Formal XAI via Syntax-Guided Synthesis

Katrine Bjørner[1], Samuel Judson[2], Filip Cano[3(✉)], Drew Goldman[4], Nick Shoemaker[2], Ruzica Piskac[2], and Bettina Könighofer[3]

[1] New York University, New York, USA
kbjorner@nyu.edu
[2] Yale University, New Haven, USA
{samuel.judson, ruzica.piskac, nick.shoemaker}@yale.edu
[3] Graz University of Technology, Graz, Austria
{filip.cano, bettina.koenighofer}@iaik.tugraz.at
[4] University of Virginia, Charlottesville, USA
dag5wd@virginia.edu

Abstract. In this paper, we propose a novel application of syntax-guided synthesis to find symbolic representations of a model's decision-making process, designed for easy comprehension and validation by humans. Our approach takes input-output samples from complex machine learning models, such as deep neural networks, and automatically derives interpretable *mimic programs*. A mimic program precisely imitates the behavior of an opaque model over the provided data. We discuss various types of grammars that are well-suited for computing mimic programs for tabular and image input data.

Our experiments demonstrate the potential of the proposed method: we successfully synthesized mimic programs for neural networks trained on the MNIST and the Pima Indians diabetes data sets. All experiments were performed using the SMT-based `cvc5` synthesis tool.

Keywords: Syntax-Guided Synthesis (SyGuS) · Explainable Machine Learning · Program Synthesis · Programming by Example (PbE)

1 Introduction

Complex machine learning models, such as deep neural networks have achieved remarkable success across various domains, including image recognition [29], natural language processing [15], and control [18]. However, the inherently complex and opaque nature of deep neural networks renders insightful human evaluation of their opaque decision logic a challenge. In domains like healthcare, autonomous vehicles, and financial systems, where decisions can have profound consequences on human lives and societal well-being, the explainability of the model's decision making becomes a crucial requirement [25].

K. Bjørner and S. Judson—Equal contribution.

B. Steffen (Ed.): AISoLA 2023, LNCS 14380, pp. 119–137, 2024.
https://doi.org/10.1007/978-3-031-46002-9_7

Explainable AI. The field of explainable AI encompasses a range of methodologies aiming to provide human-understandable insights into complex AI models [1,4,16]. *Global interpretation methods*, such as feature importance and partial dependence plots, focus on understanding the overall behavior of a model across the entire input space [30]. *Local interpretation methods* provide explanations for the model's predictions on a case-by-case basis. Many local interpretation techniques, such as LIME [35] and SHAP [26] are based on training a local surrogate model. Surrogate models are simplified models, such as rule-based models, linear models, or decision trees, that approximate the decision-making of a black-box model locally. For example, LIME generates a surrogate model by perturbing the features of an input data point, observing how the model's predictions change, and then fits a linear model to these perturbed instances. *Decision trees* can serve as surrogate models for local interpretability by training them on a local dataset [12,34]. Decision trees are constructed by recursively partitioning the data based on the most significant features. At each node of the tree, a decision is made based on a specific feature and its corresponding threshold. These decisions create a path from the root node to a leaf node, resulting in a set of if-then rules. Analysing the path for a particular output of the model can reveal the reasons behind the model's decision.

Our Contribution - Mimic Programs. In this paper, we propose to use quantifier-free formulas in first-order logic to explain the model's decisions for a given data set. We call such a formula a *mimic program*.[1] Given a set of data points $\mathsf{Pts} = \{(x, y) \mid x \in \mathbb{R}^d, y \in \{0, 1\}\}$ sampled from a model f, *i.e.*, $f(x) = y$ for all $(x, y) \in \mathsf{Pts}$, a mimic program P_f gives the same output as f for any data point in Pts, i.e., $P_f(x) = f(x) = y$ for all $(x, y) \in \mathsf{Pts}$.

Synthesis of Mimic Programs. We formulate the problem of computing an explainable mimic program P_f as a syntax-guided synthesis (SyGuS) problem [3]. SyGuS augments the synthesis problem with a grammar, also called a syntactic template, from which the mimic program is to be constructed. The syntax-guided synthesis problem then is to find an implementation P_f that respects a given grammar and satisfies the semantic constraints given in form of Pts.

The syntactic constraints of SyGuS serve two purposes. First, it renders the search space tractable for the synthesizer. Picking a good grammar is essential for the scalability of the synthesis tools. A grammar that is too large might result in a large search space, while one that is too small may make the synthesizer unable to find a solution [24]. Second, it applies syntactic restrictions on the space of the mimic programs being searched such that the resulting mimic programs are easy to understand for humans.

In the paper, we discuss suitable grammars for the synthesis of mimic programs. Inspired by the structure of decision trees, we use grammars that allow conditionals (if-then-else control flow). In case the input data is tabular data,

[1] Note that the mathematical expressions of the formula can easily be translated into a programming language such as Python.

we allow the comparisons between variables and constants. Since the set of constants allowed for comparison heavily influences the search space, we discuss several heuristics for selecting those constants. For image data, we pick a simpler grammar that allows the comparison between variables, but does not allow the comparison with constants. This not only makes the search simpler but also ensures that the classification of the mimic program remains robust against changes in image brightness.

Experimental Evaluation. In our experiments, we synthesize mimic programs for neural networks trained on the MNIST data set [23] and the Pima Indians diabetes data set [40]. We use the `cvc5` SMT solver [6] for the synthesis procedure. For the obtained mimic programs, we provide the results of evaluating programs with respect to their interpretability, computation times and accuracy. As a baseline, we compare to surrogate models obtained from training decision trees with the off-the-shelf tool scikit-learn [33].

Outline. In Sect. 2, we discuss an illustrating example in which we compute mimic programs for small image input data. We give the background on SyGuS in Sect. 3. We discuss the synthesis of mimic programs in Sect. 4 and report our experimental results in Sect. 5. Finally, we discuss related work in Sect. 6 and conclude in Sect. 7.

2 Illustrative Example

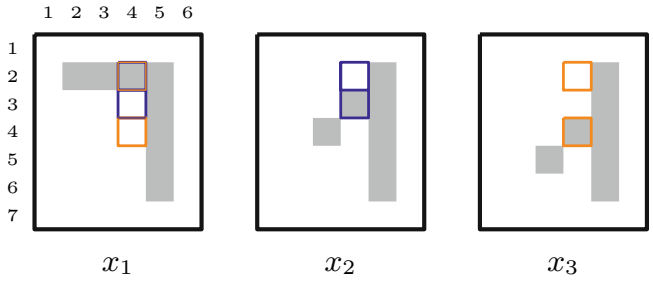


Fig. 1. Illustrative example - mimic programs for an image classification task.

We showcase how mimic programs can be used to explain the decisions of a model trained on image data. In this example, we are given a set of three images that depict handwritten digits, as illustrated in Fig. 1. Additionally, we have given the classification of each image obtained by a model f. Our goal is to compute a mimic program for this data set that explains the decisions of f.

In particular, the images are of size 6×7 pixels and are flattened into a vector $[p_{1,1}, \ldots, p_{6,7}] \in \mathbb{R}^{42}$. Each pixel is a real valued number in [0, 1], where

0 is white and 1 is black. A given black-box model f classifies the picture x_1 as the digit 7 and the pictures x_2 and x_3 as the digit 1.

We start by computing a mimic program $P_{f,1,2}$ for the dataset $\mathsf{Pts}_{1,2} = \{(x_1, 7), (x_2, 1)\}$. In order to compute $P_{f,1,2}$ via SyGuS, we need to define both the semantic constraints and the syntactic constraints (the grammar). The semantic constraints are defined by $\mathsf{Pts}_{1,2}$. Thus, we require that $P_{f,1,2}(x_1) = 7$ and $P_{f,1,2}(x_2) = 1$. For the syntactic constraints, we use a simple ITE-grammar that allows as conditions Boolean combinations of comparisons between pixels values. Invoked on this problem, a SyGuS solver might instantiate the following program:

$$P_{f,1,2} \coloneqq \mathsf{if}\ p_{4,2} > p_{4,3}\ \mathsf{then}\ 7\ \mathsf{else}\ 1.$$

$P_{f,1,2}$ is an interpretable program that mimics the decision making of f for the two given examples. However, it is not a correct mimic program for x_3. Therefore, we extend the set of examples: $\mathsf{Pts}_{1-3} = \{(x_1, 7), (x_2, 1), (x_3, 1)\}$. Using SyGuS, we might now get the synthesized program

$$P_{f,1-3} \coloneqq \mathsf{if}\ (p_{4,2} > p_{4,3}\ \vee\ p_{4,2} > p_{4,4})\ \mathsf{then}\ 7\ \mathsf{else}\ 1.$$

Note that both programs, $P_{f,1,2}$ and $P_{f,1-3}$, only compare the values of two and three pixels respectively, and so their decision logic is easily comprehensible. Another valid mimic program $P'_{f,1-3}$ for the set of points Pts_{1-3} would be

$$P'_{f,1-3} \coloneqq \mathsf{if}\ p_{2,2} > p_{2,3}\ \mathsf{then}\ 7\ \mathsf{else}\ 1.$$

This demonstrates a strength of our approach: adding more input-output examples does not imply a more complicated mimic program. On the contrary, given a large data set of input-output examples, SyGuS finds (for many practical instances) relatively small and interpretable programs that only compare data values relevant to the classification.

3 Preliminaries

A *Syntax-Guided Synthesis* (SyGuS) [3] problem is specified with respect to a background theory $\mathbb{T}$, such as linear real arithmetic (`LRA`) or linear integer arithmetic (`LIA`), that fixes the domain of variables and the types and interpretations of the used functions and predicates.

SyGuS searches for an implementation P_f in form of a quantifier-free first-order logic formula within the theory $\mathbb{T}$ that satisfies two types of constraints:

i *Semantic constraints.* In classical SyGuS, the semantic constraints are given as an arbitrary formula φ built from symbols in the theory $\mathbb{T}$. We work with a special instance of the SyGuS problem, called *Programming-by-Example* (PbE) [21,31], where the semantic constraints are given as a set of input-output examples $\mathsf{Pts} = \{(x, y) \mid x \in X^d, y \in \{0, 1\}\}$, where X is the domain defined within the theory $\mathbb{T}$.

ii *Syntactic constraints.* The syntactic constraints are given as a (possibly infinite) set $\mathcal{E}$ of expressions from $\mathbb{T}$ specified by a context-free grammar G.

The Syntax-Guided Synthesis Problem for Programming-by-Example: The computational problem is then to find an implementation P_f, that is permitted by a grammar G, and such that $\forall(x, y) \in \mathsf{Pts}$ it holds that $P_f(x) = y$ within the theory $\mathbb{T}$.

The grammar $G = (V, N, R, S)$ is a context-free grammar, where V is a set of symbols in the theory $\mathbb{T}$, N is a set of non-terminals, R is a set of production rules such that $R : (N \cup S)^* \to (N \cup V)^*$ and S is the start-symbol. G must also ensure that every sentence generated by the grammar is well-formed for the considered logic. For example, a grammar for synthesizing linear real arithmetic programs must guarantee that no boolean variable is used in an addition operation.

Most SyGuS solvers work in a counterexample-guided refinement loop. Candidates for P_f are enumerated and checked through SMT solving, with the resultant counterexamples informing the adaptive construction of the next candidate.

4 Synthesizing Mimic Programs

In this section, we define mimic programs and discuss their computation. A mimic program serves as a surrogate model for complex opaque models. It is computed from a set of input-output data points sampled from the original model and replicates the decisions of the model in these data points precisely. Additionally, due to the declarative nature of the mimic program and the syntactic restrictions on its structure, the computed mimic programs are generally easy for humans to understand and to analyse.

Definition of Mimic Programs. To replicate machine learning models like deep neural networks, we consider input-output examples of type $\mathsf{Pts} \subseteq \mathbb{R}^d \times \{0, 1\}$. We use the theory of linear real arithmetic (`LRA`), since that theory both captures the arithmetical statements used in statistical inference and has decision procedures available for determining satisfaction modulo $\mathbb{T}$. Within `LRA`, each variable is either a boolean or a real, and the vocabulary consists of boolean and real constants, standard boolean connectives, addition ($+$), comparison ($\leq$), and conditionals (If-Then-Else). We define mimic programs as follows:

Definition 1. *Let G be a context-free grammar, $f : \mathbb{R}^d \to \{0, 1\}$ be a model, and $\mathsf{Pts} \subseteq \mathbb{R}^d \times \{0, 1\}$ be a set of points of size $|\mathsf{Pts}| = n$ that are consistent with f, i.e., $f(x) = y$ for all $(x, y) \in \mathsf{Pts}$. A **mimic program** P_f **on** Pts is a well-formed formula within the theory of `LRA` that is permitted by the grammar G and satisfies that*

$$\forall(x,\, y) \in \mathsf{Pts}.\; P_f(x) = f(x) = y.$$

Synthesis of Mimic Programs. To compute a mimic program P_f using syntax-guided synthesis (SyGuS), we fix `LRA` as background theory and define a set of input-output data points Pts to form the semantic constraints and a grammar G that P_f needs to satisfy. In the following, we give details regarding both the selection of the data points and the grammar.

4.1 Semantic Constraints for Mimic Programs

A mimic program P_f gives the same output as a model f for a given set of data points Pts, i.e., for all $(x, y) \in \mathsf{Pts}$ we have $P_f(x) = f(x) = y$. Depending on the Pts from which the mimic program was created, P_f might also serve as a good *approximation* for the decision-making of the model f in data points not included in Pts.

If Pts is a large enough set that was uniformly sampled on the entire data set, the mimic program can serve as a global surrogate model for f. If Pts was sampled locally around a given data point x^*, the mimic program may serve as a local surrogate model to explain the classification of data points close to x^*. Different strategies, like distance sampling and feature manipulation strategies [35], can be applied to compute samples close to x^*. Note, that P_f gives no correctness guarantees for any data points that are not contained in Pts.

4.2 Syntactic Constraints for Mimic Programs

Designing an effective grammar G is likely to be at least domain- and possibly even dataset- and model-specific. In order to mimic f, G must enable P_f to include statements able to express (approximations of) the statistical patterns that f depends on. But this reliance does not necessarily demand G be complex, as we will show in this section for both image data and tabular data.

Syntactic Constraints for Mimic Programs from Image Data
For image data, we suggest using a very simple grammar that only supports ITE branching and comparison between variables.

We assume that the image data is in a domain of $\mathbb{R}^m \times \mathbb{R}^n$, where $m \times n$ is the size of the input images in pixels. For a given instance $x \in \mathbb{R}^m \times \mathbb{R}^n$, the value $x_{i,j}$ represents the value of the pixel at position i, j. We use the following grammar G_{image} to compute mimic programs for image data:

$$\begin{aligned} \mathcal{B} &:= \top \mid \bot \mid \mathcal{R} \leq \mathcal{R} \mid \text{if } \mathcal{B} \text{ then } \mathcal{B} \text{ else } \mathcal{B}, \\ \mathcal{R} &:= x_{ij}. \end{aligned}$$

Using a simple grammar G_{image} has several advantages:

- *Scalability.* Due to the high-dimensionality of image data, synthesizing mimic programs from images is particularly challenging. A simple grammar limits the search space for P_f. Allowing slighter richer grammars can already have huge negative performance impacts.

- *Robustness.* A mimic program P_f that is permitted by G_{image} is robust to monotonic transformations applied uniformly to the whole image, since it does not allow the comparison of pixels to absolute values. For example, applying the transformation $x_{ij} \mapsto \alpha x_{ij} + k$ for constants k, $\alpha > 0$ on the entire image would make the image brighter, but otherwise leave its structure intact, and the mimic program would still be correct.
- *Interpretability.* More complex rules may also be more difficult for humans to understand. For example, it might be easier to understand the relevance of a branching condition $x_{ij} > x_{i'j'}$, than the reasoning behind a condition like $x_{ij} + 0.29 > x_{i'j'} - 0.12$.

Syntactic Constraints for Mimic Programs from Tabular Data
The most important distinction between tabular and image data is that tabular features are not homogeneous, as is the case for image data. Each features of tabular data has its own meaning and can represent a different quantity and distribution. This makes it difficult to interpret the meaning of comparing the values of different features. Therefore, we consider a grammar that only allows the comparison of individual features with constants.

We assume that the tabular data is in some domain $\mathbb{R}^d$, where d is the number of features (*i.e.*, columns in a dataset). For an instance $x \in \mathbb{R}^d$, the value x_i represents the value of the i-th feature. We use the following grammar $G_{tabular}$ to compute mimic programs for tabular data:

$$\begin{aligned}
\mathcal{F}_0 &:= m_{0,1} \mid m_{0,2} \mid \cdots \mid m_{0,k_0} \\
&\cdots \\
\mathcal{F}_d &:= m_{n,1} \mid m_{n,2} \mid \cdots \mid m_{n,k_d} \\
\mathcal{FC}_0 &:= \mathcal{F}_0 \leq x_0 \mid x_0 \leq \mathcal{F}_0 \\
&\cdots \\
\mathcal{FC}_d &:= \mathcal{F}_d \leq x_d \mid x_d \leq \mathcal{F}_d \\
\mathcal{BC} &:= \mathcal{FC}_0 \mid \mathcal{FC}_1 \mid \cdots \mid \mathcal{FC}_d. \\
\mathcal{B} &:= \top \mid \bot \mid \text{if } \mathcal{BC} \text{ then } \mathcal{B} \text{ else } \mathcal{B}
\end{aligned}$$

For each feature x_j, we define for it a feature-specific set of constants $M_j = \{m_{j,1} \ldots, m_{j,k_j}\}$. Having feature-specific constants offers two benefits. First, it allows for the comparison of feature values with constants that are relevant for the meaning and distribution of the feature. Second, it restricts the search space more than having one joint set of constants that every feature value is allowed to compare to. As for image data, restricting the search space can have a crucial impact on the synthesis performance.

Different heuristics can be applied for selecting feature-specific constants. One such heuristic is to use quantiles as constants, for example, quartiles that divide the data into four sections, or octiles that divide the data into eight sections. Other summary statistics and domain-specific knowledge are other possible sources of good constants.

5 Experimental Evaluation

We computed mimic programs for both the MNIST [23] and Pima Indians diabetes [37] datasets. They are widely-cited benchmarks for image and socially-consequential tabular data classification tasks respectively. For both, we implemented an opaque classifier as a deep neural network trained using Tensorflow (Keras). We employed a standard 80/20 train-test split. All reported results are averaged values over 10 execution runs.

For mimic program synthesis, we used `cvc5` [6]. The semantic constraints that encode the input-output examples, as well as the grammar, were written in SMT-LIBv2 format and fed to `cvc5` to obtain the corresponding mimic program P_f. As a baseline comparison, we also build mimic programs by training binary decision trees over the same Pts sets as the mimic programs and study them as surrogate models. For such decision trees, we used the off-the-shelf implementation from scikit-learn [33]. We executed all experiments on an AMD Ryzen 9 5900x CPU, with 32GB of RAM and a Nvidia GeForce RTX 3700Ti GPU, running Ubuntu 20.04. Our code for both training the neural network and performing all synthesis experiments is publicly available.[2]

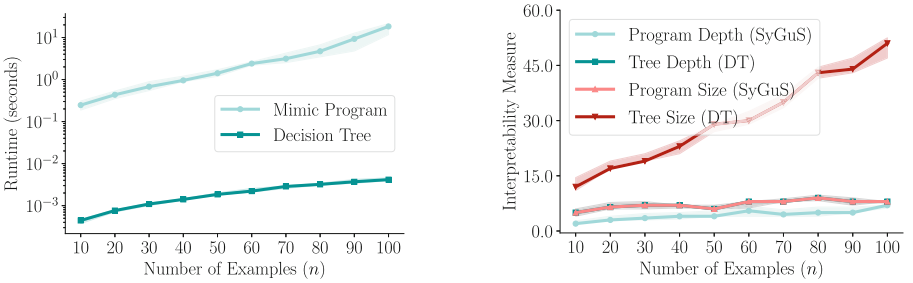
5.1 MNIST Dataset

The MNIST dataset of handwritten digits [23] is a well-known benchmark in image classification problems. Each instance is a vector $x \in \mathbb{R}^{784}$ encoding a 28×28 greyscale image, each representing a digit in $\{0, 1, \ldots, 9\}$. In our case-study, we use the subset of all instances of the digits 1 and 7. This gives us a data set with 15170 instances. For the model f we used a trained deep neural network from [11] with two convolutional and two max-pool layers, followed by a dropout layer and a fully connected layer. With a total of 34k trainable parameters, the network achieves a 99% test accuracy when fully trained.

We computed mimic programs using the grammar defined in Sect. 4.2 for an increasing number of data points and evaluate the resulting programs with respect to synthesis times, interpretability, and global accuracy. As a baseline, we compare our results in runtime, interpretability, and accuracy to the ones obtained by training a binary decision tree on the same set of examples Pts. We use the off-the-shelf decision tree classifier from scikit-learn [33].

Results - Synthesis Times. Figure 2a shows the synthesis time to compute the corresponding mimic programs as a function of $n = |\mathsf{Pts}|$, averaged over 10 runs. Even though each input data has the high dimension of $\mathbb{R}^{784}$, `cvc5` was still able to find mimic programs for relatively large sizes of Pts in a reasonable time. For example, computing a mimic program for $|\mathsf{Pts}| = 100$ took an average 19 seconds. However, the steep growth of the curve highlights the challenge of using SyGuS on high-dimensional image data, even when using very restricted grammars. On the other hand, scikit-learn manages to learn decision trees orders of magnitude faster.

[2] https://github.com/kbjorner/synthesis.



(a) Average time for synthesis. (b) Program size and depth.

Fig. 2. Results on synthesis times and program sizes for MNIST with growing size of Pts, for both mimic programs and decision trees.

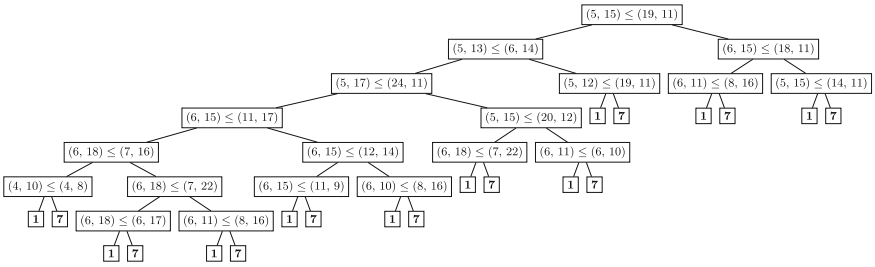
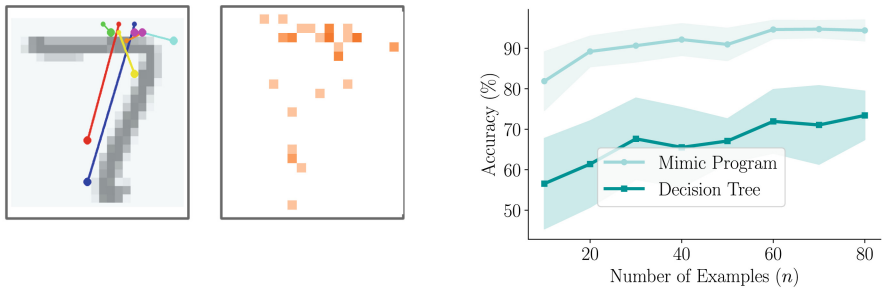


Fig. 3. Tree representation of the decision logic for $P_{f,100}$.

Results - Interpretability. We consider two measures of interpretability: program size and program depth. For the program size, we consider the total number of If-Then-Else statements in the program. The program depth is the maximum depth of nested If-Then-Else statements. In Fig. 2b, we report the average program size and depth of the computed mimic programs for an increasing number of data points $n = |\mathsf{Pts}|$. The results show that the computed mimic programs are quite small and are therefore well suited for humans to analyse the decision-making of f. Our results also show that the mimic programs produce smaller (i.e. more interpretable) surrogate models than the decision trees produced by scikit-learn. On average, the mimic programs only need to compare the values of 13 pixels to mimic the classification of 100 images, while this number goes up to 51 for decision trees. We observed that the depths of the resulting programs are relatively high compared to the size of the program, with no improvement with respect to decision trees. We leave the guidance of the synthesis procedure to obtain programs with smaller depths for future work.

The grammar G_{image} that we use to synthesize mimic programs for image data allows to represent the program as a tree. Figure 3 gives a graphical representation of a mimic program $P_{f,100}$ computed from $|\mathsf{Pts}| = 100$ data points. The resultant tree (of depth = 6) has only 19 branching nodes and compares

the values of 23 pixels. The decision logic of f for a given input data in Pts can be analyzed by following the corresponding path in the tree.



(a) Comparisons in decision logic (left) and heatmap (right).

(b) Global Accuracy of mimic programs and decision trees for growing $n = |\mathsf{Pts}|$.

Fig. 4. Graphical explanation of a mimic program as well as accuracy results.

Figure 4a gives alternative graphical representations of $P_{f,100}$ to facilitate interpretability. Figure 4a (left) illustrates which pixels are compared by $P_{f,100}$ for a given input image when following the left-most path through the tree (the background is an exemplary digit from the MNIST dataset). In the Appendix in Fig. 8, we give the representation of the full tree with a graphical visualisation of the pixel comparisons in every node. Figure 4a (right) gives the heat map of all pixels used in $P_{f,100}$. The darkness of each pixel corresponds to the number of times it appears in a guard within $P_{f,100}$.

The graphical representations in Fig. 4a may show which areas of the image are highly relevant for the classification performed by f. For the classification between 1 and 7, it captures the intuition that the distinguishing features between the two digits are the width of the horizontal stroke as well as the slant and depth of the vertical.

Results - Global Accuracy. We also evaluated how well the trained mimic programs globally approximate the classifications of f for data points outside of Pts. To do so, we build mimic programs $P_{f,n}$ for different numbers of examples n and compare the predictions given by $P_{f,n}$ with the predictions given by f. In Fig. 4b we plot the accuracy of the mimic program for different sizes $n = |\mathsf{Pts}|$. We perform each evaluation by uniformly sampling 10 data points over the entire data set and checking whether the outputs of $P_{f,n}$ and f match, and average the results of 10 runs. Our results show that mimic programs computed from only 40 images already reach an accuracy of about 90%. Therefore, the mimic programs serve as a reasonably good approximation for the classification distinguishing between the 1's and 7's of f. The same experiment for decision trees shows that they are less accurate than our mimic programs.

5.2 Pima Indians Diabetes Dataset

In a second set of experiments, we computed mimic programs for the Pima Indians diabetes dataset [37]. Each data point is composed of a feature vector $x \in \mathbb{R}^8$ encoding medical data, with a binary output that represents a diabetes diagnosis. The entire data set consists of 768 data points. For the model f we used a simple feedforward neural network architecture with three dense layers with ReLU activation, followed by a last sigmoid layer. With a total of 722 trainable parameters, the model achieves an accuracy on the test set of about 78%, which is close to optimal for this dataset.

For the synthesis of mimic programs, we use the grammar $G_{tabular}$ described in Sect. 4.2. We perform experiments with four different sets of feature-specific constants. The first three grammars use statistical measures of the data set to define the feature-specific constants. In particular, we use the following sets:

- *Quartiles.* For each feature, we use the three quartiles ($Q1$, $Q2$, $Q3$) for the feature constants.
- *Sextiles.* For each feature, we use the five sextiles as feature constants.
- *Octiles.* For each feature, we use the seven octiles as feature constants.

Lastly, we also use the constants obtained from a trained decision tree as feature-specific constants for a fourth grammar. Concretely, we train the decision tree over the test fraction of the dataset, and use the values of the tree splits as feature-specific constants. We perform this experiment mainly to study the effects of selecting good constants on the synthesis times and sizes of the resulting mimic programs, under the assumption that state-of-the art methods for training decision trees find good constants for branching conditions. We use the off-the-shelf decision tree classifier from scikit-learn [33] with default parameters to train the decision tree. We refer to the grammar using the constants obtained from the decision tree as the *bootstrapped* grammar.

Finally, as a baseline comparison, we train decision trees with scikit-learn on the same set of examples Pts as the mimic programs, and study their properties as surrogate models in terms of runtime, interpretability, and both global and local accuracy.

Results - Synthesis Times. Figure 5 gives the synthesis times to compute the mimic programs, averaged over 10 runs, for the four grammars and decision trees, over an increasing number of data points (*i.e.*, different sizes of the set Pts). For each run, the data points are sampled uniformly at random from the data set. The experiments show that SyGuS is able to find mimic programs in less than one second for this tabular dataset, with only the *bootstrapped* grammar struggling in some queries with a large number of examples. As with the case of image data, scikit-learn is orders of magnitude faster.

Results - Interpretability. Figure 6 presents the results for the four different grammars and decision trees over an increasing number of data points. We observe that for all grammars, the size of the mimic program grows linearly and the depth

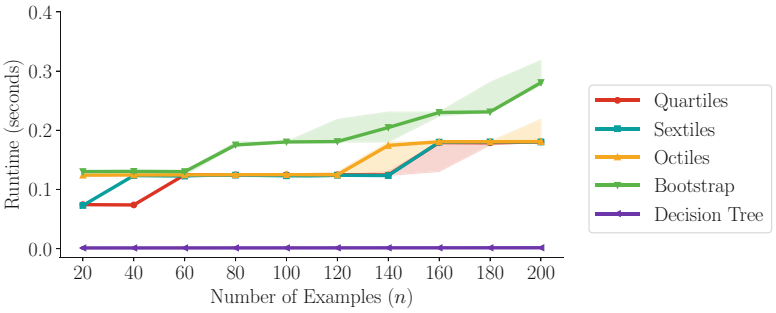
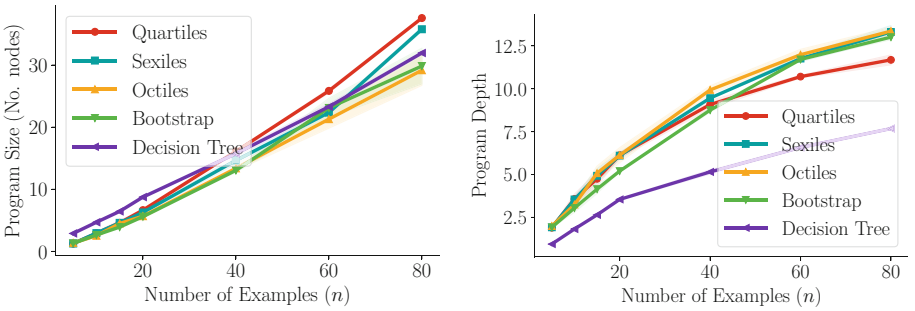


Fig. 5. Synthesis times for Pima dataset.

of the program grows logarithmically. Note that a mimic program restricted by $G_{tabular}$ can also be graphically represented as a tree. Since the depth of the program grows slowly, the mimic program is well suited to analyse individual decisions of the model f. As for the decision trees, in this case we observe that the size of the tree is the same as the size of the mimic programs, while being significantly shallower. Observe that this different size-to-depth ratio between decision trees and mimic programs was already present in the MNIST experiments, with scikit-learn produced more balanced trees than our mimic programs.



(a) Size of Mimic Programs. (b) Tree-depth of Mimic Programs

Fig. 6. Interpretability and runtime for different grammars with growing size.

Results - Global and Local Accuracy. Next, we evaluate how well the trained mimic programs approximate (locally and globally) the decisions of f for data points outside of Pts.

Therefore, we randomly select a data point x^* from the training data set, and select the n closest training data points (with corresponding classification from f) to obtain Pts_{x^*}. From Pts_{x^*}, we compute *local* mimic programs for x^*.

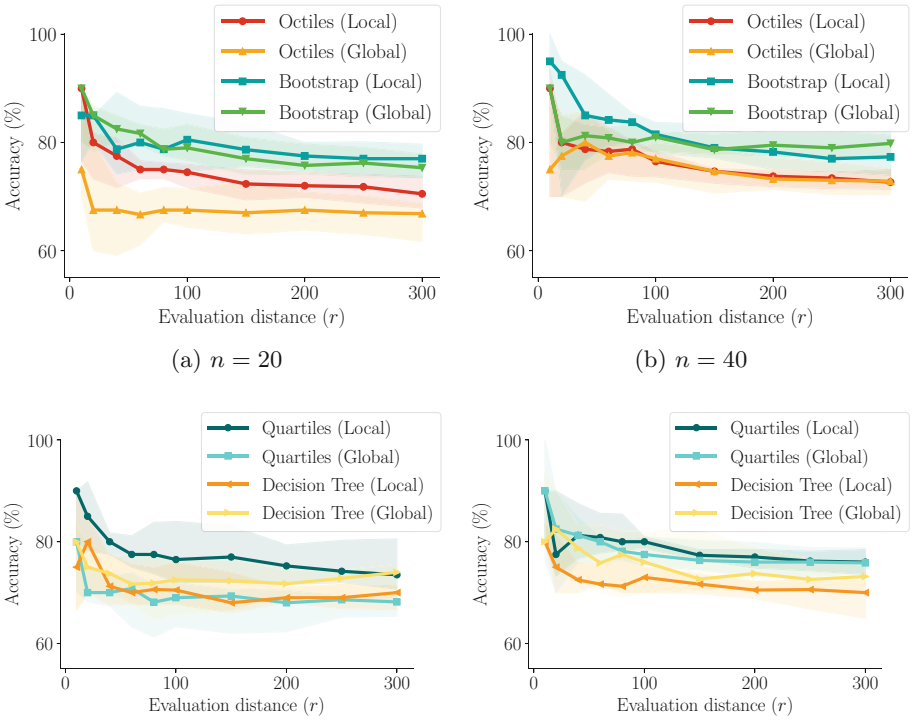


Fig. 7. Global vs. local accuracy evaluation for the *octiles* and *bootstrapped* grammars, as well as for the *quartiles* grammar and for decision trees.

Additionally, we compute *global* mimic programs from n uniformly sampled data points from the training data set.

For the accuracy evaluation, we pick the r closest points from the test data set with their classification from f, and evaluate the accuracy of local and global mimic programs on these data points, as well as local and global decision trees as surrogate models. Figure 7(a) and Fig. 7(b) give the accuracy results for global mimic programs and local mimic programs computed from the $n = 20$ and $n = 40$ closest data points to x^*, respectively. We evaluate the accuracy for an increasing number r of closest points to x^* in the test data set.

In analogous terms, in Fig. 7(c) and Fig. 7(d) we illustrate an accuracy comparison between mimic programs and decision trees. In this case, the mimic programs are built from the *quartiles* grammar. To provide a clear comparison, we have illustrated accuracy results by comparing two approaches side by side. Since the results obtained by grammars using quantiles as feature constants are very similar, we have omitted the results for sextiles.

The results show that for our case study, the obtained mimic programs serve as good approximations of f. As expected, the local mimic programs have a slightly higher accuracy on local test data points then the global mimic programs. With increasing r, the accuracy of the local mimic programs decreases. For higher distances ($r \geq 240$) the global mimic programs have a slightly higher accuracy than the local ones. We also observe that the bootstrapped grammar performs slightly better in accuracy compared to the quantiles-based grammars, and these ones, in turn, perform slightly better than scikit-learn decision trees. However, note that the accuracy is in general high for all test instances.

6 Related Work

Explainable artificial intelligence (XAI) has been receiving significant attention across multiple application domains [2,10]. The ability to explain the decision making of an opaque model has become a standard requirement for the development of trustworthy AI systems to be applied in critical domains. Consequently, an increasing number of XAI methods and tools have been proposed both in industry and academia. We refer to recent surveys that classify and discuss various state-of-the-art XAI techniques [14,28,30].

Formal Methods for XAI. Most existing XAI techniques rely on stochastic methods without any correctness guarantees for the provided explanations. In contrast, there have been several recent works that use formal methods to generate provably correct explanations [7,19,27]. Several of these approaches build upon the verification of deep neural networks (DNN). These approaches typically compute a minimal subset of input features which by themselves already determine the classification produced by the DNN [20,36]. There are also recent approaches using formal methods specifically for explainable reinforcement learning and policies Markov decision processes (MDP) [8,9,39].

Several works exist to make the decision-making of *black-box systems used for control* explainable. Automata learning refers to techniques that infer a surrogate model (e.g., in the form of an input-output automaton [41], a timed automaton [13] or an MDP [38]) from a given black-box system by observing its behavior. The tool dtControl [5] learns decision trees for hybrid and probabilistic control systems, and has been recently extended to support richer algebraic predicates as splitting rules with the use of support vector machines [22]. While not a direct comparison with dtControl, our experimental baseline (scikit-learn) uses the same kind of binary decision trees as surrogate models.

Our approach follows this line of research on formal XAI, studying classification problems for image and tabular data. We rely on *syntax-guided synthesis (SyGuS)* [3] to generate provably correct explanations. In particular, we use SyGuS via formulating the semantic constraints as input-output examples [21,31]. The concept of Programming-by-Example is well known due to the success of the FlashFill [17] feature in Microsoft Excel spreadsheet software. The broad acceptance of FlashFill is due to the fact that it is very simple to use: the user only has to provide the examples. Our approach for computing formal explanations from examples follows this idea. Neider et al. [32] followed a similar direction and proposed to use a combination of probably approximately correct learning (PAC) and syntax-guided synthesis (SyGuS) to produce explanations that with a high probability make only few errors. In contrast to our work, [32] does not compute explanations for image data.

7 Conclusion

In this paper, we synthesize formal explanations for the decisions of opaque machine learning models used for classification tasks. From a given set of data points and the corresponding classification of the opaque model, we compute a mimic program in the form of a quantifier-free first-order logic formula that produces the same output as the given model for all given examples. We formulate the synthesis problem as SyGuS problem and use if-then-else grammars to obtain mimic programs that can be represented as decision trees. For future work, we want to perform more comprehensive case studies on tabular data, since we see the most potential of our method in analysing such data. In particular, we want to investigate the effect of richer grammars on the size of the explanations.

Acknowledgements. This work was supported in part from the European Union's Horizon 2020 research and innovation programme under grant agreement N° 956123 - FOCETA, by the State Government of Styria, Austria - Department Zukunftsfonds Steiermark, by the United States Office of Naval Research (ONR) through a National Defense Science and Engineering (NDSEG) Graduate Fellowship, and by the United States National Science Foundation (NSF) award CCF-2131476. The authors thank Benedikt Maderbacher and William Hallahan for their assistance with SyGuS encodings, and Timos Antonopoulos for his helpful comments on an earlier draft.

Appendix

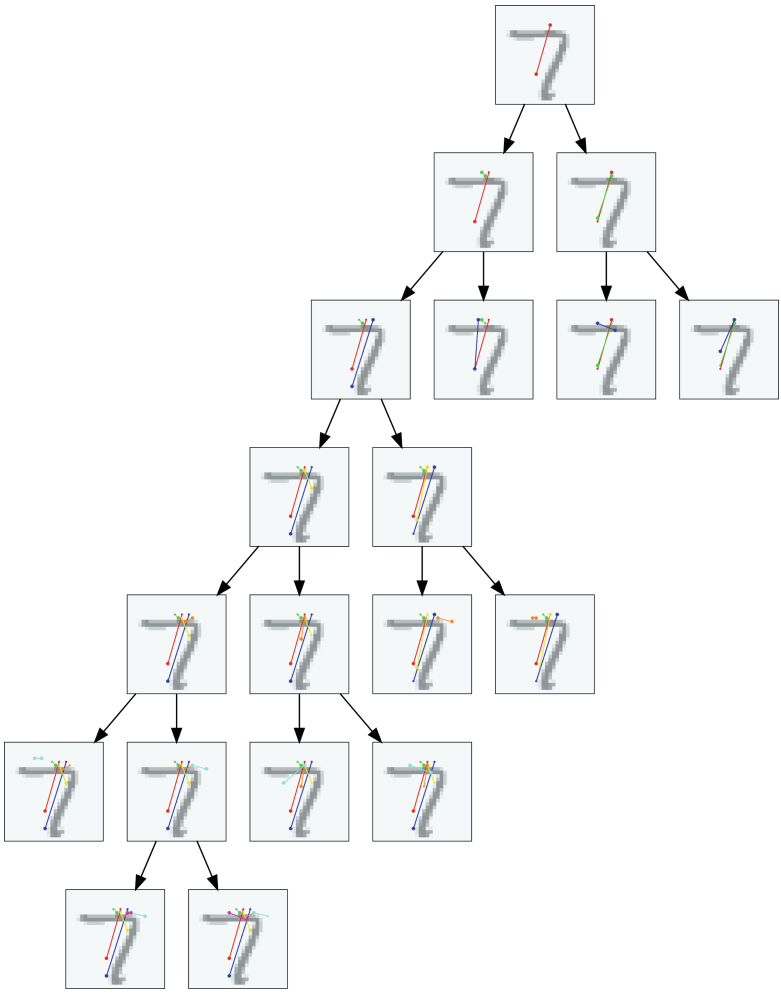


Fig. 8. Decision logic representation for an MNIST mimic program $P_{f,100}$, showing the progression of pixel comparisons.

References

1. Adadi, A., Berrada, M.: Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). IEEE Access **6**, 52138–52160 (2018)

2. Ahmed, I., Jeon, G., Piccialli, F.: From artificial intelligence to explainable artificial intelligence in industry 4.0: a survey on what, how, and where. IEEE Trans. Ind. Inf. **18**(8), 5031–5042 (2022)
3. Alur, R., et al.: Syntax-guided synthesis. In: FMCAD, pp. 1–8. IEEE (2013)
4. Arrieta, A.B., et al.: Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. Inf. Fusion **58**, 82–115 (2020)
5. Ashok, P., Jackermeier, M., Křetínský, J., Weinhuber, C., Weininger, M., Yadav, M.: dtControl 2.0: explainable strategy representation via decision tree learning steered by experts. In: TACAS 2021. LNCS, vol. 12652, pp. 326–345. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-72013-1_17
6. Barbosa, H., et al.: cvc5: a versatile and industrial-strength SMT solver. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2022), pp. 415–442 (2022)
7. Bassan, S., Katz, G.: Towards formal XAI: formally approximate minimal explanations of neural networks. In: TACAS (1). Lecture Notes in Computer Science, vol. 13993, pp. 187–207. Springer, Heidelberg (2023). https://doi.org/10.1007/978-3-031-30823-9_10
8. Cano Córdoba, F., et al.: Analyzing intentional behavior in autonomous agents under uncertainty. In: Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23, pp. 372–381 (2023)
9. Carr, S., Jansen, N., Topcu, U.: Task-aware verifiable rnn-based policies for partially observable markov decision processes. J. Artif. Intell. Res. (JAIR) **72**, 819–847 (2021)
10. Chaddad, A., Peng, J., Xu, J., Bouridane, A.: Survey of explainable AI techniques in healthcare. Sensors **23**(2), 634 (2023)
11. Chollet, F.: Simple MNIST convnet (2015). https://keras.io/examples/vision/mnist_convnet/. Accessed 19 July 2023
12. Costa, V.G., Pedreira, C.E.: Recent advances in decision trees: an updated survey. Artif. Intell. Rev. **56**(5), 4765–4800 (2023)
13. Dierl, S., et al.: Learning symbolic timed models from concrete timed data. In: Rozier, K.Y., Chaudhuri, S. (eds.) NASA Formal Methods, vol. 13903, pp. 104–121. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-33170-1_7
14. Dwivedi, R., et al.: Explainable AI (XAI): core ideas, techniques, and solutions. ACM Comput. Surv. **55**(9), 194:1–194:33 (2023)
15. Fathi, E., Shoja, B.M.: Deep neural networks for natural language processing. In: Handbook of Statistics, vol. 38, pp. 229–316. Elsevier (2018)
16. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., Pedreschi, D.: A survey of methods for explaining black box models. ACM Comput. Surv. (CSUR) **51**(5), 1–42 (2018)
17. Gulwani, S., Harris, W.R., Singh, R.: Spreadsheet data manipulation using examples. Commun. ACM **55**(8), 97–105 (2012)
18. Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., Meger, D.: Deep reinforcement learning that matters. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32 (2018)
19. Ignatiev, A.: Towards trustable explainable AI. In: Bessiere, C. (ed.) Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020, pp. 5154–5158 (2020). https://www.ijcai.org/
20. Izza, Y., Huang, X., Ignatiev, A., Narodytska, N., Cooper, M.C., Marques-Silva, J.: On computing probabilistic abductive explanations. Int. J. Approx. Reason. **159**, 108939 (2023)

21. Jha, S., Gulwani, S., Seshia, S.A., Tiwari, A.: Oracle-guided component-based program synthesis. In: 2010 ACM/IEEE 32nd International Conference on Software Engineering (ICSE 2010), vol. 1, pp. 215–224 (2010)
22. Jüngermann, F., Kretínský, J., Weininger, M.: Algebraically explainable controllers: decision trees and support vector machines join forces. CoRR arXiv:2208.1280 (2022)
23. LeCun, Y., Cortes, C., Burges, C.J.: The MNIST database (1998). http://yann.lecun.com/exdb/mnist. Accessed 13 Aug 2022
24. Li, M., Chan, N., Chandra, V., Muriki, K.: Cluster usage policy enforcement using slurm plugins and an HTTP API. In: Jacobs, G.A., Stewart, C.A. (eds.) PEARC 2020: Practice and Experience in Advanced Research Computing, Portland, OR, USA, 27–31 July 2020, pp. 232–238. ACM (2020)
25. Liang, W., et al.: Advances, challenges and opportunities in creating data for trustworthy AI. Nat. Mach. Intell. **4**(8), 669–677 (2022)
26. Lundberg, S.M., Lee, S.: A unified approach to interpreting model predictions. In: NIPS, pp. 4765–4774 (2017)
27. Marques-Silva, J., Ignatiev, A.: Delivering trustworthy AI through formal XAI. In: AAAI, pp. 12342–12350. AAAI Press (2022)
28. Minh, D., Wang, H.X., Li, Y.F., Nguyen, T.N.: Explainable artificial intelligence: a comprehensive review. Artif. Intell. Rev., 1–66 (2022)
29. Mohsen, H., El-Dahshan, E.S.A., El-Horbaty, E.S.M., Salem, A.B.M.: Classification using deep learning neural networks for brain tumors. Future Comput. Inf. J. **3**(1), 68–71 (2018)
30. Molnar, C.: Interpretable Machine Learning, 2 edn. (2022). https://christophm.github.io/interpretable-ml-book
31. Morton, K., Hallahan, W.T., Shum, E., Piskac, R., Santolucito, M.: Grammar filtering for syntax-guided synthesis. In: AAAI, pp. 1611–1618. AAAI Press (2020)
32. Neider, D., Ghosh, B.: Probably approximately correct explanations of machine learning models via syntax-guided synthesis. arXiv preprint arXiv:2009.08770 (2020)
33. Pedregosa, F., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
34. Ranjbar, N., Safabakhsh, R.: Using decision tree as local interpretable model in autoencoder-based LIME. In: CSICC, pp. 1–7. IEEE (2022)
35. Ribeiro, M.T., Singh, S., Guestrin, C.: "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 (KDD 2016), pp. 1135–1144 (2016)
36. Shih, A., Choi, A., Darwiche, A.: A symbolic approach to explaining bayesian network classifiers. In: Lang, J. (ed.) Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, 13–19 July 2018, pp. 5103–5111 (2018). https://www.ijcai.org/
37. Smith, J.W., Everhart, J.E., Dickson, W., Knowler, W.C., Johannes, R.S.: Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In: Proceedings of the Annual Symposium on Computer Application in Medical Care (1988)
38. Tappler, M., Aichernig, B.K., Bacci, G., Eichlseder, M., Larsen, K.G.: L^*-based learning of markov decision processes (extended version). Formal Aspects Comput. **33**(4–5), 575–615 (2021)

39. Verma, A., Murali, V., Singh, R., Kohli, P., Chaudhuri, S.: Programmatically interpretable reinforcement learning. In: International Conference on Machine Learning (ICML), pp. 5045–5054. PMLR (2018)
40. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: automated decisions and the GDPR. Harvard J. Law Technol. **31**, 841 (2017)
41. Wang, F., Cao, Z., Tan, L., Zong, H.: Survey on learning-based formal methods: taxonomy, applications and possible future directions. IEEE Access **8**, 108561–108578 (2020)