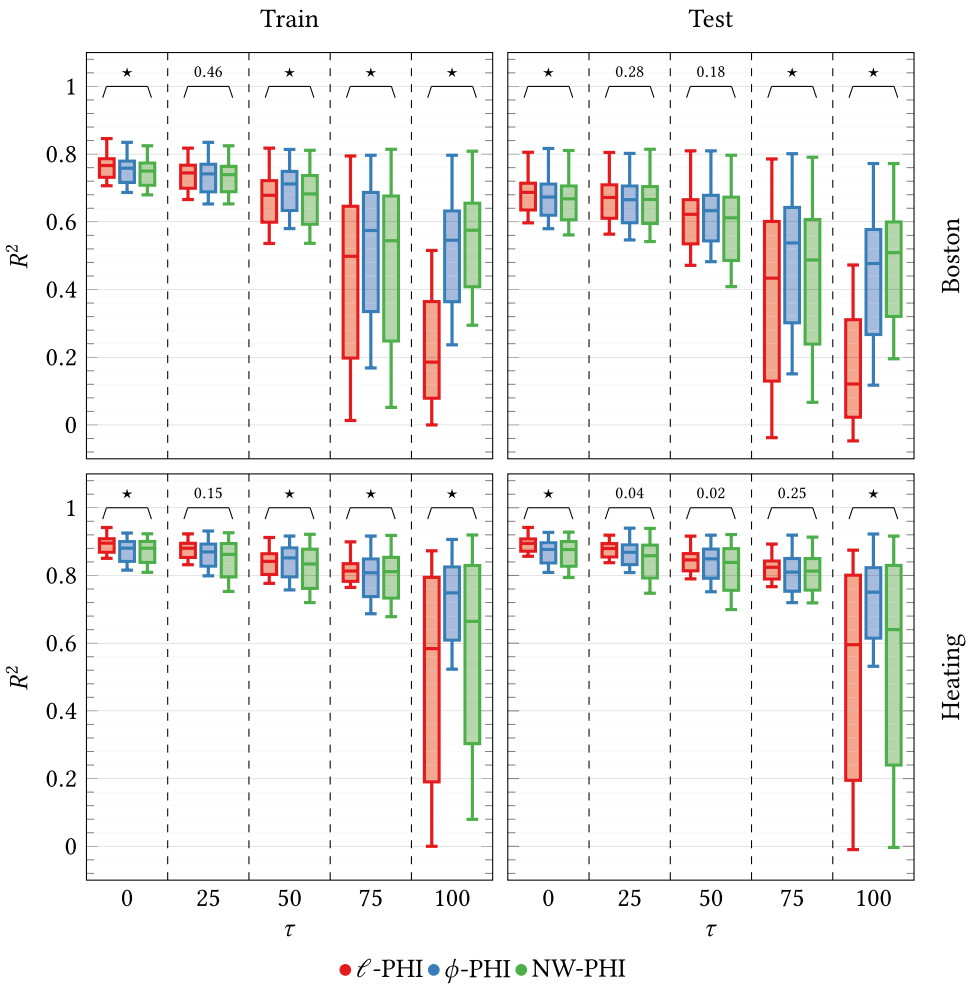

Fig. 6. Distribution of R^2 for different levels of τ on training and test sets w.r.t. the different simulated user profiles. For each τ statistical significance is reported.

total of $10 \cdot 2 \cdot 3 \cdot 3 = 180$ runs. At the end of each run, we re-evaluate each solution found on the test set, to have an estimate of its generality.

From each Pareto front (as determined on the training set), we take the formulae corresponding to different levels of interpretability and compare their performance in terms of accuracy. To this end, we rank the solutions in each front by estimated interpretability, and use the percentile τ to represent the position in the rank ($\tau = 100$ means maximal interpretability and minimal accuracy, $\tau = 0$ means minimal interpretability and maximal accuracy). This allows us to compare models even though the underlying interpretability measure, i.e., PHI, is different.

Before delving into quantitative results, in Table 2 we show examples of models, and their accuracy for different percentile of interpretability τ, produced by GP. Qualitatively, it can be observed that the larger τ is, the simpler (and less accurate) the formulae are, as reasonable to expect.

Figure 6 shows distribution of train and test accuracy—in terms of R^2 score—of the aforementioned models, for $\tau \in \{0, 25, 50, 75, 100\}$. We also report above each triplet of box-plots (corresponding to the three PHIs) the p-values resulting from a Kruskal-Wallis statistical test,

with the null hypothesis that the considered distributions are not different from one another. We use a $\star$ to indicate cases with a p-value smaller than 0.01.

Figure 6 shows a tradeoff between interpretability and accuracy. In fact, for all PHIs and for both datasets, the higher the τ the lower the (distribution of) R^2. This is to be expected, yet it serves as a sanity check for verifying that ML-PIE works correctly. Assuming each PHI corresponds to a different user profile, i.e., to a different subjective notion of interpretability, we can compare the distributions of R^2 relative to the employed PHIs for each percentile τ, and assess whether having a different PHI induces finding differently-accurate models. In particular, according to the reported p-values, in 8 out of 10 cases for the training set, and in 7 out of 10 cases for the test set, the samples appear originated by different distributions. Thus, in most cases, using a different PHI does lead to a different level of accuracy. In other words, this implies that the tradeoff is strongly dependent on the subjective notion of interpretability of the user. This result suggests that GP may need to be configured differently for different users. For example, if the user needs very accurate models but their notion of interpretability makes it hard to discover accurate models, then GP must be set to run with a large budget.

6.4.2 Robustness of the Search Performance with Respect to User Engagement Profile. In the previous experiment, we simulated a constant rate of response of the user. In this section, we wish to assess whether changing this rate of response changes the behavior of GP. We thus repeat the previous experiment using two different user engagement profiles: a lazy-start profile and a lazy-end profile. For the lazy-start profile, we set the simulated user to provide feedback every 8 s for the first half of the model search (in terms of number of generations), and every 2 s for the remainder of the experiment; vice versa for the the lazy-end profile (2 s for the first half, 8 s for the second half). Moreover, we also consider an additional configuration where we use the estimator learned with a constant rate of response of 5 s, and use it directly from the start, i.e., as if it was an oracle. In this case, the user is not involved (i.e., the oracle is not updated). Similarly to the previous experiment, we repeat the experiment 10 times for each configuration, for a total of $10 \cdot 2 \cdot 3 \cdot 3 \cdot 3 = 540$ runs, and at the end of each run, we re-evaluate all solutions on the test set.

While in the previous experiment we were interested in assessing whether using different PHIs leads to different accuracy, now we are interested to see if the overall fronts are different when the user's rate of response changes. To that end, we employ the **HyperVolume** (**HV**) indicator, which measures the "size of the space covered" by the solutions in the front [Zitzler and Künzli 2004]. Intuitively, a larger HV indicates a better search performance, as it corresponds to a larger portion of the search space being dominated. Since we train ANNs to rank and not to regress specific interpretability scores, ANNs trained on different runs produce different scores (even if trained for the same PHI). Thus, using those scores produces HVs that cannot be compared with each other. For this reason, we actually compute the PHI of each formula, for the HV computation. This is reasonable because the estimator is supposed to learn to rank models in the same way that PHI does. Moreover, to be able to compare HVs across different PHIs, we normalize PHI scores with min-max normalization, so that all PHI values are in $[0, 1]$.

Figure 7 shows the distribution of HVs for each user profile, diving them by dataset (one per row), PHI (one per column), and train/test set (x-axis). We also report, above each quadruplet of box-plots, the p-values resulting from a Kruskall-Wallis statistical test with the null hypothesis of equality of the distributions.

From the plots and the p-values, we can see that there are no significant differences among the HVs resulting from different engagement profiles. Hence, we can conclude that ML-PIE is robust to different user behaviors, provided that a decent amount of feedback is given throughout the model search process. Moreover, considering the results obtained with the oracles, we see that

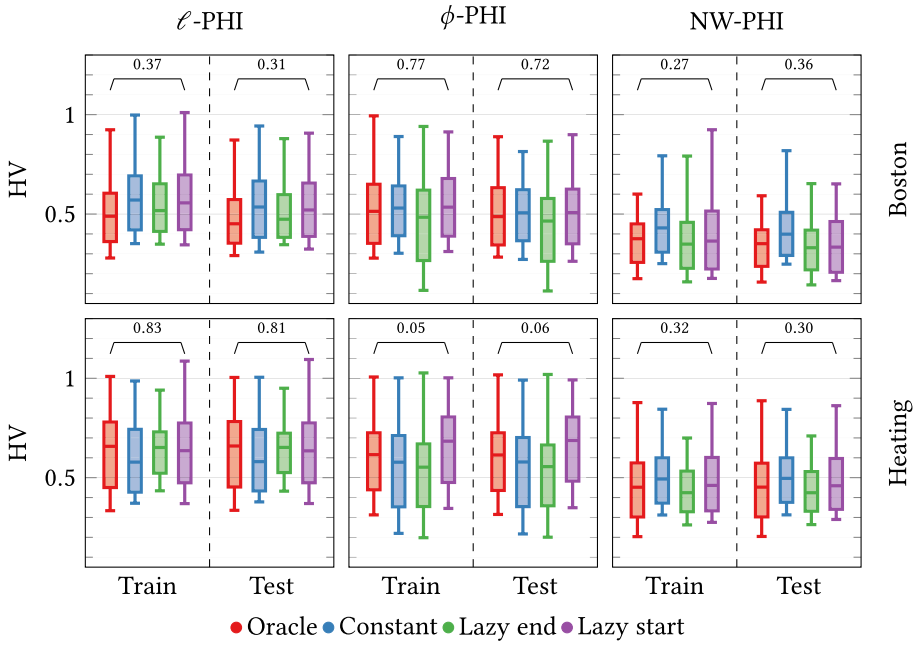

Fig. 7. HVs distribution for different engagement profiles.

model search is not compromised by the fact that the objective of interpretability changes over time.

These results support the hypothesis that an approach like ML-PIE is a feasible one to pursue. Indeed, having the human in the loop may ultimately save time for the user, since the user annotates relevant models that the model search is discovering; rather than a collection of examples taken off-line, which may be out-of-distribution for a specific instance of model search.

Take-home message. We simulate the intervention of the user during the feedback collection process with different PHIs. We test different user behaviors with different engagement trends. We observe, by analyzing the final Pareto fronts, that the more interpretable the formulae, the less accurate. This confirms that our general estimator is able to capture the tested PHIs and it can be integrated within an arbitrary GP-based evolutionary process. Moreover, we assess that using a different PHI may lead to a different level of accuracy, and thus the qualitative performance of the discovered models also depends upon the interpretability notion to learn. Hence, according to the interpretability notion, there may be the need to change the GP parameters (e.g., population size and number of generations) accordingly. Finally, by analyzing different user profiles, we demonstrate that our framework is robust to different user behaviors, provided that enough feedback is collected.

6.5 Experiment 3: Survey on Real Users

As last experiment, we deploy ML-PIE with actual users. We involve 42 participants, who are B.Sc. and M.Sc., and Ph.D. students from computer science and engineering from the University of Trieste, Italy. We do not give any kind of reward to the users for performing the experiment. The users are presented with a web interface, where the landing page describes the task to be performed (repeatedly select which of two formulae is more interpretable), one time for the Boston

dataset, and one time for the Heating dataset. The order by which the datasets are presented is randomized for each user. When the user clicks "start on dataset Boston/Heating", ML-PIE is started and the interface of Figure 2 is presented. ML-PIE runs with the same settings as in the previous experiments (population size of 200, 50 generations, i.e., approximately 10 minutes of runtime per dataset).

After ML-PIE terminates on a dataset, a further feedback page is presented. On this page, the user is presented with 4 pairs of formulae. The first pair contains the formula at $\tau = 25$ from the front obtained by the very ML-PIE run upon which the user had been giving feedback, to be confronted with another formula obtained in an off-line run that used ϕ as notion of interpretability, and that has the closest accuracy to the first formula (for the same dataset and train-test split). The other pairs are obtained in a similar manner, changing τ to 5, and/or ϕ to ℓ. We do not tell the user which formula was obtained with ML-PIE, and randomize whether the formula obtained with ML-PIE appears as first or second in the pair (blind test). In this comparison, besides indicating preference for the first or second formula, the user can indicate that they find both formulae to be equally (un)interpretable. Ultimately, our survey should capture whether users actually prefer the personalized approach that ML-PIE is intended to provide.

In Figure 8, we show the results of the survey conducted with users. We observe that, on average, users tend to prefer models found off-line using ℓ or ϕ, rather than ML-PIE-based ones. This is a negative result that may be due to several reasons: users are not domain experts and, also, they are not much involved in the feedback collection process (no reward is given). We delve deeper into this by looking at quantitative results. We can observe in Figure 9 that we barely reach 50 feedback rounds during these experiments. If we compare this result with Figure 4, we see that, typically across different PHIs, the estimator cannot learn to rank models well when trained with less than 50 feedback rounds. Indeed, Figure 9 also shows that the amount of mispredictions—we take the relative ranking between the two formulae in the query as predicted by the estimator and compare it with the user's feedback—is relatively large (0.5 corresponds to coin flip).

To better understand this outcome, we analyze the results presented in [Virgolin et al. 2021]. In that work, the users gave more feedback on average than here (100 instead of 50 by the end of an ML-PIE run). In other words, users had been more engaged in [Virgolin et al. 2021]. Another key difference between our study and the one of Virgolin et al. [2021] is that the encoding used in that study was smaller (it relied on four formulae features) and better tailored for SR than the one used in this experiment, i.e., counts encoding. Hence, although attempting to use more general encodings is an interesting avenue for making ML-PIE more general and easy to deploy across different problems, it may be the case that general encodings are too large to be effective for a human-in-the-loop setting. Yet, it may be the case that further effort can be done in designing better warm-up strategies, so that very limited feedback is needed to achieve satisfactory results.

Take-home message. We test our framework with real users involved in the feedback collection process. We highlight that if a low number of feedback is provided then the user, who is not a domain expert in our experiments, tends to prefer other models than those built with ML-PIE. On the other hand, ML-PIE-based models are preferred in cases in which many feedbacks are given. This highlights that this system has the potential to discover interpretable models. However, according to the interpretability definition to approximate, the system may need a reasonably large amount of feedback for an accurate estimation.

7 CONCLUSION

In this article, we investigated several aspects to generalize *ML-PIE*, a human-in-the-loop approach to learn personalized interpretable models.

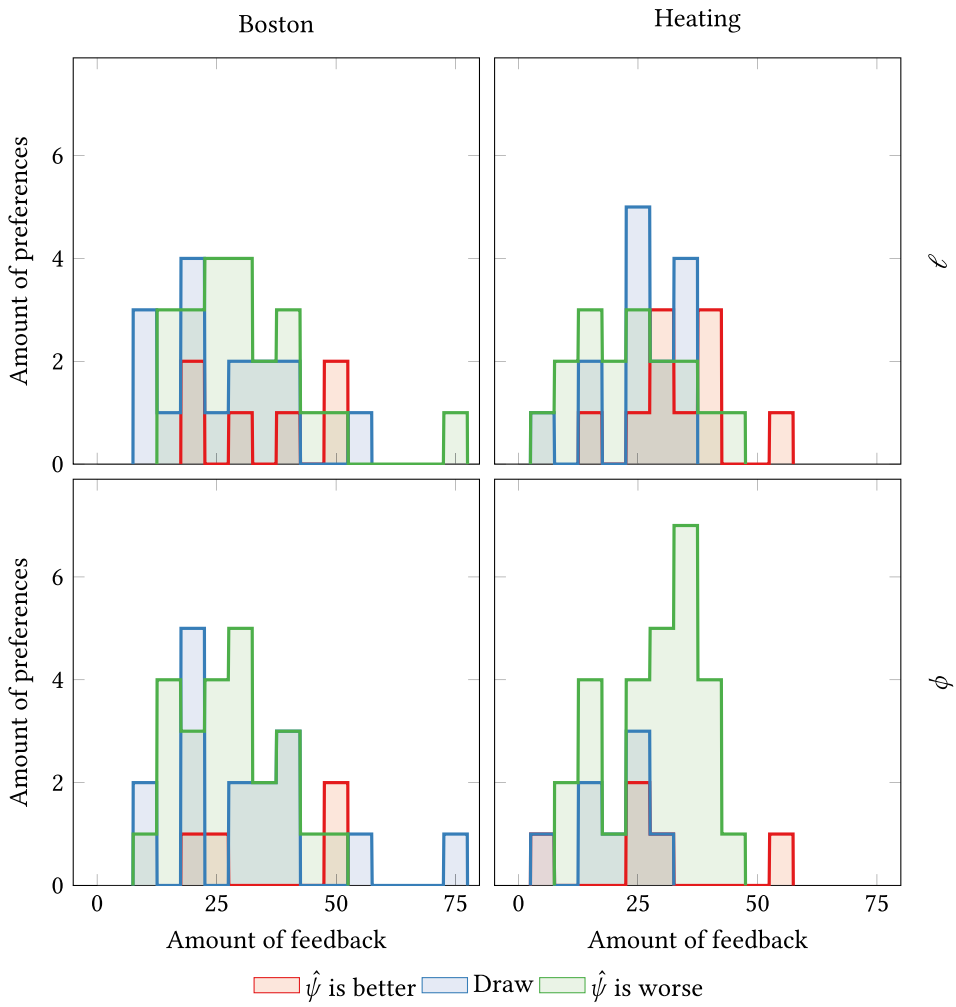

Fig. 8. Amount of preferences for each model collected in the survey at the end of the user study w.r.t. the amount of feedback provided by the user during the GP run. We consider bins of size 5 for the x-axis.

We proposed three different representations, at varying level of complexity and expressiveness, that can be employed to map GP trees with arbitrary function and terminal sets to fixed-sized numerical embeddings. This way, we enabled the original *ML-PIE* to be applied even with problems other than SR, which will be experimentally verified in future works.

We studied how well these representations enable an estimator of interpretability (here, an artificial neural network) to learn to rank models in agreement with different PHIs from the literature (i.e., different simulated users). We found that while larger, more complex representations allow the estimator to better approximate more complex PHIs, provided that a large training is executed, simpler representations require much less feedback to achieve decent results, even though they struggle to further improve accuracy when more feedback is provided.

By using GP as model search algorithm, we then simulated different user profiles and assessed how these affect what models ML-PIE can discover, in terms of accuracy and HV of the Pareto fronts. We found that using different PHIs, to simulate what the user may want, results in different

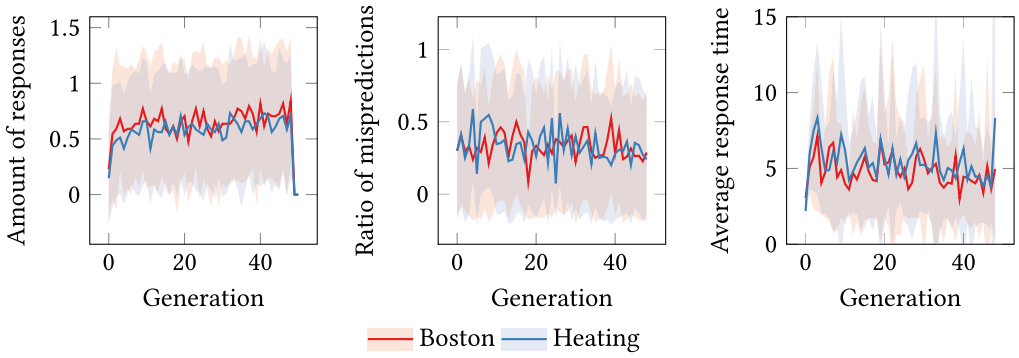

Fig. 9. For each problem, mean and standard deviation range of the amount of responses, ratio of mispredictions the estimator makes in ranking the models (prior to the user's feedback signal), and average response time, along generations.

Table 2. Examples of Models Found by the Bi-objective GP with a Simulated User (NW-PHI), Employing Random Sampling within Active Learning and a ϕ-driven Warm-up for the Interpretability Estimator

		R^2		
Dataset	τ	Train	Test	Model
Boston	10	0.81	0.59	$\max(x_{12}, -0.15 - 4.39) - \max(x_9, x_5) + x_9 - x_3$
		0.77	0.78	$x_{12} - x_3 - x_{12} - \max(x_{12}, x_5)$
		0.75	0.72	$x_{12} - \max\left({x_1}^3 x_3 x_6, x_5 - x_{12} + x_{10}\right)$
	50	0.71	0.56	$x_{12} - x_5 - x_3 + x_{10}$
		0.72	0.68	$x_{12} - \max(x_{12} - \max(x_6, x_5), x_5)$
		0.72	0.63	$(x_4 + x_{10})\, x_5 - x_5 + x_{12}$
	90	0.57	0.54	$x_5 - x_8$
		0.27	0.29	$x_{11} - x_{11} + x_5 - x_8 - x_0$
		0.42	0.43	$x_{12} + \max(x_7, x_2 - x_7 - x_7)$
Heating	10	0.9	0.9	$x_6 - \ln(\lvert x_0 - \ln(\lvert x_4 \rvert)\rvert)$
		0.9	0.86	$x_6 + 4.61 x_2 + 2.33^3 (x_6 - x_3) - \max(x_2 x_2, x_5 + x_0) - \ln\left(\lvert \frac{x_4}{-3.75} \rvert\right)$
		0.93	0.94	$\ln(\lvert 0.14 - x_0 \rvert) - x_2 - \left(x_6 - x_0 - -2.22^3\right)\left({x_6}^3 + x_4\right)$
	50	0.82	0.79	$x_0 - x_4 - x_0 - \ln(\lvert x_3 \rvert)$
		0.9	0.86	$x_1 - x_4 - x_2 - x_4 - x_6$
		0.91	0.93	$x_3 - 4.5 - x_4 - x_6 + x_4 - x_0 - x_4$
	90	0.7	0.7	$x_6 - x_1 - x_4$
		0.8	0.73	$x_6 - x_3 - x_0 - x_2 - x_6$
		0.68	0.66	$x_0 - x_4 - x_0 - x_6 - x_0$

For each dataset, we report the results of three runs. We sample models according to their interpretability percentile (τ) in the Pareto front—higher τ meaning higher interpretability, and report the R^2 score of each model on both train and test sets.

obtainable accuracy for the models. At the same time, ML-PIE showed a good robustness with respect to the rate at which the (simulated) user provides feedback.

Finally, we involved real users (students) to use ML-PIE and indicate, in a blind survey, whether they prefer models learned with the system over models learned with unpersonalized PHIs. This

last experiment showed a negative result, suggesting that the use of a general but large encoding can require too much feedback for ML-PIE to be successful. Further research is needed to assess whether this problem can be ameliorated with better warm-up strategies and better types of general representations, or designing problem-specific representations is the most promising avenue for real-world usage.

ACKNOWLEDGMENTS

The authors wish to thank the participants to the feedback collection for their patience and dedication.

REFERENCES

Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: A survey on explainable artificial intelligence (XAI). *IEEE Access* 6 (2018), 52138–52160. DOI: https://doi.org/10.1109/ACCESS.2018.2870052

Jaume Bacardit, Alexander EI Brownlee, Stefano Cagnoni, Giovanni Iacca, John McCall, and David Walker. 2022. The intersection of evolutionary computation and explainable AI. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1757–1762.

Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2016. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering* 28, 5 (2016), 1217–1230.

Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. 2017. Active learning of regular expressions for entity extraction. *IEEE Transactions on Cybernetics* 48, 3 (2017), 1067–1080.

Michaela Benk and Andrea Ferrario. 2020. Explaining interpretable machine learning: Theory, methods and applications. *Methods and Applications* (December 11, 2020), 87 pages. DOI: https://dx.doi.org/10.2139/ssrn.3748268

J. Blank and K. Deb. 2020. pymoo: Multi-objective optimization in python. *IEEE Access* 8 (2020), 89497–89509. DOI: https://doi.org/10.1109/ACCESS.2020.2990567

Leonard A. Breslow and David W. Aha. 1997. Simplifying decision trees: A survey. *The Knowledge Engineering Review* 12, 01 (1997), 1–40.

Karina Brotto Rebuli, Mario Giacobini, Sara Silva, and Leonardo Vanneschi. 2023. A comparison of structural complexity metrics for explainable genetic programming. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 539–542.

Alberto Cano, Amelia Zafra, and Sebastián Ventura. 2013. An interpretable classification rule mining algorithm. *Information Sciences* 240 (2013), 1–20. DOI: https://doi.org/10.1016/j.ins.2013.03.038

Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. *Advances in Neural Information Processing Systems* 30 (2017), 4299–4307.

Leonardo Lucio Custode and Giovanni Iacca. 2023. Evolutionary learning of interpretable decision trees. *IEEE Access* 11 (2023), 6169–6184. DOI: 10.1109/ACCESS.2023.3236260

Leonardo Lucio Custode and Giovanni Iacca. 2021. A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In *Proceedings of the 2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 1–8.

Leonardo Lucio Custode and Giovanni Iacca. 2022. Interpretable AI for policy-making in pandemics. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion (GECCO'22)*. Association for Computing Machinery, New York, NY, 1763–1769. DOI: https://doi.org/10.1145/3520304.3533959

Leonardo Lucio Custode and Giovanni Iacca. 2022a. Interpretable pipelines with evolutionary optimized modules for reinforcement learning tasks with visual inputs. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 224–227.

Arun Das and Paul Rad. 2020. Opportunities and challenges in explainable artificial intelligence (xai): A survey. arXiv:2006.11371. Retrieved from https://arxiv.org/abs/2006.11371

Junio De Freitas, Gisele L. Pappa, Altigran S. da Silva, Marcos A. Gonc, Edleno Moura, Adriano Veloso, Alberto H. F. Laender, and Moisés G. de Carvalho. 2010. Active learning genetic programming for record deduplication. In *Proceedings of the IEEE Congress on Evolutionary Computation*, IEEE, Barcelona, 1–8. DOI: https://doi.org/10.1109/CEC.2010.5586104

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation* 6, 2 (2002), 182–197. DOI: https://doi.org/10.1109/4235.996017

Persi Diaconis and Ronald L. Graham. 1977. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 2 (1977), 262–268.

Derek Doran, Sarah Schulz, and Tarek R. Besold. 2017. What does explainable AI really mean? A new conceptualization of perspectives. arXiv:1710.00794. Retrieved from https://arxiv.org/abs/1710.00794

Filip Karlo Došilović, Mario Brčić, and Nikica Hlupić. 2018. Explainable artificial intelligence: A survey. In *Proceedings of the 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 0210–0215. DOI: https://doi.org/10.23919/MIPRO.2018.8400040

Aniko Ekart and Sandor Z. Nemeth. 2001. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines* 2, 1 (2001), 61–73.

Andrea Ferigo, Leonardo Lucio Custode, and Giovanni Iacca. 2023. Quality diversity evolutionary learning of decision trees. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing (SAC'23)*. Association for Computing Machinery, New York, NY, 425–432. DOI: https://doi.org/10.1145/3555776.3577591

Alberto Fernandez, Francisco Herrera, Oscar Cordon, Maria Jose del Jesus, and Francesco Marcelloni. 2019. Evolutionary fuzzy systems for explainable artificial intelligence: Why, when, what for, and where to? *IEEE Computational Intelligence Magazine* 14, 1 (2019), 69–81.

Richard P. Feynman, Robert B. Leighton, and Matthew Sands. 1965. The feynman lectures on physics; vol. i. *American Journal of Physics* 33, 9 (1965), 750–752.

Alex A. Freitas. 2014. Comprehensible classification models: A position paper. *ACM SIGKDD Explorations Newsletter* 15, 1 (2014), 1–10.

Randy Goebel, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. 2018. Explainable AI: The new 42?. In *Proceedings of the Machine Learning and Knowledge Extraction*. Andreas Holzinger, Peter Kieseberg, A. Min Tjoa, and Edgar Weippl (Eds.), Springer International Publishing, Cham, 295–303.

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini, and Fosca Giannotti. 2018a. Local rule-based explanations of black box decision systems. arXiv:1805.10820. Retrieved from https://arxiv.org/abs/1805.10820

Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018b. A survey of methods for explaining black box models. *ACM Computing Surveys* 51, 5 (2018), 1–42.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C. Courville. 2017. Improved training of wasserstein gans. *Advances in Neural Information Processing Systems* 30 (2017), 5767–5777.

Hani Hagras. 2018. Toward human-understandable, explainable AI. *Computer* 51, 9 (2018), 28–36.

David Harrison Jr and Daniel L. Rubinfeld. 1978. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management* 5, 1 (1978), 81–102.

Joshua James Hatherley. 2020. Limits of trust in medical AI. *Journal of Medical Ethics* 46, 7 (2020), 478–481.

Daniel Hein, Steffen Udluft, and Thomas A. Runkler. 2018. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence* 76 (2018), 158–169. DOI: https://doi.org/10.1016/j.engappai.2018.09.007

Robert R. Hoffman, Shane T. Mueller, Gary Klein, and Jordan Litman. 2018. Metrics for explainable AI: Challenges and prospects. arXiv:1812.04608. Retrieved from https://arxiv.org/abs/1812.04608

Andreas Holzinger. 2018. From machine learning to explainable AI. In *Proceedings of the 2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. 55–66. DOI: https://doi.org/10.1109/DISA.2018.8490530

Johan Huysmans, Karel Dejaeger, Christophe Mues, Jan Vanthienen, and Bart Baesens. 2011. An empirical evaluation of the comprehensibility of decision table, tree and rule based predictive models. *Decision Support Systems* 51, 1 (2011), 141–154.

Robert Isele and Christian Bizer. 2013. Active learning of expressive linkage rules using genetic programming. *Journal of Web Semantics* 23 (2013), 2–15. DOI: https://doi.org/10.1016/j.websem.2013.06.001

Yacine Izza, Alexey Ignatiev, and Joao Marques-Silva. 2020. On explaining decision trees. arXiv:2010.11034. Retrieved from https://arxiv.org/abs/2010.11034

Noman Javed, Fernand R. Gobet, and Peter Lane. 2022. Simplification of genetic programs: A literature survey. *Data Mining and Knowledge Discovery* 36 (2022), 1279–1300. DOI: https://doi.org/10.1007/s10618-022-00830-7

Anna Jobin, Marcello Ienca, and Effy Vayena. 2019. The global landscape of AI ethics guidelines. *Nature Machine Intelligence* 1 (2019), 389–399. DOI: https://doi.org/10.1038/s42256-019-0088-2

Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285. DOI: https://doi.org/10.1613/jair.301

Maarten Keijzer. 2003. Improving symbolic regression with interval arithmetic and linear scaling. In *Proceedings of the European Conference on Genetic Programming*. Springer, 70–82.

Maarten Keijzer. 2004. Scaled symbolic regression. *Genetic Programming and Evolvable Machines* 5, 3 (2004), 259–269.

Varun Kompella, Roberto Capobianco, Stacy Jong, Jonathan Browne, Spencer Fox, Lauren Meyers, Peter Wurman, and Peter Stone. 2020. Reinforcement learning for optimization of COVID-19 mitigation policies. arXiv:2010.10560. Retrieved from https://arxiv.org/abs/2010.10560

Boris Kovalerchuk, Muhammad Aurangzeb Ahmad, and Ankur Teredesai. 2021. Survey of explainable machine learning with visual and granular methods beyond quasi-explanations. *Interpretable Artificial Intelligence: A Perspective of Granular Computing. Part of the Studies in Computational Intelligence book series (SCI, volume 937, chapter 8)*, 217–267. DOI: https://doi.org/10.1007/978-3-030-64949-4_8

John R. Koza. 1994. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing* 4, 2 (1994), 87–112.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabricio Olivetti de Franca, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. 2021. Contemporary symbolic regression methods and their relative performance. In *Proceedings of the 35th Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Himabindu Lakkaraju, Stephen H. Bach, and Jure Leskovec. 2016. Interpretable decision sets: A joint framework for description and prediction. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1675–1684.

Andrew Lensen. 2021. Mining feature relationships in data. In *Proceedings of the European Conference on Genetic Programming (Part of EvoStar)*. Springer, 247–262.

Andrew Lensen, Bing Xue, and Mengjie Zhang. 2020. Genetic programming for evolving a front of interpretable models for data visualization. *IEEE Transactions on Cybernetics* 51, 11 (2020), 5468–5482.

Zachary C. Lipton. 2018. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue* 16, 3 (2018), 31–57.

Dazhuang Liu, Marco Virgolin, Tanja Alderliesten, and Peter Bosman. 2022. Evolvability degeneration in multi-objective genetic programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 973–981.

Sean Luke and Liviu Panait. 2001. A survey and comparison of tree generation algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*. Citeseer, 81–88.

Zahra Mahoor, Jack Felag, and Josh Bongard. 2017. Morphology dictates a robot's ability to ground crowd-proposed language. arXiv:1712.05881. Retrieved from https://arxiv.org/abs/1712.05881

Eric Medvet, Alberto Bartoli, Barbara Carminati, and Elena Ferrari. 2015. Evolutionary inference of attribute-based access control policies. In *Proceedings of the International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 351–365.

Yi Mei, Qi Chen, Andrew Lensen, Bing Xue, and Mengjie Zhang. 2022. Explainable artificial intelligence by genetic programming: A survey. *IEEE Transactions on Evolutionary Computation* 27, 3 (2022), 621–641. DOI: https://doi.org/10.1109/TEVC.2022.3225509

Risto Miikkulainen, Olivier Francon, Elliot Meyerson, Xin Qiu, Darren Sargent, Elisa Canzani, and Babak Hodjat. 2021. From prediction to prescription: Evolutionary optimization of nonpharmaceutical interventions in the COVID-19 pandemic. *IEEE Transactions on Evolutionary Computation* 25, 2 (2021), 386–401.

Christoph Molnar. 2020. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*. Leanpub book, ISBN-10 0244768528, ISBN-13 978-0244768522, 318 pages.

Christoph Molnar, Gunnar König, Julia Herbinger, Timo Freiesleben, Susanne Dandl, Christian A. Scholbeck, Giuseppe Casalicchio, Moritz Grosse-Wentrup, and Bernd Bischl. 2020. Pitfalls to avoid when interpreting machine learning models. *XXAI: Extending Explainable AI Beyond Deep Models and Classifiers, ICML 2020 Workshop*. http://eprints.cs.univie.ac.at/6427/

Christopher J. Moore, Alvin J. K. Chua, Christopher P. L. Berry, and Jonathan R. Gair. 2016. Fast methods for training gaussian processes on large datasets. *Royal Society Open Science* 3, 5 (2016), 160125.

Aidan Murphy, Gráinne Murphy, Jorge Amaral, Douglas MotaDias, Enrique Naredo, and Conor Ryan. 2021. Towards incorporating human knowledge in fuzzy pattern tree evolution. In *Proceedings of the European Conference on Genetic Programming (Part of EvoStar)*. Springer, 66–81.

Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the Icml*.

Michael O'Neill and Conor Ryan. 2001. Grammatical evolution. *IEEE Transactions on Evolutionary Computation* 5, 4 (2001), 349–358.

Michael O'Neill, Riccardo Poli, William B. Langdon, and Nicholas F. McPhee. 2009. A field guide to genetic programming. *Genetic Programming and Evolvable Machines* 10, 2 (2009), 229–230. DOI: https://doi.org/10.1007/s10710-008-9073-y

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proceedings of the Advances in Neural Information Processing Systems 32*. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (Eds.), Curran Associates, Inc., 8024–8035. Retrieved from http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

Forough Poursabzi-Sangdeh, Daniel G. Goldstein, Jake M. Hofman, Jennifer Wortman Wortman Vaughan, and Hanna Wallach. 2021. Manipulating and measuring model interpretability. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. 1–52.

Apostolos F. Psaros, Xuhui Meng, Zongren Zou, Ling Guo, and George Em Karniadakis. 2023. Uncertainty quantification in scientific machine learning: Methods, metrics, and comparisons. *Journal of Computational Physics* 477 (2023), 111902. DOI: https://doi.org/10.1016/j.jcp.2022.111902

Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-Yao Huang, Zhihui Li, Brij B. Gupta, Xiaojiang Chen, and Xin Wang. 2021. A survey of deep active learning. *ACM Computing Surveys* 54, 9 (2021), 1–40.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1135–1144.

Luigi Rovito, Lorenzo Bonin, Luca Manzoni, and Andrea De Lorenzo. 2022. An evolutionary computation approach for twitter bot detection. *Applied Sciences* 12, 12 (2022), 5915–5939. DOI: https://doi.org/10.3390/app12125915

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

Omer Sagi and Lior Rokach. 2020. Explainable decision forest: Transforming a decision forest into an interpretable tree. *Information Fusion* 61 (2020), 124–138. DOI: https://doi.org/10.1016/j.inffus.2020.03.013

Wojciech Samek and Klaus-Robert Müller. 2019. Towards explainable artificial intelligence. In *Proceedings of the Explainable AI: Interpreting, Explaining and Visualizing Deep Learning*. Springer, 5–22.

Jimmy Secretan, Nicholas Beato, David B. D'Ambrosio, Adelein Rodriguez, Adam Campbell, Jeremiah T. Folsom-Kovarik, and Kenneth O. Stanley. 2011. Picbreeder: A case study in collaborative evolutionary exploration of design space. *Evolutionary Computation* 19, 3 (2011), 373–403.

Burr Settles. 2009. *Active Learning Literature Survey*. University of Wisconsin-Madison, Department of Computer Sciences. http://digital.library.wisc.edu/1793/60660

Shubham Sharma, Jette Henderson, and Joydeep Ghosh. 2020. CERTIFAI: A common framework to provide explanations and analyse the fairness and robustness of black-box models. *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, (AIES'20, New York, NY, USA)*, Association for Computing Machinery, New York, NY, 166–172. https://doi.org/10.1145/3375627.3375812

Guido F. Smits and Mark Kotanchek. 2005. Pareto-front exploitation in symbolic regression. In *Proceedings of the Genetic Programming Theory and Practice II*. Springer, 283–299.

Charles Spearman. 1906. Footrule for measuring correlation. *British Journal of Psychology* 2, 1 (1906), 89.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

Guolong Su, Dennis Wei, Kush R. Varshney, and Dmitry M. Malioutov. 2015. Interpretable two-level boolean rule learning for classification. arXiv:1511.07361. Retrieved from https://arxiv.org/abs/1511.07361

Robert Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)* 58, 1 (1996), 267–288.

Alexander Trott, Sunil Srinivasa, Douwe van der Wal, Sebastien Haneuse, and Stephan Zheng. 2021. Building a foundation for data-driven, interpretable, and robust policy design using the ai economist. arXiv:2108.02904. Retrieved from https://arxiv.org/abs/2108.02904

Athanasios Tsanas. 2012. *Accurate Telemonitoring of Parkinson's Disease Symptom Severity Using Nonlinear Speech Signal Processing and Statistical Machine Learning*. Ph.D. Dissertation. Oxford University, UK.

Athanasios Tsanas and Angeliki Xifara. 2012. Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy and Buildings* 49 (2012), 560–567. DOI: https://doi.org/10.1016/j.enbuild.2012.03.003

Ryan J. Urbanowicz and Jason H. Moore. 2009. Learning classifier systems: A complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, Volume 2009, Article ID 736398, 25 pages. DOI: 10.1155/2009/736398

Berk Ustun and Cynthia Rudin. 2016. Supersparse linear integer models for optimized medical scoring systems. *Machine Learning* 102, 3 (2016), 349–391.

Mathurin Videau, Alessandro Leite, Olivier Teytaud, and Marc Schoenauer. 2022. Multi-objective genetic programming for explainable reinforcement learning. In *Proceedings of the European Conference on Genetic Programming (Part of EvoStar)*. Springer, 278–293.

Giulia Vilone and Luca Longo. 2020. Explainable artificial intelligence: A systematic review. arXiv:2006.00093. Retrieved from https://arxiv.org/abs/2006.00093

Marco Virgolin. 2022. genepro. Retrieved from https://github.com/marcovirgolin/genepro. Accessed 10 July 2022.

Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. 2019. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 1084–1092.

Marco Virgolin, Tanja Alderliesten, and Peter A. N. Bosman. 2020a. On explaining machine learning models by evolving crucial and compact features. *Swarm and Evolutionary Computation* 53 (2020), 100640. DOI: https://doi.org/10.1016/j.swevo.2019.100640

Marco Virgolin, Andrea De Lorenzo, Eric Medvet, and Francesca Randone. 2020b. Learning a formula of interpretability to learn interpretable formulas. In *Proceedings of the Parallel Problem Solving from Nature - PPSN XVI: 16th International Conference, PPSN 2020, Leiden, The Netherlands, September 5-9, 2020 Part II*. Springer-Verlag, Berlin, 79–93. DOI: https://doi.org/10.1007/978-3-030-58115-2_6

Marco Virgolin, Andrea De Lorenzo, Francesca Randone, Eric Medvet, and Mattias Wahde. 2021. Model learning with personalized interpretability estimation (ML-PIE). In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 1355–1364.

Marco Virgolin and Solon P. Pissis. 2022. Symbolic regression is NP-hard. *Transactions on Machine Learning Research*. https://openreview.net/forum?id=LTiaPxqe2e

Ekaterina (Katya) Vladislavleva, Guido Smits, and Dick den Hertog. 2009. Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Transactions on Evolutionary Computation* 13, 2 (2009), 333–349. DOI: https://doi.org/10.1109/TEVC.2008.926486

Christopher J. C. H. Watkins and Peter Dayan. 1992. Q-learning. *Machine Learning* 8, 3 (1992), 279–292.

Dennis G. Wilson, Sylvain Cussat-Blanc, Hervé Luga, and Julian F. Miller. 2018. Evolving simple programs for playing atari games. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 229–236.

Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. 2019. Explainable AI: A brief survey on history, research areas, approaches and challenges. In *Proceedings of the CCF International Conference on Natural Language Processing and Chinese Computing*. Springer, 563–574.

Yazhou Yang and Marco Loog. 2016. Active learning using uncertainty information. In *Proceedings of the 2016 23rd International Conference on Pattern Recognition (ICPR)*. IEEE, 2646–2651.

Zijun Zhang. 2018. Improved adam optimizer for deep neural networks. In *Proceedings of the 2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–2.

Eckart Zitzler and Simon Künzli. 2004. Indicator-based selection in multiobjective search. In *Proceedings of the International Conference on Parallel Problem Solving from Nature*. Springer, 832–842.

Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 67, 2 (2005), 301–320.

Received 16 November 2022; revised 12 September 2023; accepted 24 January 2024