Efficiently Transforming Neural Networks into Decision Trees: A Path to Ground Truth Explanations with RENTT

HELENA MONKE*, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Germany, and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Germany
BENJAMIN FRESZ*, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Germany, and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Germany
MARCO BERNREUTHER, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Germany
YILIN CHEN, Technical University Munich, Germany and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Germany
MARCO F. HUBER, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Germany and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Germany

Background: Although neural networks are a powerful tool, their widespread use is hindered by their black-box nature. Methods in the field of explainable AI (XAI) try to solve this problem by explaining entire models or single decisions. But they are plagued by missing faithfulness/correctness, meaning that their explanations do not always conform to the model in question. Recently, methods to transform neural networks into decision trees have been proposed to overcome such problems. Unfortunately, they typically lack exactness, scalability, or interpretability as the size of the neural network grows.
Objectives: We provide a way to efficiently compute ground truth explanations for neural networks via a theoretically founded and practically implemented transformation from neural networks into multivariate decision trees, called Runtime Efficient Network to Tree Transformation (RENTT). These trees can - when simplified via feature importance explanations - serve as ground truth explanations which perfectly conform to the neural network in question.
Methods: We generalize previous results, especially by considering convolutional neural networks, recurrent neural networks, non-rectified linear unit activation functions, and bias terms. Our findings are accompanied by rigorous proofs and we present a novel algorithm RENTT designed to compute an exact equivalent decision tree representation of neural networks in a runtime and memory efficient manner. The resulting decision trees are multivariate and thus possibly too complex to understand. To alleviate this problem, we also provide a method to calculate the ground truth feature importance for neural networks via the equivalent decision trees—for entire models (global), specific input regions (regional), or single decisions (local). We compare the runtime and memory consumption of our transformation to an existing implementation from Nguyen et al. (2020) for rectified linear unit (ReLU)-based classification networks and the results of our method for regression and classification tasks to common feature importance methods (local: LIME, SHAP, BreakDown; global: Shapley Effects, SAGE). All code is publicly available via GitHub.

*These authors contributed equally to this work.

Authors' Contact Information: Helena Monke, helena.monke@ipa.fraunhofer.de, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany, and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Stuttgart, Germany; Benjamin Fresz, benjamin.fresz@ipa.fraunhofer.de, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany, and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Stuttgart, Germany; Marco Bernreuther, marco.bernreuther@ipa.fraunhofer.de, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany; Yilin Chen, yilin.chen@tum.de, Technical University Munich, Munich, Germany and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Stuttgart, Germany; Marco F. Huber, marco.huber@ieee.org, Fraunhofer Institute for Manufacturing Engineering and Automation IPA, Stuttgart, Germany and Institute of Industrial Manufacturing and Management IFF, University of Stuttgart, Stuttgart, Germany.

Results: The experimental results emphasize the computational efficiency and scalability of our algorithm, which directly stem from our theoretical insights. For more complex networks and data, e.g., image tasks, scalability issues may still arise. Common feature importance methods produce different explanations than RENTT, while only RENTT conforms to expectation for cases with known ground truth explanations. These differences increase in more complex cases.
Conclusions: Further developing the implementation shows significant potential for practically achievable ground truth explainability for complex neural networks. Post-hoc explanation methods should be treated with care, as they fail to reproduce known ground truth explanations.

1 Introduction

Neural networks (NNs) have gained significant importance in the field of machine learning (ML), offering high-performing solutions across various domains like image recognition, natural language processing, and more (Pinecone, 2023). Despite their success, the inherent opacity of NNs poses a significant challenge, limiting their broader adoption in sensible or safety-related application. The lack of interpretability and the "black-box" nature of NNs often lead to skepticism regarding their trustworthiness and internal decision-making processes, as users cannot understand and thus, trust, the rationale behind the outputs. This problem is particularly acute in fields such as healthcare, finance, and legal systems, where the consequences of decisions can be profound (Dzindolet et al., 2003; Rudin, 2019).

To address these concerns, the field of eXplainable Artificial Intelligence (XAI) aims developing techniques that explain and clarify the decision-making processes of ML. The goal of XAI is to make the outputs of ML models more understandable to humans, which can help in validating the fairness and reliability of the ML model's decisions.

In general, XAI methods try to provide insights into the reasoning behind a model's behavior, which could be used to show biases or other flaws in models or to calibrate user trust (Ma et al., 2023; Zhang et al., 2020). Despite ongoing efforts, many current XAI methods struggle with providing explanations that accurately reflect the true reasoning of the NNs (Adebayo et al., 2018; Kindermans et al., 2017). Common methods to provide explanations for given "black-box" models are either local ones like LIME (Local interpretable model-agnostic explanations) (Ribeiro et al., 2016), SHAP (SHapley Additive exPlanations) (Lundberg and Lee, 2017), and gradient based methods or global methods for example SE (Shapley Effects) and SAGE (Shapley Additive Global importancE).

For enhancing interpretability—the general intelligibility of an ML model—a common approach is training an interpretable surrogate model like a decision tree (DT). DTs, if small enough, are more interpretable because their structure allows for the tracing of decision paths, providing clear and logical reasoning for each decision. Instead of training a surrogate model, new approaches aim to transform NNs into equivalent DTs (Aytekin, 2022; Nguyen et al., 2020). Nonetheless, the transformation from NNs to DTs is fraught with challenges, such as providing a correct derivation of the transformation also for advanced network architectures such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs) with general activation functions and not just fully connected neural networks (FcNNs) with Rectified Linear Unit (ReLU) activation. Additionally, an efficient implementation of the transformation, which also works for large networks, is missing. As the size of the decision tree grows, it becomes increasingly difficult to maintain its interpretability due to the complex branching and depth of the tree. To accurately represent the NNs, the corresponding DTs need to be multivariate, i.e., use multiple features per decision node. As such, direct interpretation of these DTs is challenging. Nevertheless, they provide a foundation for deriving more intuitive explanations, in our case so-called feature importance.

To alleviate the various limitations of existing NN to DT transformation methods, we make the following contributions to the field of XAI methods:

- A clear and comprehensive derivation of the theory of transforming NNs to multivariate DTs, covering traditional networks and more complex architectures such as CNNs and RNNs, with general (including non-ReLU-like) activation functions and bias terms, as well as networks for regression and classification tasks.
- A novel algorithm for efficiently computing an exact equivalent DT representation of an NN, called Runtime Efficient Network to Tree Transformation (RENTT), enhancing computational efficiency in terms of memory consumption and runtime.
- A theoretical framework and implementation for extracting the ground truth feature importance from NNs, called RENTT-Feature Importance (RENTT-FI), at global (entire dataset), regional (specific input regions), and local (individual decisions) levels, providing comprehensive insights into model decision-making and identifying key output-driving features.

We begin by contextualizing our work within the existing literature in Section 2. Section 3 introduces the theoretical foundation for transforming NNs—including FcNNs, CNNs, and RNNs—into DTs, while Section 4 presents our RENTT transformation algorithm. We validate our approach through comprehensive numerical experiments that demonstrate the effectiveness of the proposed transformation. In Section 5, we introduce RENTT-FI, a feature importance (FI) calculation method based on the transformed DT. Section 6 presents experimental validation of this approach, assessing both the correctness and utility of our FI calculations, as well as evaluating the computational efficiency of our implementation. By providing a more accurate and scalable approach to XAI, we aim to bridge the gap between the capabilities of NNs and the need for their decisions to be interpretable and trustworthy. Section 7 discusses our results and their limitations, while outlining future research directions aimed at overcoming these identified challenges. Finally, Section 8 provides our concluding remarks.

2 Related Works

The need for better understanding the inner workings of NNs has led to a plethora of innovative approaches. In this section, we will explore how these methodologies aim to provide a clearer understanding of NNs' internal mechanisms and decision processes. Traditional methods for interpreting NNs often lack guarantees of exactness and consistency, leading to potential misinterpretations. Some approaches address this by providing an equivalent description of an NN. It has been shown that NNs can be exactly written as compositions of max-affine spline operators (MASOs) (Balestriero and Baraniuk, 2020), which also enable the approximation of non-piecewise linear activation functions with nonlinearities. This representation simplifies the understanding of their internal mechanics and decision-making processes. By establishing connections to approximation theory, the authors highlight how the structure of deep NN influences their own behavior and performance. In Wang et al. (2019) this is extended to a closer focus on RNNs and their representation as simple piecewise affine spline operators. This representation reveals how RNNs partition the input space over time and suggests that the affine slope parameters function as input-specific templates, enhancing interpretability through a matched filtering perspective. In Sudjianto et al. (2020) ReLU networks are formulated as local linear models, where each activation pattern of the NN results in a local linear model. The activation regions can then be visualized for better interpretability or used to calculate the joint importance of the feature. Chu et al. (2018) proposes a closed-form solution, specifically designed for piecewise linear NNs, where the network is transformed into a mathematically equivalent set of local linear classifiers. This transformation allows exact interpretations of the model's decision boundary by identifying the features that dominate predictions for each local linear classifier.

MASOs or local linear models can also be used to represent an NN as a DT, as shown in Aytekin (2022) and Nguyen et al. (2020). This claim aims to tackle the interpretability issues associated with NNs. In Nguyen et al. (2020), the authors provide two algorithms, one for an exact and one for an approximate but less complex transformation of fully connected ReLU networks used for binary or multiclass classification tasks into multivariate DTs. Their method aims to extract interpretable decision rules and preserves the decision boundaries learned by the network. Similarly, Aytekin (2022) emphasizes the transformation of NNs into DTs, with the goal that any NN can be represented as a DT. The research covers various types of NNs, including feedforward NNs, single-layer RNNs, and CNNs, focusing on regression and classification tasks. Together, these studies highlight a concerted effort in the research community to enhance the interpretability of NNs, providing insights and methodologies that facilitate a deeper understanding of these complex models. The integration of spline representations, DT transformations, and closed-form solutions paves the way for future advancements in the interpretability of NNs, fostering trust and reliability in ML models.

However, these findings are either lacking practical usage and implementation for a transformation or have potential gaps in mathematical rigor, for example by excluding considerations for activation function offsets or bias terms in the transformation process. This especially leaves open questions regarding the implementation. All in all, there exists no efficient and scalable algorithm for transforming an FcNN, RNN or CNN with any picewise linear activation function into an interpretable model, while also providing a thorough mathematical proof of its validity. This makes the usage of existing transformations questionable when it comes to efficient implementation in real-world scenarios, and further exploration is needed to validate these claims, especially in the context of more complex architectures and practical implementations.

Another approach to give post-hoc explanations are FI methods such as LIME (Ribeiro et al., 2016) and SHAP (Lundberg and Lee, 2017). Despite their popularity, these methods are not without criticism. (Molnar et al., 2022) discuss general pitfalls in applying these methods, such as incorrect context usage, poor generalization, and ignoring feature dependencies. They propose solutions for correct interpretation and highlight areas that need further research. I. Covert et al. (2021) propose a unified framework for model explanation using removal-based methods, including SHAP and LIME. They criticize the complexity and variability in existing methods, suggesting a theoretical foundation for improving research in explainability. Similarly, Schröder et al. (2023a) criticize saliency methods like SHAP and LIME for failing to capture latent FI in time series data, especially in medical domains. They recommend redesigning certain methods to improve latent saliency scoring. Further extending this critique, Schröder et al. (2023b) emphasize the need for improved latent feature detection in time series classification, identifying the limitations of current saliency methods. In the domain of network intrusion detection, Tritscher et al. (2023) criticize SHAP for its challenges related to anomaly importance and the impact of replacement data, underscoring the need for rigorous evaluations. Sundararajan and Najmi (2020) and Kumar et al. (2020) both criticize Shapley values used in model explanations for their non-intuitive attributions and mathematical complexity, respectively. They argue that these methods require causal reasoning for accurate interpretations. (Merrick and Taly, 2020) point out substantial attributions to non-influential features and propose a framework for generating explanations that include confidence intervals and contrastive explanations. In the field of natural language processing (NLP), Mosca et al. (2022) review SHAP-based methods, criticizing their adaptations and core assumptions, while Broeck et al. (2022) address the computational complexity challenges of SHAP explanations, highlighting intractability in common settings. Mangalathu et al. (2020) employ SHAP for seismic damage assessment but criticize existing strategies for not adequately identifying failure modes, suggesting the need for improved methods. Slack et al. (2020) demonstrate vulnerabilities in LIME and SHAP, showing how they can be manipulated to produce misleading explanations, and Kim et al. (2022) propose a benchmarking framework to evaluate saliency methods, calling for improvements based on ground-truth model reasoning. These studies collectively provide a critical view of SHAP and LIME, emphasizing their limitations and suggesting areas for improvement and future research in model explainability for FI.

One fundamental issue of the FI methods above is the difficulty in establishing commonly agreed properties and related evaluation metrics for XAI. Here, Nauta et al. (2023) provide a list of 12 so-called "Co-Properties" and corresponding evaluation metrics, often more than one metric per property. They note that some of the properties are in conflict with each other, e.g., Completeness and Compactness of explanations. Two properties are especially relevant in the context of this work: Correctness, which denotes whether an explanation agrees with the model it is created for, and Consistency, which denotes whether an XAI method is deterministic and implementation-invariant, resulting in the same results for the same data sample and model. For Correctness, it has been observed that various evaluation metrics do not consistently agree, leading to conflicting results regarding which XAI method is accurate. Consequently, this inconsistency makes it difficult to obtain easily interpretable or reliable results (Tomsett et al., 2019). For Consistency and especially implementation-invariance, different tests were proposed (Adebayo et al., 2018) but also criticized (Yona and Greenfeld, 2021), resulting in the conclusion "that saliency map evaluation beyond ad-hoc visual examination remains a fundamental challenge". Our Runtime Efficient Network to Tree Transformation (RENTT) and its FI remedy this in two ways: RENTT produces two functionally equivalent models, the NN and the DT, which can be used to test whether FI methods provide the same FI for both, i.e., whether the FI methods are implementation-invariant. Additionally, RENTT-FI provides explanations that are correct by design—as well as deterministic for a given NN—as they are directly based on the NN weights, allowing to compare the other FI methods to "ground-truth" explanations. Note that "importance" can be interpreted differently, thus such a comparison may also be skewed towards one's favored interpretation of this concept (Merrick and Taly, 2020; Sundararajan and Najmi, 2020). For further details see Chapters 5 and 6.

3 Transformation Theory

In this section we provide a rigorous, step-by-step derivation of the underlying transformation theory. It describes how to transform an NN into an equivalent DT.

3.1 Linearizing Fully Connected Neural Networks

Let us start by introducing the notation for formulating an FcNN.

Definition 1 (Fully Connected Neural Networks)**.** *We assume that the network has an input layer, indexed with 0, $N-1 \in \mathbb{N}$ hidden layers, indexed with $i = 1, \dots, N-1$, and an output layer, indexed with N. The dimension of each layer is $n_i \in \mathbb{N}$ for $i = 0, \dots, N$ with neurons indexed with $j = 0, \dots, n_i - 1$. The input to the network is $\mathbf{x}_0$, and the output is $\mathbf{x}_N$. For each layer i, the input is $\mathbf{x}_{i-1}$ and the output is $\mathbf{x}_i$. The set of all input feature vectors is denoted as $\mathcal{X}_0 \subset \mathbb{R}^{n_0}$. Then, we define a FcNN as*

$$M^N : \mathcal{X}_0 \to \mathbb{R}^{n_N} , \quad \mathbf{x}_0 \mapsto M^N(\mathbf{x}_0) \coloneqq M_N^N \circ \dots \circ M_1^N(\mathbf{x}_0) ,$$

where each mapping M_i^N is defined via

$$M_i^N : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i} , \quad \mathbf{x}_{i-1} \mapsto \mathbf{x}_i \coloneqq \sigma_i(\mathbf{W}_{i-1}\mathbf{x}_{i-1}) , \quad i = 1, \dots, N . \tag{1}$$

Here, $\mathbf{W}_{i-1} \in \mathbb{R}^{n_i \times n_{i-1}}$ is the weight matrix, and $\sigma_i : \mathbb{R}^{n_i} \to \mathbb{R}^{n_i}$ is the activation function for the i-th layer.

Remark 1 (Bias). *To simplify the notation, we assume that the biases are already incorporated into the weight matrices, i.e., there exists an original input feature vector $\tilde{\mathbf{x}}_0 \in \mathbb{R}^{n_0-1}$, original weight matrices $\tilde{\mathbf{W}}_i \in \mathbb{R}^{(n_{i+1}-1)\times(n_i-1)}$, bias vectors $\mathbf{b}_i \in \mathbb{R}^{n_i-1}$ and original activation functions $\tilde{\sigma}_i : \mathbb{R}^{n_i-1} \to \mathbb{R}^{n_i-1}$ such that*

$$\mathbf{W}_i \coloneqq \begin{pmatrix} 1 & 0 \\ \mathbf{b}_i & \tilde{\mathbf{W}}_i \end{pmatrix} \in \mathbb{R}^{n_{i+1}\times n_i} , \quad \mathbf{x}_0 \coloneqq (1, \tilde{\mathbf{x}}_0)^\top \in \mathbb{R}^{n_i} ,$$
$$\sigma_i : \mathbb{R} \times \mathbb{R}^{n_i-1} \to \mathbb{R} \times \mathbb{R}^{n_i-1} , \quad \sigma_i = (\mathrm{id}_{\mathbb{R}}, \tilde{\sigma}_i)^\top .$$

The weight matrices $\mathbf{W}_i$ thus include the bias and additional constant dummy entries of 0 and 1. Similarly, the input feature $\mathbf{x}_0$ has one dummy entry of 1. This setup ensures that the dummy input feature propagates through the network and is used to incorporate the bias. This bias entry is represented in the NN for example as shown in Figure 1.

In the following, we show how to linearize the FcNN, which will then be used in the next section to build a DT. For that, some assumptions are necessary.

Assumption 1 (Activation Function)**.** *We assume, that*

(1) the activation functions σ_i are Lipschitz continuous,
(2) the activation functions σ_i act component-wise, where for a given layer, all component-wise activation functions are the same, and
(3) per layer and component, the activation functions $\tilde{\sigma}_i$ are piecewise linearly defined within $r_i \in \mathbb{N}$ activation regions that partition $\mathbb{R}$. In a monotonically increasing order, the boundary points of all linear regions, i.e., all intervals, are $\mathbf{r}_i \in \mathbb{R}^{r_i - 1}$.

From Assumption 1.1, it follows by Rademacher's theorem, see for instance (Evans, 2018), that σ_i is differentiable almost everywhere. Therefore, we will assume, without loss of generality, that the activation functions are differentiable at the points of interest.

Common activation functions such as ReLU, LeakyReLU, or parametric ReLU satisfy Assumption 1.

Remark 2 (Non-piecewise-linear Activation Function). *If the activation function is not inherently piecewise linear, it can be approximated as such, allowing the following derivation to serve as an approximation rather than in an exact manner.*

However, this approximating approach may result in a significant increase in the number of linear activation regions. Consequently, this would not be feasible for the transformation into a DT. Therefore, the linearization of the FcNN excludes the last layer's activation function, which typically lacks Assumption 1.3 (e.g., tanh or sigmoid function). Instead, this activation function is applied to the resulting linear representation.

For layer $i < N$, we formulate the mapping M_i^N acting between layers $i-1$ and i of the FcNN as

$$\begin{aligned} M_i^N(\mathbf{x}_{i-1}) &= \sigma_i(\mathbf{W}_{i-1}\mathbf{x}_{i-1}) \\ &= \underbrace{\sigma_i'(\mathbf{W}_{i-1}\mathbf{x}_{i-1})}_{=\mathbf{Z}_i \in \mathbb{R}^{n_i \times n_i}} \mathbf{W}_{i-1}\mathbf{x}_{i-1} + \underbrace{\sigma_i(\mathbf{W}_{i-1}\mathbf{x}_{i-1}) - \sigma_i'(\mathbf{W}_{i-1}\mathbf{x}_{i-1})\mathbf{W}_{i-1}\mathbf{x}_{i-1}}_{=\mathbf{\Delta}_i \in \mathbb{R}^{n_i}} \\ &= \mathbf{A}_i \mathbf{W}_{i-1}\mathbf{x}_{i-1}\,, \end{aligned} \tag{2}$$

where σ_i' is the derivative of σ_i. This reformulation is a linearization of the FcNN mapping. The linear slope of activation $\mathbf{Z}_i$ and the offset $\mathbf{\Delta}_i$ for each activation region in which $\mathbf{W}_{i-1}x_{i-1}$ lays, are represented jointly by the activation matrix $\mathbf{A}_i \in \mathbb{R}^{n_i \times n_i}$, which is defined by

$$(\mathbf{A}_i)_{kl} = \begin{cases} (\mathbf{Z}_i)_{kl} + (\mathbf{\Delta}_i)_k, & l = 1\,, \\ (\mathbf{Z}_i)_{kl}, & l > 1\,. \end{cases} \tag{3}$$

It exploits the block-matrix form of $\mathbf{W}_{i-1}$ and the fact that the first entry of $\mathbf{x}_i$ is always 1, so the leftmost column of the activation matrix includes the offset $\mathbf{\Delta}_i$. Due to Assumption 1.2 this linearization is exact.

Example 1 (Activation Function). *For clarity, let us state two special cases:*

(1) In the one-dimensional case, $\mathbf{Z}_i \in \mathbb{R}$ represents the slope, and $\mathbf{\Delta}_i \in \mathbb{R}$ denotes the y-intercept.
(2) In case of a ReLU activation functions $\sigma(x) = \max(0, x)$, we obtain a diagonal matrix $\mathbf{Z}_i$ with $(\mathbf{Z}_i)_{jj} \in \{0, 1\}$ and a vector $\mathbf{\Delta}_i = 0$.

Intuitively, the activation matrix $\mathbf{A}_i$ contains the slope and intercept of the piecewise linear activation functions to each linear piece for each neuron at layer i. In a ReLU network, as in Example 1.2, it contains either slope 1 or slope 0 for each neuron.

Overall, the FcNN can be reformulated as

$$\begin{aligned} M^N(\mathbf{x}_0) &= \sigma_N\left(\mathbf{W}_{N-1}(\mathbf{A}_{N-1}\dots\mathbf{W}_1(\mathbf{A}_1(\mathbf{W}_0\mathbf{x}_0)))\right) \\ &= \sigma_N\left(\prod_{i=1}^{N-1}(\mathbf{W}_i\mathbf{A}_i)\mathbf{W}_0\mathbf{x}_0\right) . \end{aligned}$$

Remark 3 (Product Operator). *Throughout this paper the product operator is defined to add higher indexed elements to the left, contrary to the conventional right-sided application.*

To further simplify the notation in the following, let us introduce the iteratively defined effective weight matrices similar to (Aytekin, 2022)

$$\begin{aligned} \mathbf{D}_0 &\coloneqq \mathbf{W}_0 \quad \in \mathbb{R}^{n_1\times n_0} , \\ \mathbf{D}_i &\coloneqq (\mathbf{W}_i\mathbf{A}_i)\mathbf{D}_{i-1} \quad \in \mathbb{R}^{n_{i+1}\times n_0}, \quad i = 1,\dots,N-1 , \end{aligned} \tag{4}$$

which allow to formulate the application of the FcNN as

$$M^N(\mathbf{x}_0) = \sigma_N(\mathbf{D}_{N-1}\mathbf{x}_0) . \tag{5}$$

The activation matrix $\mathbf{A}_i$ depends on $\mathbf{x}_{i-1}$ and $\sigma_i(\mathbf{W}_{i-1}\mathbf{x}_{i-1})$ and can only be calculated once $\mathbf{x}_{i-1}$ is known. But due to the linearization, each activation region possesses an explicit formula to calculate the corresponding output, as we will explain in detail in the following. As a start, we now introduce the activation pattern of sample $\mathbf{x}_0$.

Definition 2 (Activation Pattern)**.** *For an FcNN as defined in Definition 1 the i-th partial activation pattern is defined as the following mapping*

$$\mathbf{P}_i\colon \mathbb{R}^{n_0} \to \bigtimes_{\tilde{i}=1}^{i} \{0,\dots,r_{\tilde{i}}\}^{n_{\tilde{i}}} ,$$

where $\bigtimes$ denotes the Cartesian product of sets and $r_{\tilde{i}}$ is the number of the activation regions, which are concatenated for each neuron in the layer. $\mathbf{P}_i(\mathbf{x}_0)$ represents the activation states of each neuron in the network up to layer i for input $\mathbf{x}_0 \in \mathbb{R}^{n_0}$. Furthermore, the notation $\mathbf{P}(\mathbf{x}_0) = \mathbf{P}_{N-1}(\mathbf{x}_0)$ is used and simply referred to as activation pattern. The number of different activation patterns $|\mathbf{P}(\mathcal{X}_0)|$ is determined by all possible combinations of the activation region for the neurons.

This definition of activation patterns is adopted from (Raghu et al., 2017), to which we refer for further details. For a better intuition of the activation pattern, an example is given in Section 3.3. Intuitively, the activation pattern captures which piece of the piecewise linear activation function is active for each neuron at layer i for a given input. In a ReLU network, as in Example 1.2, it encodes which neurons are "firing" (slope 1) versus "not firing" (slope 0). Inputs with identical activation patterns traverse the same sequence of linear regions and are therefore processed identically by the network, they follow the same computational path and have identical activation matrices $\mathbf{A}_i$. This grouping allows us to partition the input space into regions where the network behaves as the same affine function. Each input vector $\mathbf{x}_0$ determines a family of activation matrices $(\mathbf{A}_i)_{i=1}^{N-1}$ and a family of partial activation patterns $(\mathbf{P}_i)_{i=1}^{N-1}$. Therefore, the input features $\mathbf{x}_0$ can be grouped into $|\mathbf{P}_i(\mathcal{X}_0)|$ regions, each having a unique activation pattern ${}_m\mathbf{P}_i$ with $m = 1,\dots,|\mathbf{P}_i(\mathcal{X}_0)|$. Consequently, we introduce the notation ${}_m\mathbf{D}_i$

and ${}_m\mathbf{A}_i$ to denote that these matrices are uniquely associated to the partial activation pattern ${}_m\mathbf{P}_i$. The number of activation patterns $|\mathbf{P}_i(\mathcal{X}_0)|$ scales exponentially with the number of neurons and layers respectively. For $k < i$, the activation matrices ${}_m\mathbf{A}_k$ can be explicitly determined based on a given partial activation pattern ${}_m\mathbf{P}_k$ by knowing the activated region of the activation function based on the activation patter as shown in Equation (3). In ReLU Example 1.2, ${}_m\mathbf{A}_k$ consists of either 1 or 0 at the diagonal depending on whether the activation pattern is 1 or 0. This explicit calculation allows specifying Equation (5) via using Equation (4) with the now known matrices ${}_m\mathbf{A}_k$ as follows

$$ {}_mM^N(\mathbf{x}_0) = \sigma_N\left({}_m\mathbf{D}_{N-1}\mathbf{x}_0\right) , \tag{6} $$

with m indicating the input region corresponding to activation pattern ${}_m\mathbf{P}$. This corresponds to a linear description of the FcNN for each activation pattern ${}_m\mathbf{P}$, so that

$$ M^N(\mathbf{x}_0) = {}_mM^N(\mathbf{x}_0) \quad \Leftrightarrow \quad \mathbf{P}(\mathbf{x}_0) = {}_m\mathbf{P} . $$

3.2 Decision Tree Representation

This FcNN can be represented as a fixed decision tree (DT), where the activation pattern will provide a unique index for each path in the DT, as we will argue in detail in the following.

The l_{ij}-th level of the DT is characterised by the unique neuron $j = 1, \ldots, n_i$ in layer $i = 1, \ldots, N-1$, such that

$$ l_{ij} = j + \sum_{\tilde{i}=1}^{i-1} (n_{\tilde{i}} - 1) . \tag{7} $$

Each level in the DT corresponds to one neuron in the FcNN, with the DT having $\sum_{i=1}^{N}(n_i - 1)$ levels in total. Here, $n_i - 1$ accounts for the number of neurons excluding the dummy bias neuron. The number of nodes e_{ij} at level l_{ij} is given by

$$ e_{ij} = r_i^j \prod_{\tilde{i}=1}^{i-1} r_{\tilde{i}}^{n_{\tilde{i}}} , \tag{8} $$

where r_i denotes the number of linear regions of the activation function in layer i (as specified in Assumption 1.3). This formula indicates that the number of nodes grows exponentially with the number of neurons and layers respectively. For layer i and neuron j, the decision rule at level l_{ij} is given by

$$ \mathbf{1} \cdot \sum_{k=1}^{n_0} ({}_m\mathbf{D}_{i-1})_{jk}(\mathbf{x}_0)_k > \mathbf{r}_i \quad \in \mathbb{R}^{r_i - 1} , \tag{9} $$

where $\mathbf{1} \in \mathbb{R}^{r_i - 1}$ denotes a vector of all ones. The decision rule involves multiplying the input features $\mathbf{x}_0$ by the j-th column of the effective matrix $({}_m\mathbf{D}_i)_j$, which incorporates the weights and activations of neurons from previous layers, and then sums up for all features. These values are compared in a piecewise manner with the current activation function and thus, are separated according to the activation regions. The output is interpreted as a boolean vector, with each element representing a partial activation pattern. It corresponds to a split at a particular level of the DT, thereby making a decision of which activation region is used. Recall that index m corresponds to the activation region associated to the different activation pattern ${}_m\mathbf{P}$ and thus, also to the partial activation pattern $\mathbf{P}_{i-1}$. Therefore, each index m provides a unique path in the DT. And so, each unique branch determines one unique activation pattern, denoted as ${}_m\mathbf{P}_i$. Note that this decision rule is slightly more complex than for univariate DT: Instead of using only one feature per decision node, multiple features can be used. As such, the resulting DT is multivariate.

Remark 4 (Multi-Way Tree). *For $r_i = 2$, the tree is binary and for $r_i > 2$, it becomes a multi-way tree, where nodes split into more than two branches.*

Each leaf node in the DT gives the linear equation for the NN mapping defined in Equation (6) for the corresponding unique activation pattern ${}_m\mathbf{P}$ without the last activation function. This often non-piecewise activation function can be applied afterwards because it is not feasible for the transformation into a DT as stated in Remark 2.

3.3 Example: Transformation of Fully Connected Neural Network into Decision Tree

To illustrate the transformation of an FcNN into a DT, consider the following example. The network consists of:

- Input Layer: Size $\tilde{n}_0$ with ReLU activation.
- Hidden Layers: first layer $\tilde{n}_1 = 2$, second layer $\tilde{n}_2 = 3$, with ReLU activation each.
- Output Layer: Size $\tilde{n}_3 = 1$, with tanh activation function.

The dimension of each layer n_i is incremented by 1 to account for the bias term. This network is visualized in Figure 1. The network's mapping is given by Equation (1), the activation matrix for ReLU activation is given by Equation (3) and detailed in Example 1.2 in Section 3.1. Using the effective matrices

$$\begin{aligned}\mathbf{D}_0 &\coloneqq \mathbf{W}_0 \in \mathbb{R}^{3\times n_0}\,,\\ \mathbf{D}_1 &\coloneqq (\mathbf{W}_1\mathbf{A}_1)\mathbf{D}_0 \in \mathbb{R}^{4\times n_0}\,,\\ \mathbf{D}_2 &\coloneqq (\mathbf{W}_2\mathbf{A}_2)\mathbf{D}_1 \in \mathbb{R}^{1\times n_0}\,,\end{aligned}$$

and Equation (6), the FcNN can be reformulated as

$${}_mM^N(\mathbf{x}_0) = \tanh({}_m\mathbf{D}_2\mathbf{x}_0).$$

For each neuron j in layer i, the level l_{ij} in the DT is given by

$$l_{ij} = j + \sum_{\tilde{i}=1}^{i-1}(n_{\tilde{i}} - 1)\,, \qquad j = \begin{cases} 1,2 & \text{if } i = 1\,,\\ 1,2,3 & \text{if } i = 2\,,\\ 1 & \text{if } i = 3\,.\end{cases}$$

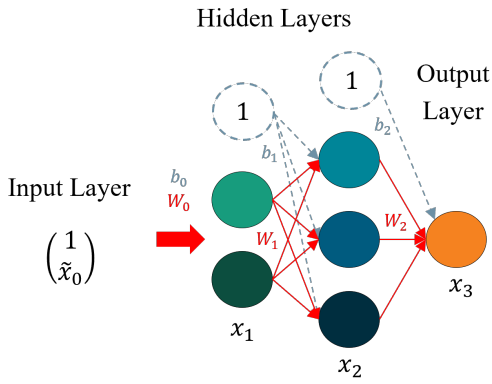


Fig. 1. Fully connected neural network with two hidden layers of size $[2, 3]$ and ReLU activation. The bias is depicted as neurons with dashed lines and appended as a 1 to the input.

Thus, the DT has $\sum_{i=1}^{3}(n_i - 1) = 6$ levels. The number of nodes e_{ij} at each level l_{ij} are given by

$$e_{ij} = 2^j \prod_{\tilde{i}=1}^{i-1} 2^{n_{\tilde{i}}} = 2^{j+\sum_{\tilde{i}=1}^{i-1} n_{\tilde{i}}},$$

where the number of activation regions is $r_i = 2$ for all layers. The decision rule at level l_{ij} representing layer i and neuron j is given by Equation (9) with boundary point $\mathbf{r}_i = 0$. The decision rules are calculated by

$$\sum_{k=1}^{n_0} ({}_m\mathbf{D}_{i-1})_{jk}(\mathbf{x}_0)_k > 0 \quad \text{with } i = 1, 2, 3 \tag{10}$$

with activation function tanh being applied afterwards for the final output.
The second neuron in the first layer corresponds to level $l_{12} = 2$ in the DT with $e_{12} = 4$ nodes. We have the partial activation pattern

$$\mathbf{P}_1(\mathcal{X}_0) = \{(0,0), (0,1), (1,0), (1,1)\}.$$

The next level $l_{21} = 3$ with $e_{21} = 8$ nodes has the activation pattern

$$\mathbf{P}_2(\mathcal{X}_0) = \{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)\}$$

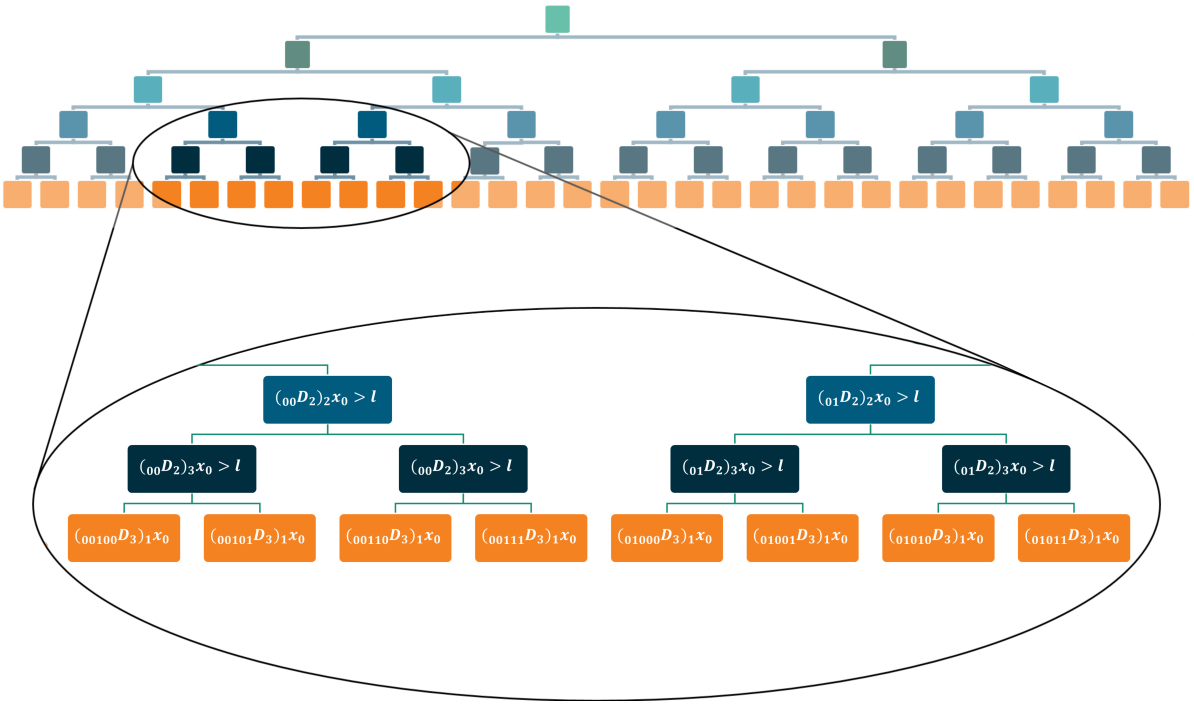


Fig. 2. DT for FcNN with hidden layers of size [2, 3] and ReLU activation. The colors are referring to Figure 1, i.e., green depicts the first, blue the second and orange the output layer. The different shades of the colors denote the individual neurons in each layer. Index m is written in binary system for better visualization of the activation state.

attaching the information from the next layer's first neuron's activation. Each tuple in this pattern encodes the activation states across multiple neurons, For instance, the element $(0, 1, 1)$ indicates that neuron 1 of layer 1 is in region 0 (ReLU slope= 0, hence $[\mathbf{A}_1]_1 = 0$), neuron 2 of layer 1 is in region 1 (ReLU slope= 1, hence $[\mathbf{A}_1]_2 = 1$), and neuron 1 of layer 2 is in region 1 (ReLU slope= 1, hence $[\mathbf{A}_2]_1 = 1$). Figure 2 visualizes the DT construction, showing how each path through the DT corresponds to a unique activation pattern and represents specific decision boundaries within the NN.

3.4 Extension to Convolutional Neural Networks

The transformation described in Section 3.1 and Section 3.2 holds not only for FcNNs but can also be extended for various network architectures, for example CNNs. In the following, we will demonstrate how to transform a CNN into a DT including convolutional and pooling layers, focusing primarily on max pooling layers, though other pooling types can also be used. These layers can then be combined into different CNN architectures.

3.4.1 Convolutional Layers. Our linearization of a convolutional layer builds on the foundation of (Balestriero and Baraniuk, 2020), but adopts a more implementation-oriented notation that aligns with our algorithm RENTT.

Definition 3 (Convolutional Mapping)**.** *Convolutional mappings M_i^C for layer i are defined for input $\tilde{\mathbf{x}}_{i-1}$ with channel $n_{i-1}^c \in \mathbb{N}$, height $n_{i-1}^h \in \mathbb{N}$ and width $n_{i-1}^w \in \mathbb{N}$. For each filter $[\Psi_{i-1}]_c \in \mathbb{N}$ with height k_{i-1}^h, width k_{i-1}^w and indexed with $c = 1, \ldots, k_i^c$, where $k_i^c \in \mathbb{N}$ is the number of filters, the mapping is given by*

$$M_i^C : \mathbb{R}^{n_{i-1}^c \times n_{i-1}^h \times n_{i-1}^w} \longrightarrow \mathbb{R}^{k_i^c \times n_i^h \times n_i^w},$$

$$[\tilde{\mathbf{x}}_{i-1}] \mapsto [\tilde{\mathbf{x}}_i]_c \coloneqq \sigma_i \left(\left(\sum_{k=1}^{n_{i-1}^c} [\Psi_{i-1}]_{c,k} * [\tilde{\mathbf{x}}_{i-1}]_k \right) + [\mathbf{b}_{i-1}]_c \right)$$

$$[\tilde{\mathbf{x}}_i]_{c,w,h} = \sigma_i \left(\left(\sum_{k=1}^{n_{i-1}^c} \sum_{m=1}^{k_{i-1}^h} \sum_{n=1}^{k_{i-1}^w} [\Psi_{i-1}]_{c,k,m,n} \, [\tilde{\mathbf{x}}_{i-1}]_{k,m+w-1,n+h-1} \right) + [\mathbf{b}_{i-1}]_{c,w,h} \right)$$

with bias term $\mathbf{b}$ and $*$ denoting the convolution operation. The square brackets are for clearer notation.

To express this convolution operation as matrix multiplication, the input $\tilde{\mathbf{x}}_{i-1}$ is flattened into a vector. The flattening process converts a 3D tensor into a 1D vector by sequentially processing each slice in row-major order. Starting with the first slice, elements are taken row by row, then the process repeats for subsequent slices. The elements are then concatenated into a single column vector, preserving the original order of slices and rows. Again, to include the bias, a 1 is attached to the front to get $\mathbf{x}_{i-1}$. This whole process is shown in Figure 3. The filters $[\Psi_{i-1}]_c$ are flattened into matrices analogously, and stacked while filled with zeros in-between to obtain $\left[\bar{\Psi}_i\right]_{c,h,k}$, which represents the flattened convolution filter applied to the k-th channel of the input with filter height index h. This process is shown in Figure 4. These flattened filters $\left[\bar{\Psi}_i\right]_{c,h,k} \in \mathbb{R}^{(n_{i-1}^w - k_{i-1}^w + 1) \cdot (n_{i-1}^w - k_{i-1}^w + 1)) \times (n_{i-1}^w)}$ are stacked and replicated to form the weight component of the super convolutional matrix $\mathbf{C}$. The bias vector is obtained by replicating $\mathbf{b}_i$ on all spatial positions w.r.t. the number of filters k^c. Combining both leads to the super convolutional matrix

$$\mathbf{C}_i = \begin{pmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 & \cdots & 0 \\ \mathbf{b}_i & [\Psi_i]_{1,1,1} & \cdots & [\Psi_i]_{1,n_i^h,1} & [\Psi_i]_{1,1,2} & \cdots & [\Psi_i]_{1,n_i^h,2} & \cdots & [\Psi_i]_{1,n_i^h,n_i^c} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \mathbf{b}_i & [\Psi_i]_{k_i^c,1,1} & \cdots & [\Psi_i]_{k_i^c,n_i^h,1} & [\Psi_i]_{k_i^c,1,2} & \cdots & [\Psi_i]_{k_i^c,n_i^h,2} & \cdots & [\Psi_i]_{k_i^c,n_i^h,n_i^c} \end{pmatrix},$$

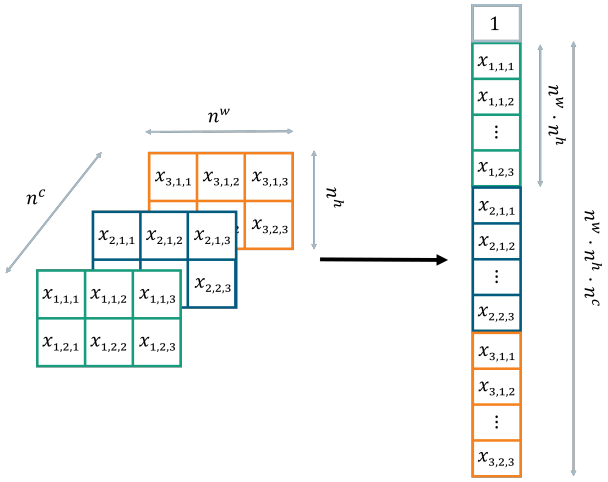


Fig. 3. Flattening process of $\tilde{\mathbf{x}}_i$ with $n^c = 3$, $n^w = 3$ and $n^h = 2$ to obtain $\mathbf{x}_i$. In the image we neglect $\tilde{}$ and index i for better visualization.

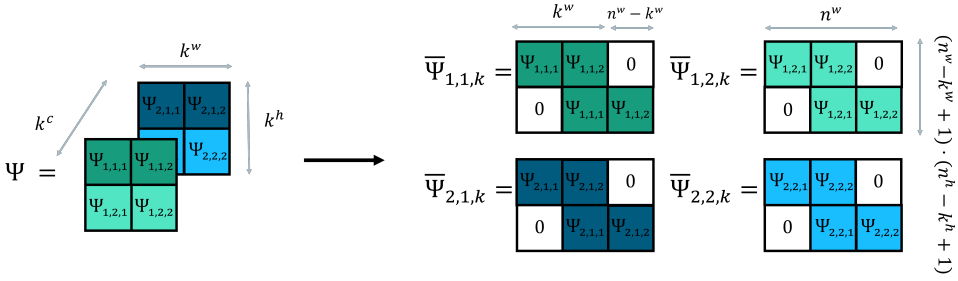


Fig. 4. Flattening process of Ψ_i with $k^c = 2$, $k^w = 2$ and $k^h = 2$ when $n^w = 3$ and $n^h = 2$ to obtain $\left[\bar{\Psi}_i\right]_{c,h,k}$, where c indicates the filter number, h the filter height index and k the channel number of $[\mathbf{x}_i]_k$. The image index i is neglected for better visualization.

with $\mathbf{C}_i \in \mathbb{R}^{n_{i+1} \times n_i}$, where $n_i = n_i^c \cdot (n_i^h \cdot n_i^w)$ and $n_{i+1} = k_{i+1}^c \cdot (n_{i+1}^h - k_{i+1}^h + 1) \cdot (n_{i+1}^w - k_{i+1}^w + 1))$. The super convolutional matrix reformulates the convolutional operation as a matrix multiplication, enabling us to treat convolutional layers equivalently to fully connected layers and apply the same linearization framework described in Section 3.1. Note that $[\Psi_i]_{k,n,1}, [\Psi_i]_{k,n,2}, \ldots, [\Psi_i]_{k,n,n_i^c}$ are identical for each $k = 1, \ldots, k_i^c$ and $n = 1, \ldots, n_i^h$ as indicated in Figure 4. This super convolutional matrix $\mathbf{C}_i$ is also shown in Figure 5.
By using this matrix for $\mathbf{C}_{i-1}$ and the flattened input $\mathbf{x}_{i-1}$, the convolutional mapping can be written as

$$M_i^C : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, \quad \mathbf{x}_{i-1} \mapsto \mathbf{x}_i = \sigma_i \left(\mathbf{C}_{i-1}\mathbf{x}_{i-1}\right) \ .$$

Analogous to the linearization of the fully connected layer in Equation (2), it follows

$$M_i^C : \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}, \quad \mathbf{x}_{i-1} \mapsto \mathbf{x}_i = \mathbf{A}_i \left(\mathbf{C}_{i-1}\mathbf{x}_{i-1}\right) \ .$$

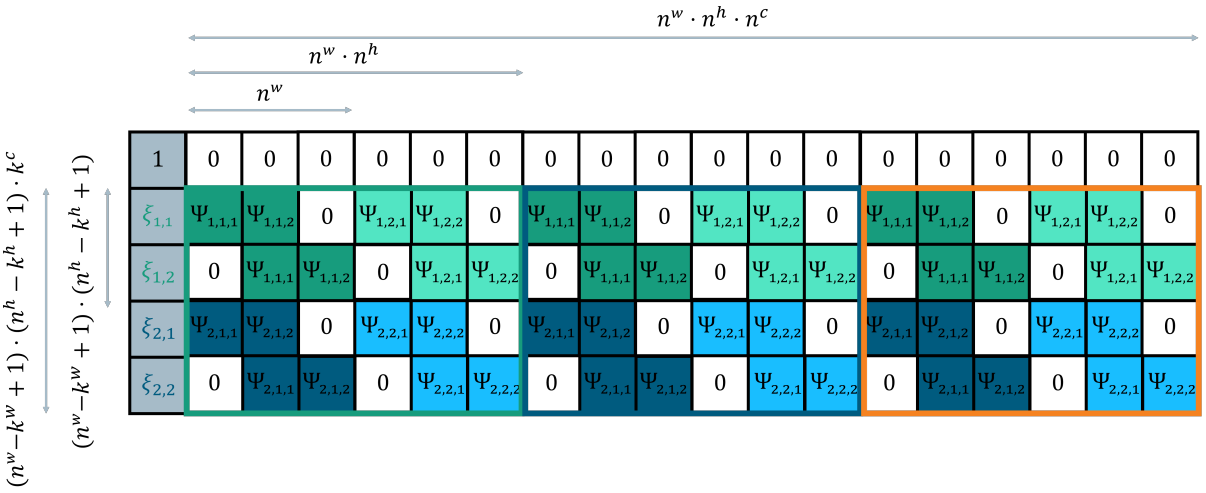


Fig. 5. Super convolutional matrix $\mathbf{C}_i$ of stacked $\left[\bar{\Psi}_i\right]_{c,h,k}$. Note in the image index i is neglected for better visualization.

Now we can introduce the effective weight matrix analogously to Equation (4)

$$\begin{aligned} \mathbf{D}_0^C &\coloneqq \mathbf{C}_0 && \in \mathbb{R}^{n_1 \times n_0}\,, \\ \mathbf{D}_i^C &\coloneqq (\mathbf{C}_i \mathbf{A}_i)\mathbf{D}_{i-1}^C && \in \mathbb{R}^{n_{i+1} \times n_0}\,, \quad i = 1, \ldots, N-1\,, \end{aligned} \tag{11}$$

which allows to formulate the application of multiple convolutional layers as

$$M^C(\mathbf{x}_0) = \sigma_N \left(\mathbf{D}_{N-1}^C \mathbf{x}_0 \right) .$$

3.4.2 Max Pooling Layer. Next, let us introduce a similar reformulation of max-pooling layers.

Definition 4 (Max Pooling). *The max pooling mapping for layer i is defined according to (Balestriero and Baraniuk, 2020) as*

$$P_i^{\max} \colon \mathbb{R}^{n_{i-1}} \to \mathbb{R}^{n_i}\,, \quad \tilde{\mathbf{x}}_{i-1} \mapsto \tilde{\mathbf{x}}_i = \left(\max_{r \in \mathcal{R}_1} [\tilde{\mathbf{x}}_{i-1}]_r, \ldots, \max_{r \in \mathcal{R}_{n_i}} [\tilde{\mathbf{x}}_{i-1}]_r \right)^\top ,$$

thus taking the maximum value of the input $\tilde{\mathbf{x}}_{i-1}$—without the bias dummy—in each region $\mathcal{R}_k$.

These regions $\mathcal{R}_k$ can have different cardinalities, meaning they can vary in size, and can overlap, allowing some parts of the input feature map to be included in multiple regions.

This mapping can also be formulated as a decision set, comparing each input value with the others in one filter region. Therefore, a decision matrix $\mathbf{V}_i^r \in \mathbb{R}^{s_i \cdot p_i \times n_i}$ is used with $p_i = \frac{d_i \cdot (d_i - 1)}{2}$, where $d_i = d_i^w \cdot d_i^h$ represents the size of the pooling filter and s_i denotes the number of pooling regions. It holds $n_i = d_i \cdot s_i + 1$. Thus, $\mathbf{V}_i^r$ is

composed of all s_i pooling filters $\mathbf{\Gamma}_i$ with entries in $\{-1, 0, 1\}$ and adding zeroes on the left for the bias handling according to

$$\mathbf{V}_i^r = \begin{pmatrix} 0 & [\mathbf{\Gamma}_1]_i \\ 0 & [\mathbf{\Gamma}_2]_i \\ \vdots & \vdots \\ 0 & [\mathbf{\Gamma}_{s_i}]_i \end{pmatrix} \in \mathbb{R}^{s_i \cdot p_i \times n_i} \tag{12}$$

$$\text{with, e.g., } [\mathbf{\Gamma}_1]_i = \begin{pmatrix} 1 & -1 & 0 & \cdots & 0 \\ 1 & 0 & -1 & \cdots & 0 \\ & & \vdots & & \\ 1 & 0 & \cdots & 0 & -1 \\ 0 & 1 & -1 & \cdots & 0 \quad 0 \\ & & \vdots & & \\ 0 & 1 & 0 & \cdots & -1 \\ & & \vdots & & \\ 0 & \cdots & 0 & 1 & -1 \end{pmatrix} \in \mathbb{R}^{p_i \times n_i - 1}, \tag{13}$$

where each row contains exactly one 1 and one -1, with zeros elsewhere. The 1 selects the first input value in a pairwise comparison, while the -1 selects the second input value, such that each row computes the difference between two specific input values within the pooling region to determine their relative ordering. Using this decision matrix, the decision rules are given by

$$\mathbf{V}_{i-1}^r \mathbf{x}_{i-1} > \mathbf{0}\,. \tag{14}$$

These decision rules result in a boolean vector, indicating in each entry which of the compared feature values is larger, thereby identifying the largest value in one region. The effective pooling matrix $\mathbf{V}_i^e$ is then used to calculate the output of the pooling layer. This effective pooling matrix consists of a 1 at the positions of the largest feature in a filter region and at the position of the bias, and zeros elsewhere, i.e.,

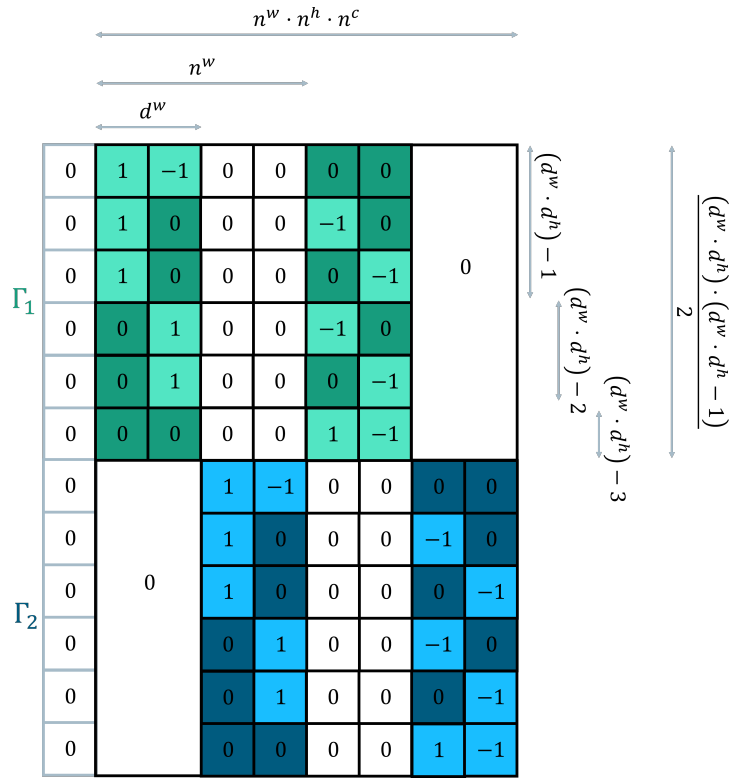
$$\mathbf{V}_i^e \in \{0, 1\} \in \mathbb{R}^{s_i + 1 \times n_i}\,. \tag{15}$$

For the subsequent layer, the effective pooling matrix is used to incorporate the output of the pooling layer into the following decision rules and effective matrices.

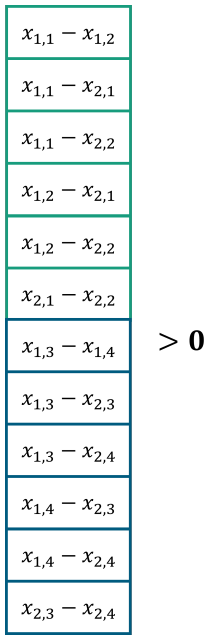
Building on this approach, we can introduce the effective weight matrix for the pooling layers. This is analogous to Equation (4) by simply omitting the activation

$$\begin{aligned} \mathbf{D}_0^P &\coloneqq \mathbf{V}_0^e && \in \mathbb{R}^{n_1 \times n_0}\,, \\ \mathbf{D}_i^P &\coloneqq \mathbf{V}_i^e \mathbf{D}_{i-1}^P && \in \mathbb{R}^{n_{i+1} \times n_0}\,, \quad i = 1, \ldots, N-1\,. \end{aligned} \tag{16}$$

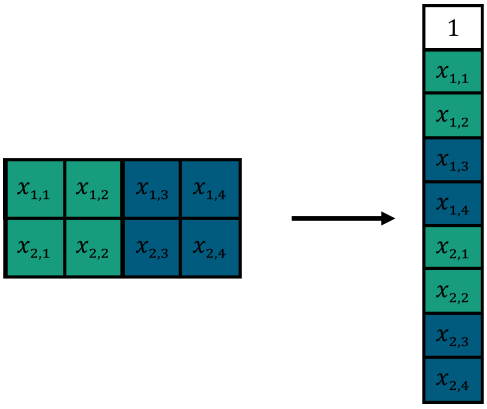
An example of this transformation for $\mathbf{x} = (1, 2, 1, 4, -1, -4, 2, 3)^\top$ is presented in Figures 6. Figure 6(a) displays the decision matrix $\mathbf{V}^r$ along with the two filters $\mathbf{\Gamma}_1$ and $\mathbf{\Gamma}_2$. The resulting decision rules, which create the paths in the DT, are shown in Figure 6(b). Figure 6(c) illustrates the input in both matrix form and its flattened vector version. Finally, Figure 6(d) presents the effective matrix $\mathbf{V}^e$ corresponding to our example.



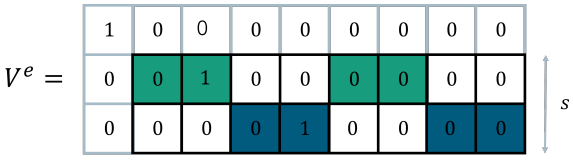
(a) Example for $\mathbf{V}^r$ consisting of two filters $\mathbf{\Gamma}_1$ and $\mathbf{\Gamma}_2$, which are colored green and blue, with $d^w = 2$, $d^h = 2$ and $s = 2$. We choose $n^w = 4$, $n^h = 2$, and $n^c = 1$. Image index i is neglected for better visualization.



(b) Example for resulting decision rule as in Equation (14).



(c) Example for flattened $\mathbf{x}$ used for max pooling with two filter regions colored green and blue.



(d) Example for resulting effective matrix $\mathbf{V}^e$ for max pooling as in Equation (15) when $\mathbf{x}_{1,2}$ and $\mathbf{x}_{1,4}$ are largest. Note in the image index i is neglected for better visualization.

Fig. 6. Example for the transformation of a max pooling layer.

3.4.3 Decision Tree Representation of Convolutional Neural Networks. The convolutional and pooling layers can be combined and transformed into a DT analogously to the FcNN by choosing the effective weight matrix $\mathbf{W}_i = \mathbf{C}_i$ or $\mathbf{W}_i = \mathbf{V}_i^e$ for each layer i and using the decision rule of Equation (9) for a convolutional layer similar to the FcNN or substitute it for a pooling layer via Equation (14). For convolutional layers the depth of the DT is determined as for fully connected layers, defined in Equation (7), replacing the number of nodes with

$$n_i \coloneqq k_i^c \cdot (n_i^h - k_i^h + 1) \cdot (n_i^w - k_i^w + 1)) \ .$$

For maxpooling, the number of nodes is replaced by

$$n_i \coloneqq p_i = \frac{d_i \cdot (d_i - 1)}{2} s_i \quad .$$

Remark 5 (Number of Nodes). *When containing only the information about the maximum values instead of the entire ranking of the feature values, the number of nodes can also be given by*

$$n_i \coloneqq s_i \cdot d_i \ .$$

These types of layers can be combined for different network architectures, for example, to a typical CNN

$$M^C : \mathbb{R}^{n_0} \to \mathbb{R}^{n_N}, \quad \mathbf{x}_0 \mapsto M^C(\mathbf{x}_0) \coloneqq M_N^N \circ P_{N-1}^{\max} \circ M_{N-2}^C \circ \ldots \circ P_2^{\max} \circ M_1^C(\mathbf{x}_0)$$

comprising an alternating sequence of convolutional and pooling layer as well as one fully connected output layer. For this network, the previously shown decision rules and effective matrices just have to be combined, so that a DT similar to the one for the FcNN is created.

3.5 Extension to Recurrent Neural Networks

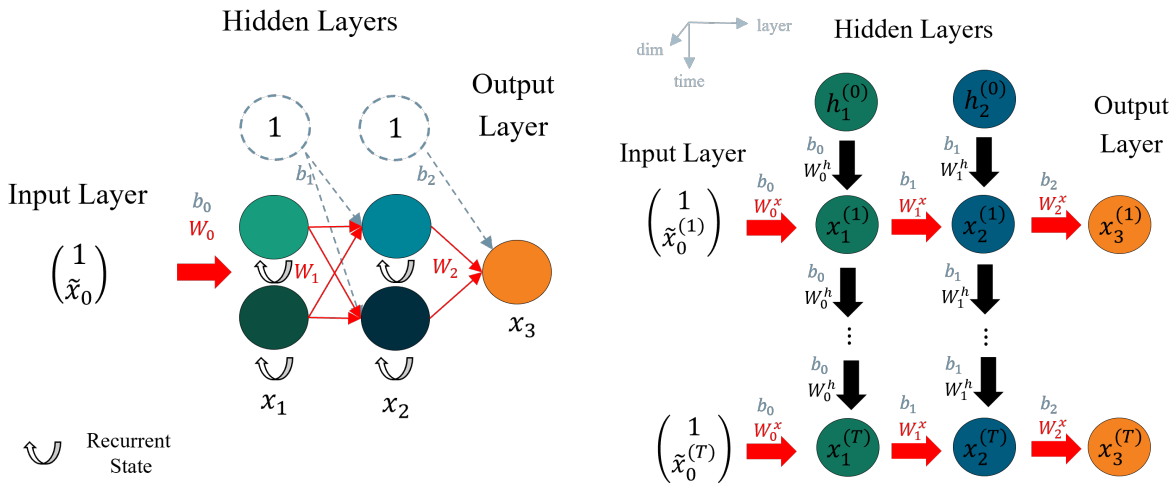
The RNNs were introduced by (Rumelhart et al., 1986) allowing backward connections where the output of one layer at time t is fed back as input to the network at the next time step $t + 1$. This introduced an internal state to the NN and is beneficial for processing sequential data like time series or text.

Definition 5 (Recurrent layers)**.** *Following (Goodfellow et al., 2016) the mapping of a recurrent layer with size n_i is defined as*

$$M_i^R : \mathbb{R}^{n_{i-1}+n_i^h} \to \mathbb{R}^{n_i} \ ,$$

$$\mathbf{x}_{i-1}^{(t)} \mapsto \mathbf{x}_i^{(t)} \coloneqq h_i^{(t)} = \sigma_i^{(t)}(\mathbf{W}_i^h h_i^{(t-1)} + \mathbf{W}_{i-1}^x \mathbf{x}_{i-1}^{(t)}) \ , \quad i = 1, \ldots, N-1 \ . \tag{17}$$

Here, $h_i^{(t)}$ denotes the hidden state of layer i at time step $t = 1, \ldots, T$, with arbitrary initial hidden state $h_i^{(0)} \in \mathbb{R}^{n_i^h}$, where n_i^h is the dimension of the hidden state. The recurrent weight matrix $\mathbf{W}_i^h \in \mathbb{R}^{n_i^h \times n_i^h}$ connects the previous hidden state $h_i^{(t-1)}$ to the current time step, while the input weight matrix $\mathbf{W}_{i-1}^x \in \mathbb{R}^{n_i \times n_{i-1}}$ processes the input features $\mathbf{x}_{i-1}^{(t)}$ from the previous layer. We assume that the layer output directly corresponds to the hidden state, which implies $n_i^h = n_i$.



(a) Compact version (b) Unfold version

Fig. 7. Recurrent neural network with two hidden layers and two neurons each. The dotted circles and B indicating the bias consideration. (a) Classical view with arrows indicating the recurrent neurons. (b) Unfolded network showing the recurrent states downwards.

An example RNN is shown in Figure 7 in a compact and an unfolded view. The bias term is taken into account as in the sections above by adding a 1 to the initial feature vector $\mathbf{x}_0^{(t)}$ and the initial hidden state $h_i^{(0)}$ and by adding an identity function to the activation function σ_i. Analogously, the weight matrix is expanded by 1 at the upper left and filled with 0. Both the hidden state $h_i^{(t)}$ and the feature vector $\mathbf{x}_i^{(t)}$ change over time, but the weight matrices $\mathbf{W}_i^{\mathrm{h}}$ and $\mathbf{W}_i^{\mathrm{x}}$ remain constant over time, though they differ based on whether they are applied to the hidden state vector or the feature vector. The output of the recurrent layer can be rewritten by linearizing the activation function σ_i analogous to the FcNN

$$\mathbf{x}_i^{(t)} = h_i^{(t)} = \mathbf{A}_i^{(t)}(\mathbf{A}_i^{(t)}(\mathbf{W}_{i-1}^{\mathrm{x}}\mathbf{x}_{i-1}^{(t)} + \mathbf{W}_i^{\mathrm{h}}h_i^{(t-1)}))$$

with activation matrix $\mathbf{A}_i^{(t)} \in \mathbb{R}^{n_i \times n_i}$. This can be formulated in dependence of the initial hidden state $h_i^{(0)}$ at time $t = 0$ using the notation $\mathbf{x}_i = \left(\mathbf{x}_i^{(1)}, \ldots, \mathbf{x}_i^{(T)}\right)^\top \in \mathbb{R}^{n_i \cdot T}$, $h_i = \left(h_i^{(0)}, \ldots, h_i^{(0)}\right)^\top \in \mathbb{R}^{n_i \cdot T}$ and $\mathbf{A}_i = (\mathbf{A}_i^{(1)}, \ldots, \mathbf{A}_i^{(T)}) \in \mathbb{R}^{n_i \times n_i \cdot T}$ as

$$\mathbf{x}_i = \mathbf{A}_i \left(\begin{pmatrix} 1 & & \mathbf{0} \\ \prod_{k=1}^{1} \mathbf{W}_i^{\mathrm{h}} \mathbf{A}_i^{(k)} & \ddots & \\ \vdots & & \\ \prod_{k=1}^{T-1} \mathbf{W}_i^{\mathrm{h}} \mathbf{A}_i^{(k)} & \prod_{k=2}^{T-1} \mathbf{W}_i^{\mathrm{h}} \mathbf{A}_i^{(k)} \quad \cdots \quad & 1 \end{pmatrix} \begin{pmatrix} \mathbf{W}_{i-1}^{\mathrm{x}} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{W}_{i-1}^{\mathrm{x}} \end{pmatrix} \mathbf{x}_{i-1} \right.$$

$$\left. + \begin{pmatrix} 1 & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \prod_{k=1}^{T-1} \mathbf{W}_i^{\mathrm{h}} \mathbf{A}_i^{(k)} \end{pmatrix} \begin{pmatrix} \mathbf{W}_i^{\mathrm{h}} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{W}_i^{\mathrm{h}} \end{pmatrix} h_i \right).$$

Remark 6. *(Product Operator) When $k > T-1$, we consider the product operator to be* 1.

To simplify the subsequent computations and expressions we make the following assumption.

Assumption 2 (Hidden State Initialization)**.** *For simplicity we assume that $h_i^{(0)}$ is initialized as zero.*

Therefore, the second part of the equation involving $h_i^{(0)}$ can be omitted in the following. We define the effective weight matrices as

$$\begin{aligned} \mathbf{D}^h_{t,p,i} &= 1 && \text{with } p = t\,, \\ \mathbf{D}^h_{t,p,i} &= \prod_{k=p}^{t-1} \mathbf{W}^{\mathrm{h}}_i \mathbf{A}^{(k)}_i && \text{with } p = t-1,\dots,1\,. \end{aligned}$$

Using these simplifies the previous notation to

$$\mathbf{x}_i = \mathbf{A}_i \left(\begin{pmatrix} \mathbf{D}^h_{1,1,i} & & \mathbf{0} \\ \vdots & \ddots & \\ \mathbf{D}^h_{T,1,i} & \cdots & \mathbf{D}^h_{T,T,i} \end{pmatrix} \begin{pmatrix} \mathbf{W}^{\mathrm{x}}_{i-1} & & \mathbf{0} \\ & \ddots & \\ \mathbf{0} & & \mathbf{W}^{\mathrm{x}}_{i-1} \end{pmatrix} \mathbf{x}_{i-1} \right),$$

which can be combined into

$$\mathbf{R}_{i-1} = \begin{pmatrix} \mathbf{D}^h_{1,1,i}\mathbf{W}^{\mathrm{x}}_{i-1} & & \mathbf{0} \\ \vdots & \ddots & \\ \mathbf{D}^h_{T,1,i}\mathbf{W}^{\mathrm{x}}_{i-1} & \cdots & \mathbf{D}^h_{T,T,i}\mathbf{W}^{\mathrm{x}}_{i-1} \end{pmatrix} \in \mathbb{R}^{n_i \cdot T \times n_{i-1} \cdot T}$$

Introducing the overall effective weight matrices

$$\begin{aligned} \mathbf{D}^R_0 &\coloneqq \mathbf{R}_0 && \in \mathbb{R}^{n_1 \cdot T \times n_0 \cdot T}\,, \\ \mathbf{D}^R_i &\coloneqq \mathbf{R}_i \mathbf{D}^R_{i-1} && \in \mathbb{R}^{n_{i+1} \cdot T \times n_0 \cdot T}\,, \quad i = 1,\dots,N-1\,, \end{aligned} \tag{18}$$

we can formulate the mapping of a RNN with N layers similar to the FcNN

$$\mathbf{x}_i = \mathbf{A}_i \mathbf{D}^R_i \mathbf{x}_0\,.$$

Remark 7 (Including Initial Hidden State). *If Assumption 2 does not hold, the derivation described above still holds when introducing a combined expression*

$$\mathbf{x}^h_{i-1} = \begin{pmatrix} h_i \\ \mathbf{x}_{i-1} \end{pmatrix} \in \mathbb{R}^{n_i \cdot T + n_{i-1} \cdot T}$$

and

$$\mathbf{R}_i = \begin{pmatrix} \mathbf{1} & 0 \\ \mathbf{R}^h_i & \mathbf{R}^x_{i-1} \end{pmatrix} \in \mathbb{R}^{2 \cdot n_i \cdot T \times n_i \cdot T + n_{i-1} \cdot T}$$

with $\mathbf{R}^x_{i-1}$ being the previous used $\mathbf{R}_{i-1}$ and

$$\mathbf{R}^h_i = \begin{pmatrix} \mathbf{D}^h_{1,1,i}\mathbf{W}^h_{i-1} & & \boldsymbol{0} \\ & \ddots & \\ \boldsymbol{0} & & \mathbf{D}^h_{T,1,i}\mathbf{W}^h_{i-1} \end{pmatrix} \in \mathbb{R}^{n_i \cdot T \times n_i \cdot T}\,.$$

This modification includes the contributions of both components—feature vector $\mathbf{x}$ and hidden state h—within the equation.

3.5.1 Decision Tree Representation of Recurrent Neural Networks. Analogous to the FcNN we can build a DT of the RNN where every level in the DT represents one neuron in a layer of the NN at one time step $t = 1, \dots, T$. Thus, one decision rule in a node gives the activation state of one neuron at one time step. The depth of the DT for an RNN is the same as for a FcNN multiplied by the number of time steps indicated by the additional dimension T in the previously derived formulas in comparison to the FcNN, i.e.

$$l_{ijt} = j + \sum_{\tilde{i}=1}^{i-1} \sum_{\tilde{t}=1}^{t} (n_{\tilde{i}} - 1) \ . \tag{19}$$

The number of nodes in each level is again given by Equation (8).

4 Runtime Efficient Network to Tree Transformation (RENTT) Algorithm

While the theoretical foundations for transforming NNs into DTs have been established, direct implementation faces significant computational challenges. A complete transformation would generate a very large amout of nodes as shown in Equation (8), rendering the approach intractable for practical applications. This section presents the Runtime Efficient Network to Tree Transformation (RENTT) algorithm, which achieves computational feasibility through sparse tree construction based on empirically observed activation patterns. By building only necessary decision paths, RENTT maintains mathematical equivalence while significantly reducing computational complexity.

4.1 Implementation

From an implementation perspective, the transformed decision tree is represented as a collection of node objects, each encapsulating the essential properties required for efficient traversal and prediction. Each node maintains:

- Unique identifier (ID) following a binary tree numbering scheme where child nodes are assigned.
- Activation pattern encoding the path of neuron activations from root to current node.
- Effective weight matrix containing the effective linear weights and bias for decision-making.
- Sample memory storing training data points that flow through the node during construction.

The RENTT algorithm builds a sparse DT using ante-hoc pruning. This method builds only the relevant paths of the DT, selected by the occurring activation patterns obtained from the training data set $\mathcal{X}_0$. The implementation is shown in Algorithm 1. The process begins with the identification of activation patterns of the training samples. These patterns serve as a guide for pruning the DT, enabling us to build only branches corresponding to used activation patterns. This ensures we retain solely those nodes that process at least one sample and thus contribute to the model's accuracy or efficiency. This selective approach not only conserves memory but also accelerates the transformation of the NN into a DT representation. The DT can still be appended with additional necessary paths corresponding to new activation patterns not included in the training set.

Our implementation of Algorithm 1 is developed in Python, utilizing TensorFlow-based NNs to ensure comparability with the algorithm presented by (Nguyen et al., 2020). It accommodates NNs for both classification (binary and multiclass) and regression tasks. The exact details of the implementation can be found in our code on GitHub[1].

Regarding the applicability to CNNs and RNNs, we note the following.

Remark 8 (CNN and RNN Implementation). *Algorithm 1 can be adapted for CNNs and RNNs through these adjustments:*

(1) To transform CNNs, replace the set of weight matrices $\mathcal{W}$ with the set of filters $\ominus = \{\Psi_0, \cdots, \Psi_{N-1}\}$ and substitute Equation (4) with equations (11) or (16).

[1] https://github.com/HelenaM23/RENTT

(2) To transform RNNs, additionally incorporate the set of hidden state weight matrices $\mathcal{W}^h = \{\mathbf{W}_0^h, \cdots, \mathbf{W}_{N-1}^h\}$ *and substitute Equation* (4) *with Equation* (18).

Algorithm 1 Runtime Efficient Network to Tree Transformation (RENTT)

Input: Sets of input feature vectors $\mathcal{X}_0$, set of weight matrices $\mathcal{W} = \{\mathbf{W}_0, \ldots, \mathbf{W}_{N-1}\}$, and set of activation functions $\mathcal{A} = \{\sigma_1, \ldots, \sigma_N\}$
Output: DT T

1: Set $\mathbf{D}_0$ using Equation (4)
2: Create root node q_0 in DT T with ID, decision rule and samples in this node
3: **for** $i = 1, \ldots, N-1$ **do**
4: Initialize set of unique partial activation patterns $\mathcal{P}_i = \emptyset$
5: Initialize set of effective matrices $\mathcal{D}_i = \emptyset$
6: **for** $\mathbf{x}_0$ in $\mathcal{X}_0$ **do** ▷ Calculate used activation patterns
7: **if** $i = 1$ **then**
8: Calculate partial activation pattern $\mathbf{P}_i(\mathbf{x}_0)$
9: **else**
10: Reuse last partial activation pattern $\mathbf{P}_{i-1}(\mathbf{x}_0)$ (see Definition 2)
11: **end if**
12: $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \{\mathbf{P}_i(\mathbf{x}_0)\}$
13: **end for**
14: **for** ${}_m\mathbf{P}_i$ in $\mathcal{P}_i$ **do** ▷ Pruning: only nodes for occupied activation patterns
15: Calculate effective weight matrix ${}_m\mathbf{D}_i$ associated to ${}_m\mathbf{P}_i$ via Equation (4)
16: $\mathcal{D}_i \leftarrow \mathcal{D}_i \cup \{{}_m\mathbf{D}_i\}$
17: Append node q_i associated to ${}_m\mathbf{P}_i$ including ID, ${}_m\mathbf{P}_i$, ${}_m\mathbf{D}_i$ for decision rule and samples in this node to T
18: **end for**
19: **end for**
20: **if** σ_N is piecewise linear **then** ▷ Add activation of last layer for regression tasks
21: Repeat line 4–16 for $i = N$
22: Append leaf node q_N associated to ${}_m\mathbf{P}_N$ including ID, ${}_m\mathbf{P}_N$, ${}_m\mathbf{D}_N$ and samples in this node to T
23: **else** ▷ E.g., sigmoid or softmax activation for classification tasks
24: Use node q_{N-1} as leaf node and handle activation function σ_N separately
25: **end if**

4.2 Efficiency of the RENTT Algorithm

In this section, we present a series of experiments designed to evaluate the performance of Algorithm 1 in terms of runtime and memory consumption, particularly as the complexity of the neural architecture increases. All these numerical experiments underline the theoretical findings of Section 3, that the reformulation of NNs into DTs is exact. This means that loss, accuracy and output of both NN models and DT are equivalent up to machine precision, as can be seen in Tables 2 and 3 of the appendix. While previous sections have laid out the theoretical foundations confirming the transformation's validity and accuracy, it is essential to substantiate these claims through rigorous testing that assesses computational efficiency.

We compare the runtime and memory consumption of RENTT with the algorithm ECDT provided by (Nguyen et al., 2020), where the latter is only applicable for classification networks using ReLU activation functions. Thus, we use the classification data sets Iris, Diabetes Class and Car Evaluation described in Appendix A. All experiments are performed on a workstation equipped with an Intel Core i9-14900K processor, NVIDIA RTX 4090 GPU with 24GB VRAM, and 96GB of system RAM. We evaluate models with varying numbers of hidden neurons. For each configuration, ten models are trained and evaluated five times using different data samples to assess variability. While model-to-model differences contribute more to variability than data sampling variations, the overall variability observed is negligible.

Due to hardware limitations, transformations for ECDT on networks with a larger number of hidden neurons than presented here could not be executed. Equation (8) reveals that the number of leaf nodes depends solely on the total number of neurons and the number of linear regions in the activation function, irrespective of their distribution across layers. Consequently, we observe that runtime and memory consumption remain consistent regardless of neuron distribution, so only the total number of hidden neurons is reported. In our analysis we employ a log-log plot, anticipating the fitting of a straight line, which indicates the power-law relationship $y = a \cdot x^b$ between y, the runtime or memory consumption, and x, the number of neurons. Exponent b reflects the scaling behavior or growth rate of runtime or memory consumption as the number of neurons increases. To focus on the algorithms' limiting behavior and examine their asymptotic behavior, we fitted the line only for high numbers of neurons. In this context, we employ the Bachmann-Landau notation O (Mala and Ali, 2022) to describe how the algorithms perform. This approach allows assessing scalability and efficiency, particularly under conditions where hardware constraints limit practical execution for larger networks.

4.2.1 Computational Cost: Reducing Runtime. In this section, we analyze the runtime experiments of our proposed algorithm RENTT compared to the ECDT algorithm (Nguyen et al., 2020). The results are illustrated in Figure 8, which show the runtime performance across different data sets and configurations. The computational time is shown for both RENTT and ECDT as a function of the number of hidden neurons x.

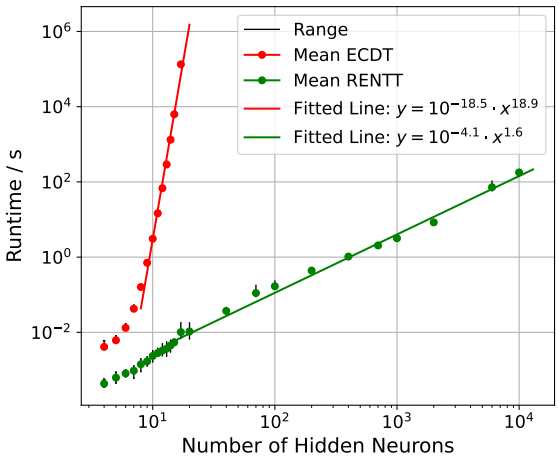
Similar trends are observed for all data sets: the results indicate that RENTT consistently outperforms ECDT, particularly as the complexity of the NN (i.e., the number of hidden neurons) increases. The fitted lines for both algorithms demonstrate that while ECDT's runtime grows exponentially with an order of magnitude of $O\left(x^{18.9}\right)$ to $O\left(x^{19.0}\right)$, RENTT exhibits significantly better scalability with $O\left(x^{1.3}\right)$ to $O\left(x^{1.5}\right)$. If hardware constraints were no limitations for 10^4 hidden neurons, RENTT needs only $1.6 \cdot 10^2$ s to $1.9 \cdot 10^3$ s, which are around 3 min to 30 min, where ECDT takes $(1.3 - 1.9) \cdot 10^{57}$ s, which are around $6 \cdot 10^{49}$ years.

4.2.2 Memory Consumption: Reducing Memory. In this section, we evaluate the memory usage of the RENTT algorithm in comparison to the ECDT algorithm across multiple data sets. Memory efficiency is crucial when dealing with large NNs, as excessive memory usage can hinder practical applications, especially in resource-constrained environments.

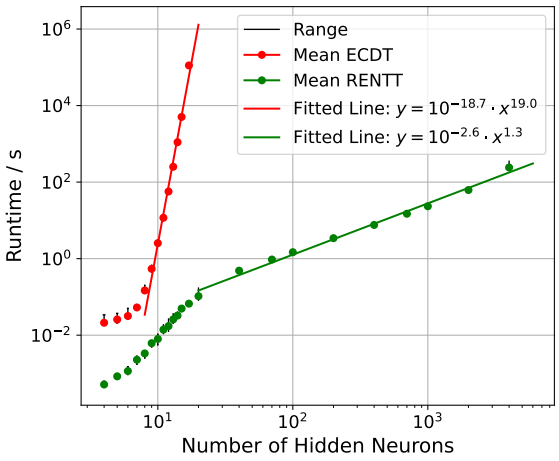
Figure 9 illustrates the memory consumption for both RENTT and ECDT as a function of the number of hidden neurons. The experiments were conducted under controlled conditions, with multiple transformations carried out to ensure reliability. Our hardware resources were limited to 96 GB Random-access memory (RAM). Thus, the results of the memory consumption in Figure 9 are below the 10^5 MB line. The results clearly show that RENTT exhibits significantly lower memory consumption than ECDT, particularly as the number of hidden neurons increases. ECDT has a scaling of $O(x^{8.6})$ to $O(x^{8.7})$, RENTT has a scaling of only $O(x^{1.7})$ to $O(x^{1.9})$. If hardware constraints were no limitations for 10^4 hidden neurons, RENTT would need 31 GB to 630 GB RAM, where ECDT would take $2.0 - 4.0 \cdot 10^{23}$ GB, which are $2.0 - 4.0 \cdot 10^{20}$ TB. Although this memory consumption remains high

in absolute terms, the improvement over ECDT is substantial. This limitation is further discussed in Section 7, but the strong difference highlights RENTT's superior memory efficiency, making it a more suitable choice for handling complex NN architectures.

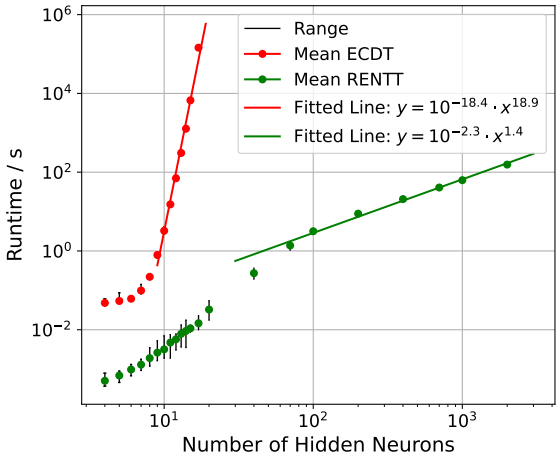
This analysis is critical for understanding the practical implications of using RENTT for large-scale applications. By optimizing memory consumption, RENTT allows for the transformation of larger and more complex NNs into interpretable DTs without overwhelming system resources.



(a) Iris data set

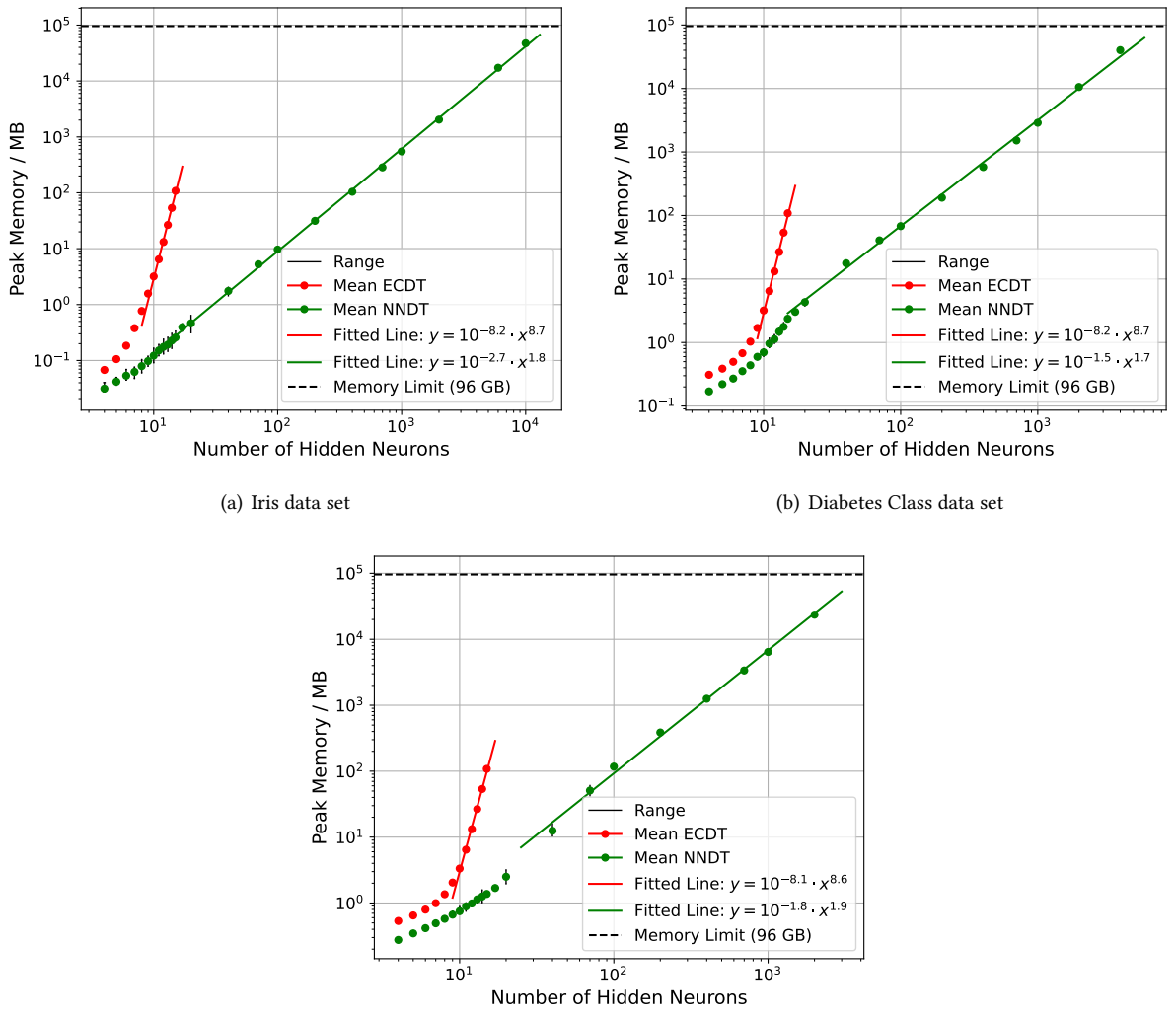


(b) Diabetes Class data set



(c) Car Evaluation data set

Fig. 8. Computational time as a function of the number of hidden neurons for ECDT algorithm (Nguyen et al., 2020) (green) and our RENTT algorithm (red). Standard deviations are represented by the error bars, determined through the transformation of multiple models of the same size. Additional lines are fitted to determine the order of scaling for a large number of neurons.



(a) Iris data set

(b) Diabetes Class data set

(c) Car Evaluation data set

Fig. 9. Maximum RAM consumption as a function of the number of hidden neurons for ECDT algorithm (Nguyen et al., 2020) (green) and our RENTT algorithm (red). Standard deviations are represented by the error bars, determined through the transformation of multiple models of the same size. Additional lines are fitted to determine the order of scaling for a large number of neurons.

The experiments confirm that RENTT significantly outperforms ECDT in terms of computational efficiency and scalability across various data sets. As the complexity of NNs increases, the RENTT algorithm maintains manageable runtime and memory consumption, making it a practical solution for transforming complex NNs into interpretable DTs. The experiments demonstrate that our approach not only preserves the original NN results as derived in the previous sections, but does so in a computationally efficient manner.

5 Feature Importance Definition (RENTT-FI)

As previously shown, NNs can be transformed into equivalent DTs. As they not only produce the exact same output as the NNs, but also have the same "inner" logic, they provide a way to inspect NNs without the fidelity problems of other post-hoc XAI methods (Bhaumik et al., 2022; Fresz et al., 2024). But due to the DTs being multivariate and similarly complex to the NN, this interpretability remains theoretical in nature without further ways to simplify the information contained in the DTs. Different explanation types, at least for networks with piecewise linear activation functions in the last layer, could be created via the DTs—all of them with the benefit of exact calculations and thus, perfect correctness:

- Counterfactual explanations: What would need to change in the input to receive a different outcome—either within the same input region (activation pattern) or across regions.
- Robustness, i.e., how close a decision is to the next activation pattern.
- Visualization of the activation patterns to show which regions in the input space would yield the same result for a given data sample.
- Local linear models of the NN to yield intelligible local decision rules and surrogate models, respectively.

The derivation of these explanation types is left for future work. In the following, we focus on aggregating the DT information by means of feature importance (FI) explanations, as they closely reflect how DTs operate and can be presented without losing information for both local and regional cases. This sort of FI will be called RENTT-FI in the following. FI explanations highlight the most "important" features in the input and how much they contributed to the model output. Such explanations can either be local, i.e., show the most important features of a single input sample, or global, i.e., show the most important features over the entire input space. Note that approaches in-between are also possible, as FI can also be calculated on a class-wise basis for classification tasks.

The term feature importance is often not clearly defined and multiple interpretations are possible—either as the feature effect **FE** specifying the weighting of the different input features (how much the prediction changes for each unit increase in the feature), or the feature contribution **FC**, showing how much a feature contributes to the final prediction. Most methods yield only one of these interpretations. In contrast, our approach provides both: the effect **FE**, corresponding to the effective weight matrix $\mathbf{D}_{N-1}$, and the contribution **FC**, as defined in the following. We provide the feature direction **FD** separately, indicating whether each feature contributes positively or negatively to the prediction. Since RENTT-FI provides both interpretations, we will use this distinction in the following.

For a DT and thus, the NN in question, the local feature contribution of an input feature vector $\mathbf{x}_0$ regarding feature j can be calculated as

$$\mathbf{FC}_{\text{local}l,j} = [{}_m\mathbf{D}_{N-1}]_{l,j} \cdot [\mathbf{x}_0]_j \,, \tag{20}$$

Here, ${}_m\mathbf{D}_{N-1}$, including all $\mathbf{x}_0$ with same activation pattern, is the effective weight matrix of the output layer, as defined in Equation (4), corresponding to the input region m, that contains $\mathbf{x}_0$. This matrix is dependent on $\mathbf{x}_0$. It is provided in the leaf node and also indicates the feature effect. Often the weight matrix for the output layer is a column vector, which implies $l = 0$. For multiclass classification involving n classes $l = 0, \ldots, n-1$, the feature contribution can be calculated separately for each class to create class-specific explanations. The local feature contribution can now be aggregated by averaging it for each feature j over all samples in the set of input feature vector $\mathcal{X}_0$ per input region m into a regional feature contribution, such as

$$\mathbf{FC}_{\text{regional}}{}^{m}_{l,j} = \frac{1}{|\mathcal{M}|} \sum_{\mathcal{M}} |[{}_m\mathbf{D}_{N-1}]_{l,j} \cdot [\mathbf{x}_0]_j| \,, \tag{21}$$

with $\mathcal{M} = \{\mathbf{x}_0 \in \mathcal{X}_0 \mid \mathbf{P}(\mathbf{x}_0) = {}_m\mathbf{P}\}$. The feature direction is calculated via

$$\mathbf{FD}_{\text{regional}}{}^{m}_{l,j} = \text{sign}\left(\frac{1}{|\mathcal{M}|}\sum_{\mathcal{M}}\left[{}_m\mathbf{D}_{N-1}\right]_{l,j}\cdot\left[\mathbf{x}_0\right]_j\right). \tag{22}$$

The regional feature effect is given by the effective weight matrix for this region m per class l and feature j according to

$$\mathbf{FE}_{\text{regional}}{}^{m}_{l,j} = \left[{}_m\mathbf{D}_{N-1}\right]_{l,j}.$$

Note that all samples within region m share the same local feature effect.

We can also calculate an average feature contribution over all input regions to obtain a global feature contribution

$$\begin{aligned}\mathbf{FC}_{\text{global}_{l,j}} &= \frac{1}{|\mathbf{P}(\mathcal{X}_0)|}\sum_{m=1}^{|\mathbf{P}(\mathcal{X}_0)|}\frac{1}{|\mathcal{M}|}\sum_{\mathcal{M}}\left|\left[{}_m\mathbf{D}_{N-1}\right]_{l,j}\cdot\left[\mathbf{x}_0\right]_j\right| \\ &= \frac{1}{|\mathbf{P}(\mathcal{X}_0)|}\sum_{m=1}^{|\mathbf{P}(\mathcal{X}_0)|}\mathbf{FC}_{\text{regional}}{}^{m}_{l,j}\end{aligned}$$

with feature direction

$$\mathbf{FD}_{\text{global}_{l,j}} = \text{sign}\left(\frac{1}{|\mathbf{P}(\mathcal{X}_0)|}\sum_{m=1}^{|\mathbf{P}(\mathcal{X}_0)|}\frac{1}{|\mathcal{M}|}\sum_{\mathcal{M}}\left[{}_m\mathbf{D}_{N-1}\right]_{l,j}\cdot\left[\mathbf{x}_0\right]_j\right) \tag{23}$$

and global feature effect for class l and feature j

$$\mathbf{FE}_{\text{global}_{l,j}} = \frac{1}{|\mathbf{P}(\mathcal{X}_0)|}\sum_{m=1}^{|\mathbf{P}(\mathcal{X}_0)|}\left[{}_m\mathbf{D}_{N-1}\right]_{l,j}.$$

6 Feature Importance Experiments

In this section, we present comparisons of our FI method with other methods commonly used in literature. Our proposed RENTT-FI approach analyzes features at local, regional, and global scales, capturing both feature contributions and feature effects. Methods in literature provide only feature contributions while RENTT-FI also computes feature effects. The transformation of NNs into equivalent DTs can be used to study different questions:

(1) For two functionally equivalent models, the NN and its DT, do the common FI methods produce the same FI values?
(2) Do the common FI methods and RENTT-FI show the expected results for simple data sets? Are their results similar for more complex ones?
(3) What are the performance (dis)advantages of the FI methods vs. RENTT-FI?

While (1) is more of a sanity check of the FI methods, also referred to as implementation-invariance (Nauta et al., 2023), (2) tests the common FI methods for correctness, defined by them conforming to the ground truth FI produced by RENTT-FI, and (3) refers to the implementation details and practical applicability of RENTT-FI.

6.1 Common Feature Importance Methods

To comprehensively compare FI methods across different datasets, we employ a diverse set of both local and global FI methods, as described in the following. This approach enables us to capture various perspectives—from individual prediction explanations to model-wide importance—thereby providing a more robust foundation for our comparative analysis.

All methods are used with their default values, with the background data samples being set to at most 1000 training data samples to reduce runtime and memory complexity. For the smaller data sets, all training data samples were used as background samples.

6.1.1 Local Methods. For the local FI methods, we employ the toolbox dalex (Baniecki et al., 2021). Three different local FI methods are used, as described in the following.

One of the first XAI methods for machine learning decisions was Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016). It uses simpler models to approximate the decision boundaries of more complex machine learning models, such as NNs. Interpretable models, such as linear regression, serve as these simpler models to create FI explanations for data instances.

As a systematization of previous explanation methods, SHapley Additive Explanations (SHAP) (Lundberg and Lee, 2017) was proposed to provide local explanations by approximating Shapley Values, a game theoretic concept. Compared to other explanation methods, this approach provides interesting theoretical properties: Local Accuracy, Missingness, and Consistency. Local Accuracy means that the explanation conforms to the model within a small area around the data sample in question. Missingness indicates that features missing in the original input have a FI of 0, and Consistency describes that increasing the influence of a feature results in a corresponding increase in its FI. While useful in theory, SHAP explanations are criticized, for example, for the resulting explanations being too complex for users (Kumar et al., 2020), FI being attributed to features without impact on the result (I. Covert et al., 2021; Merrick and Taly, 2020), or for the actual implementation of the methods not holding the theoretical properties (Slack et al., 2020; Sundararajan and Najmi, 2020).

The third local explanation method, BreakDown (BD) (Staniak and Biecek, 2018), provides FI values for a sample by computing the difference between the average model prediction and the average model prediction with the input features iteratively fixed to the specific feature values of the sample tested. Note that, for additive models, this approach yields the same results as SHAP, as noted in the original paper.

6.1.2 Global Methods. For the global FI methods, the implementation of (I. C. Covert et al., 2020a) is used. Since not many different methods for estimating global FIs are available, only two are used here. Shapley Effects (SE) (Owen, 2014) quantify the model's sensitivity to each feature. Related to SE, Shapley Additive Global importancE (SAGE) (I. C. Covert et al., 2020b) quantifies FI by estimating the predictive power each feature contributes. To account for feature interactions, the Shapley values are used.

6.2 Feature Importance Comparison

We compare the local and global FI methods described in Section 6.1 on the data sets listed in Section 6.3. As before, all experiments are performed on a workstation equipped with an Intel Core i9-14900K processor, NVIDIA RTX 4090 GPU with 24GB VRAM, and 96GB of system RAM. No significant differences between multiple training runs were observed. As such, the results will be presented for only one model per data set for a clearer presentation.

When comparing the different FIs, problems arise with their different interpretations, especially the so-called intercept. The intercept can be interpreted as "If no input feature provides any information, what is the expected model output?" or "How much of the input cannot be explained via the FI?". As such, it depends on the baseline used. Per default, SHAP and BD use the output of the black box model when all input features are at their expected value as a baseline, resulting in the same intercept for all samples. In contrast, LIME uses the bias of the local surrogate model, resulting in a sample-dependent intercept, while the intercept of RENTT-FI depends on the local input region in which the sample is located. More precisely, the intercept of RENTT is the bias of the local linear model, which accurately describes the behavior of the neural network within that region. When all FI values are summed together with this intercept, they reconstruct the final prediction of the model. Mean Abolute Error (MAE) could be used instead of Root Mean Square Error (RMSE) for comparing FI methods, which would

allow to compensate for this intercept offset, eliminating its influence on the comparison. However, doing so would defeat the purpose: The MAE would only compare the final predictions (feature scores plus intercept) rather than understanding how individual features contribute differently across methods. We choose to use RMSE without compensating for the intercept. This ensures a fair comparison because the intercept is an inherent part of how each FI method generates its explanations. Removing it would alter the fundamental nature of what we are evaluating and potentially mask important differences between methods.

Additionally, Krippendorff's α will be used to evaluate the FI-results (Krippendorff, 2004). This measure was proposed to evaluate the agreement between different psychometric tests, as in this context, results can be different depending on the specific test used and the person administering the tests. Krippendorff's α was previously used in the context of XAI, mainly to evaluate whether different metrics for XAI methods agree in their results (Fresz et al., 2024; Tomsett et al., 2019). It provides values in [-1, 1], where -1 denotes structural disagreement between given results, 0 denotes neither structural agreement nor disagreement and 1 denotes perfect agreement between results. Within this work, Krippendorff's α is used similar to the RMSE to evaluate whether the FI methods agree in their results. Two versions of α are used: Ordinal-scaled, measuring whether the ranking of the features based on the FI is the same for the different methods, and interval-scaled, measuring whether the methods agree in the magnitude of the allocated FI. We use the version of (Castro, 2017).

In addition, we provide a Relative Error (RE) to contextualize the mean differences against the typical magnitude of the FI values. As such, the RE between two methods is defined as the RMSE over all samples divided by the mean magnitude of both FI across all samples and features on the data set.

For local FI methods, the RMSE between each FI method pair will be computed samplewise and displayed as a violin plot, with the mean RMSE and standard deviation. This comparison will be done across entire data sets, to achieve a fair result. The Krippendorff's α results are also one value for α per data point, once again presented as a violin plot. Additionally, we provide the RMSE and RE as mean differences over all data points and include the standard deviation for both RMSE and RE as additional measures of variability. For the local methods, the RE between two methods is defined as the RMSE of a sample divided by the mean FI across all samples and features of both FI methods on this data set.

Since the global FI methods only provide one importance value per feature in the data set, the pairwise comparison of them via the RMSE is straightforward. Therefore, we provide the RMSE and RE, both with standard deviations, and also the Krippendorff's α results. The full results for both local and global methods can be found in Appendix E.

6.3 Data Sets and Neural Network Training

To be able to evaluate XAI methods, data sets with available ground truth explanations can be helpful. For this purpose, simple functions with known FIs are used, similar to previous publications with the same approach (Dzindolet et al., 1998; Ishigami and Homma, 1990; Liu et al., 2005). This allows to identify differences between the expected results of the XAI methods and their actual results. Note that evaluating XAI vs ground truth explanations may suffer from problems, especially if the model itself does not approximate the data perfectly and if the input space is high-dimensional, resulting in different models, and thus, explanations, being possible (e.g., for image data). As such, an explanation might conform to the model in question, i.e., be correct, but may not be the expected explanation for a given data set.

The functions used here should be simple enough to enable the NNs to (almost) perfectly approximate the expected outputs, as shown by their accuracy in Table 2 in Appendix B. The following simple functions are used:

- Absolute:

$$y_0 = |\mathbf{x}_0| \tag{24}$$

- Linear:

$$y_0 = 4 \cdot \mathbf{x}_{0,0} + \mathbf{x}_{0,1} + 0.001 \cdot \mathbf{x}_{0,2} \,, \tag{25}$$

where y_0 denotes the ground truth label and $\mathbf{x}_{0,k}$ denotes feature k of input sample $\mathbf{x}_0$.

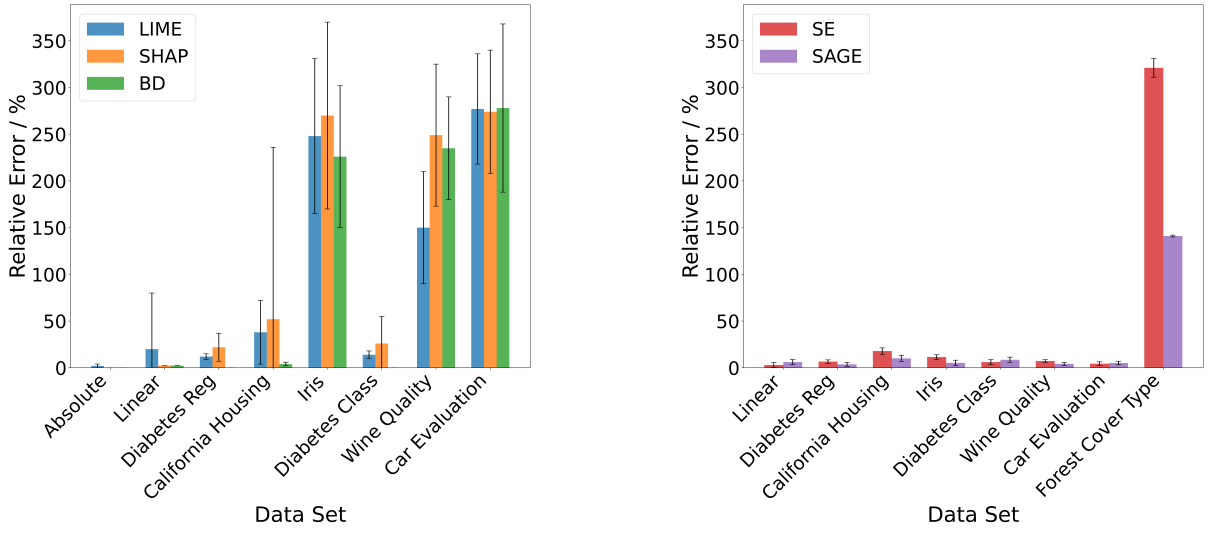
While simple functions with known ground truth explanations can serve as sanity checks for FI methods, they and their corresponding models might lack the complexity to show significant differences between the methods. Therefore, more complex data sets are used and the differences between methods quantitatively evaluated. For regression tasks, the data sets Diabetes Reg and California Housing are used, for classification Iris, a different Diabetes data set (here called Diabetes Class), Wine Quality, Car Evaluation and Forest Cover Type. A more detailed description of the data sets can be found in Appendix A. The resulting FI values of all methods directly depend on the scale of the input features. To achieve a fair comparison of the FI methods, the input values for all data sets are normalized to $[0, 1]$. Most of the previous data sets can be considered as "toy" data sets, all of them but the Forest Cover Type Data Set can be solved using a NN with two hidden layers with eight neurons in the first and four in the second hidden layer. For the Forest Cover Type data set, a network with 64 neurons in the first and 32 neurons in the second hidden layer was used. For model training, all data sets were split into 80% training data and 20% testing data. The model performance for the respective data sets is presented in Tables 2 and 3 in Appendix B.

6.4 Results

The experimental findings addressing the three key research questions are presented below. We systematically examine implementation-invariance across functionally equivalent models, evaluate the alignment of common FI methods with RENTT-FI as ground truth, and assess computational performance considerations.

6.4.1 Differences within Feature Importance Methods between Decision Tree and Neural Network. An important aspect of XAI methods is their implementation invariance (Nauta et al., 2023), as also described in Section 6. A part of implementation invariance is that the result for FI methods should be the same for functionally equivalent models. Here, RENTT provides two equivalent models: the original NN and the DT. As such, using the FI methods on both should provide the same results, or with regard to the sample-based nature of the methods at least very similar results. For the data sets described in Section 6.3, the comparison of the FIs between NN and DT for the local methods can be found in Figure 10(a) (Table 8 in Appendix D), the results for the global methods in Figure 10(b) (Table 9 in Appendix D).

For local FI methods (Figure 10(a) and Table 8), the results reveal a clear pattern across the two data set types. On regression tasks (Absolute, Linear, Diabetes Reg, California Housing), BD produces the closest results, with RE values typically below 5 %, seeming rather implementation-invariant for these tasks. However, on classification tasks, BD shows errors, with RE values frequently exceeding 200 % (226 ± 76 % for Iris, 225 ± 55 % for Wine Quality, and 278 ± 90 % for Car Evaluation). LIME and SHAP display similar patterns, but with worse RE values on regression tasks (typically $2 - 58$ %), with LIME performing slightly better than SHAP on most data sets. On classification tasks, both methods also exhibit high RE values ($14 - 277$ %), though SHAP often shows slightly higher errors than LIME. Notably, the Diabetes Class data set represents an exception where all methods show relatively low RE values (BD: $(5 \pm 2) \cdot 10^{-4}$ %, LIME: 14 ± 4 %, SHAP: 26 ± 29 %). This contrast between regression and classification performance suggests that the methodology of the FI methods may be particularly sensitive to the discrete decision boundary characteristic of classification networks. Another possible explanation may be that for classification tasks, only a limited amount of features receives weights $\gg 0$, resulting in a very low average FI across a data set and thus, high RE values.



(a) RE comparison for local FI methods.

(b) RE comparison for global FI methods.

Fig. 10. Comparative analysis of RE with standard deviation for local and global FI methods when applied to functionally equivalent NN and DT models across all data sets where FI calculations were possible. For the local methods on the Forest Cover Type data set, only RENTT-FI could be executed due to memory constraints.

For the global FI methods (Figure 10(b) and Table 9), both SE and SAGE have a slightly lower RE for the classification tasks (4.4 ± 2.2 % to 11.5 ± 2.5 % for SE and 4.2 ± 1.6 % to 8.6 ± 2.8 % for SAGE) than for the regression tasks (3.0 ± 2.7 % to 17.9 ± 3.4 % for SE and 3.7 ± 2.2 % to 10.1 ± 3.3 % for SAGE). The Forest Cover Type data set is an outlier with much higher REs (321 ± 10 % for SE and 141 ± 1 % for SAGE).

Overall, these results reveal significant implementation-dependence and underscore the lack of implementation-invariance of the current XAI methods, especially for classification tasks. Only for BD on regression tasks, consistently good results can be observed. The more consistent performance of global methods suggests they may be less implementation-dependent across diverse problems, though their relative errors remain non-negligible. Collectively, these findings show limitations in current XAI methods and underscore the value of RENTT-FI's ground truth FI as a benchmark for method evaluation.

6.4.2 Toy Example and Exploration of the Differences between LIME, SHAP, BD and RENTT-FI. As described before, the simple functions defined in Section 6.3 allow to explicitly compare the FI methods with the ground truth. For Equations (24) and (25), the small NNs approximate the corresponding functions almost perfectly, as shown via their Mean Squared Error (MSE) in Table 2 in Appendix B. As such, the explanation results for both functions can be compared to the ground truth parameters used in the functions themselves.

For the function given via Equation (24), one would expect the feature effect of $\mathbf{x}_0$ to be -1 for $\mathbf{x}_0 < 0$ and 1 for $\mathbf{x}_0 > 1$, representing $|\mathbf{x}_0|$. For this function, no meaningful global FI or Krippendorff's α can be computed, since only one feature is present. The NN approximation of the original function can be seen in Figure 11, together with the FIs. In the one-dimensional case, the FIs can be easily compensated for their intercept. As expected, the NN provides a near-perfect approximation of the original function. In addition, the FI weights of the corresponding linear models of RENTT-FI conform to the expected effective weights of -1 for $x < 0$ and 1 for $x > 0$. Qualitatively, the SHAP and BD values agree with this, when compensated for the intercept. Due to

SHAP and BD providing an intercept of 0.5 instead of 0 (Figure 12), these FIs have a constant offset compared to the FIs values of RENTT-FI and the ground truth values. The FI of LIME, based on the parameters used, results in plateaus (as shown in Figures 11 and 12), or in a hyperbolic tangent, exemplifying the parameter dependence of this method.

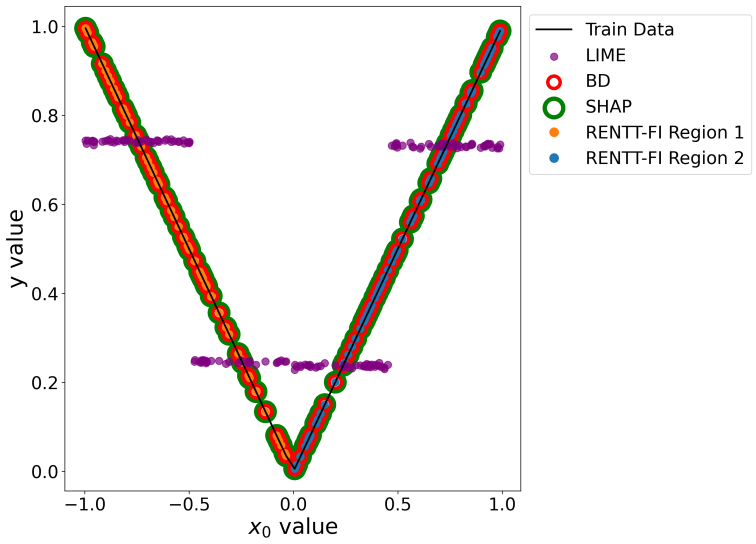


Fig. 11. Training data generated via Function (24) and prediction of the NN (in blue/orange) with the weights given by RENTT-FI. The linear model for $x < 0$ uses the effective weight -1 (orange), whereas the linear model for $x > 0$ uses the weight 1 (blue). Additionally, LIME- (red), BD- and SHAP-FIs (green) are shown, compensated for their intercept.

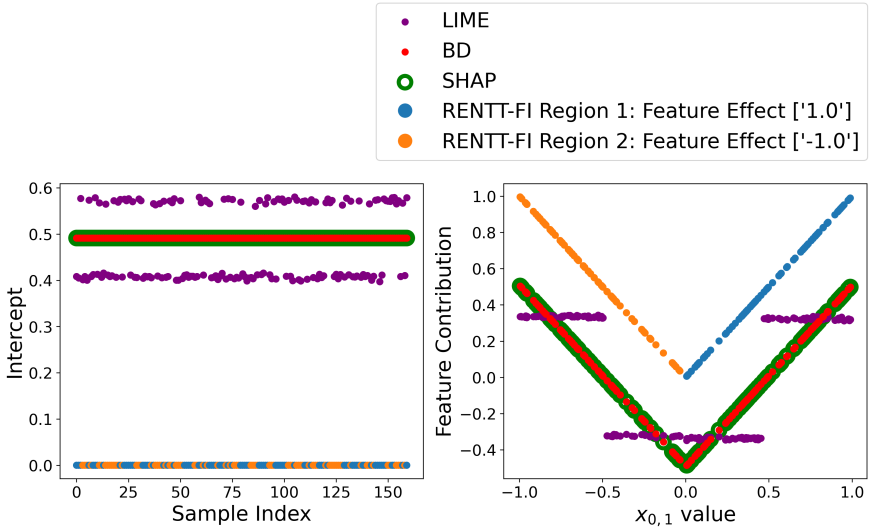


Fig. 12. Intercept (left) and FIs (right) for the different methods for data generated via Function (24).

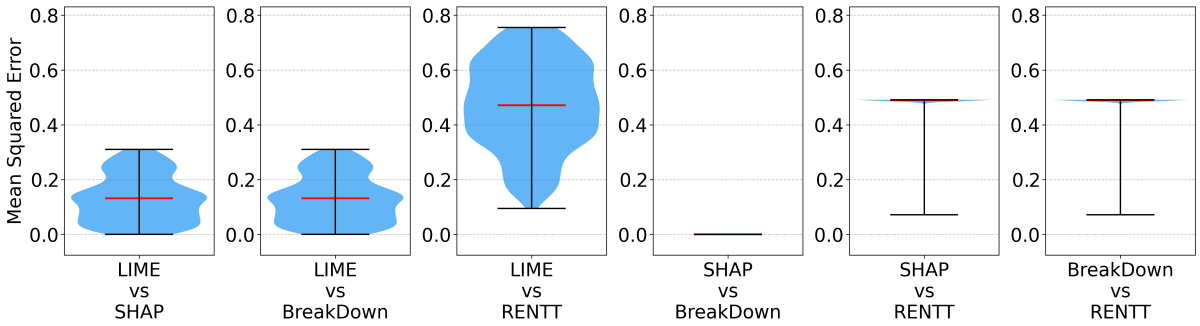
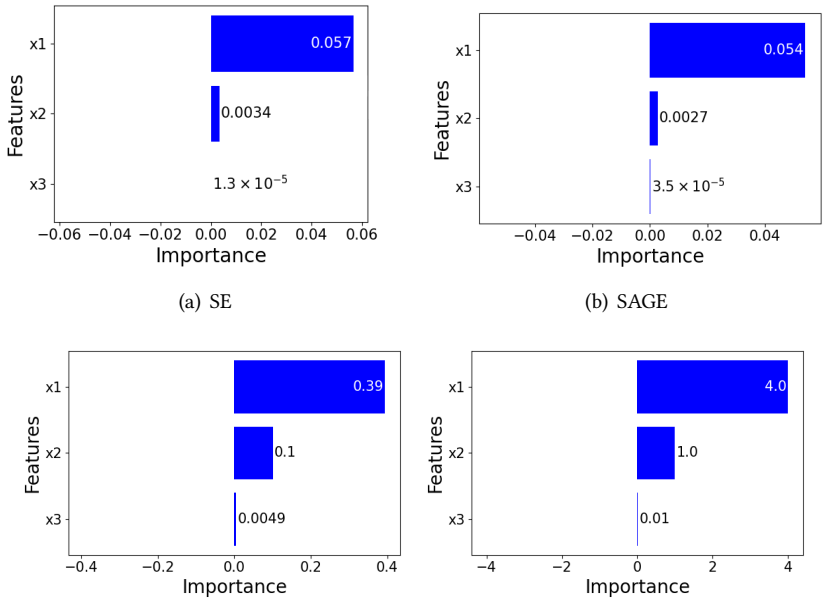


Fig. 13. Violin plot of the RMSE between the different FI methods for the data set generated via the function in Equation (24). Only the 95th percentile of samples is pictured.

Quantitatively, these differences can also be seen in Figure 13. While SHAP- and BD-FIs are the same within computational accuracy, the difference between these two methods and the RENTT-FI is quite low and only dominated by the difference in the intercept (RMSE of 0.5). The RMSE between LIME and SHAP and BD is substantially higher, ranging from 0 to 0.35 for the 95th percentile, with most samples having an RMSE below 0.2. The largest differences can be seen between RENTT-FI and LIME, with an RMSE between 0.1 and 0.78, once again showing the influence of the intercept.



(a) SE (b) SAGE

(c) RENTT-FI feature contribution (d) RENTT-FI feature effect

Fig. 14. Global FI by SE, SAGE and RENTT-FI on an NN trained on the data set generated via the function in Equation (25).

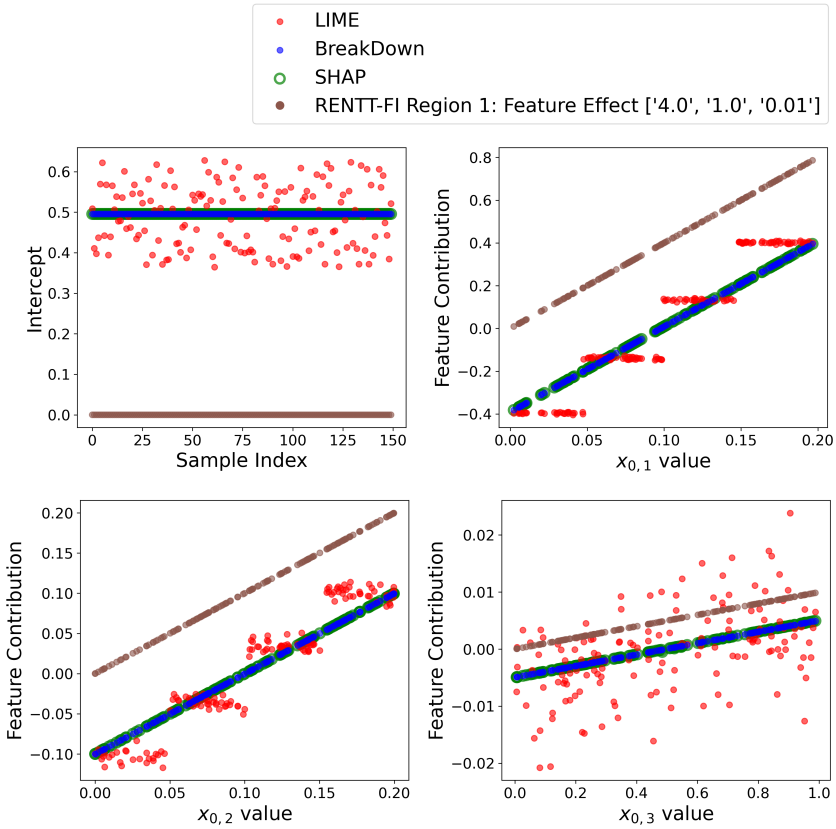


Fig. 15. Intercept (top left) and FIs for the data set generated via the function in Equation (25).

For the function given via Equation (25), local and global FI can be computed. Here, the importance of the input features should reconstruct the initial function with the FI of $\mathbf{x}_{0,0}$ being 4, the FI of $\mathbf{x}_{0,1}$ being 1 and the FI of $\mathbf{x}_{0,2}$ being 0.01. This result is expected for both local and global FI due to the linearity of the function used.

The global results for SE, SAGE, and RENTT-FI are shown in Figure 14. RENTT-FI produces feature effect and contribution, where the contribution equals the feature effect multiplied by the average feature value. In a more general case, this holds for all linear functions: Within a region, the contribution is the feature effect times the average feature value in this region. Both SE and SAGE converge on different outcomes. In these results, the influence of all features is widely underestimated (0.057 for $\mathbf{x}_{0,0}$ for SE and 0.054 for SAGE). RENTT-FI produces the expected feature effect of 4 for $\mathbf{x}_{0,0}$ and 0.39 for the feature contribution, being close to the average value of 0.1 times the effect with noise generated by the data sampling. Moreover, for SE and SAGE, the relationships among the different FIs are misrepresented, as the most important feature receives a disproportionately higher share of the FI compared to the function given via Equation (25). While providing similar values for Features $\mathbf{x}_{0,1}$ and $\mathbf{x}_{0,2}$, SE and SAGE do not agree exactly, with an FI of $3.4 \cdot 10^{-3}$ for SE and $2.7 \cdot 10^{-3}$ for SAGE for $\mathbf{x}_{0,1}$ and $1.3 \cdot 10^{-5}$ and $3.5 \cdot 10^{-5}$ for $\mathbf{x}_{0,2}$ respectively. For this data set, all global methods agree in their ranking of the important features (as shown in a value of 1.0 for Krippendorff's α in Table 1 for all method pairs), while the differences in value result in an interval-scaled α of 0.998 between SE and SAGE and 0.112 between RENTT-FI and SE and

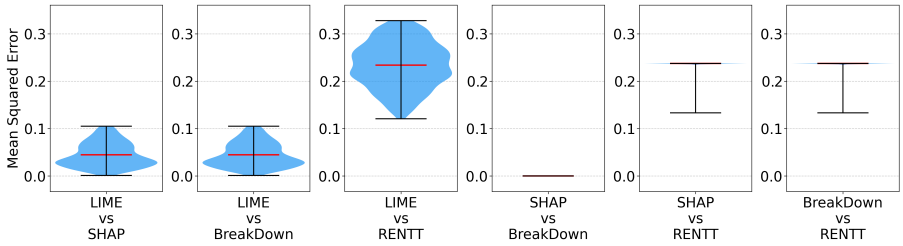
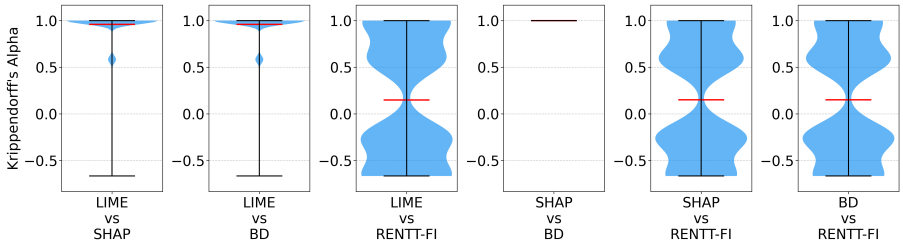
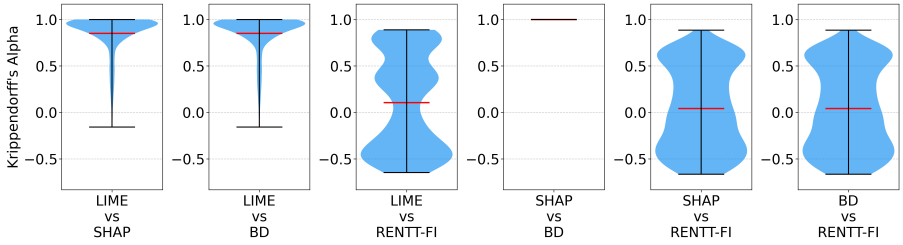


Fig. 16. Violin plot of the RMSE between the different FI methods for the data set generated via the function in Equation (25). Only the 95th percentile of samples is pictured.



(a) Ordinal (ranking)



(b) Interval (values)

Fig. 17. Violin plots of Krippendorff's α, calculated samplewise between the different FI methods for the data set generated via the function in Equation (25). Only the 95th percentile of samples is pictured.

0.104 between RENTT-FI and SAGE. Similarly, as shown in Figure 18(b), the RE between SE and SAGE is low (7.7 ± 2.7 %), while the one between RENTT-FI and the other methods is significantly higher (218 ± 0.4 % for SE and RENTT-FI, 221 ± 0.4 % for SE and RENTT-FI, Table 12 in Appendix E).

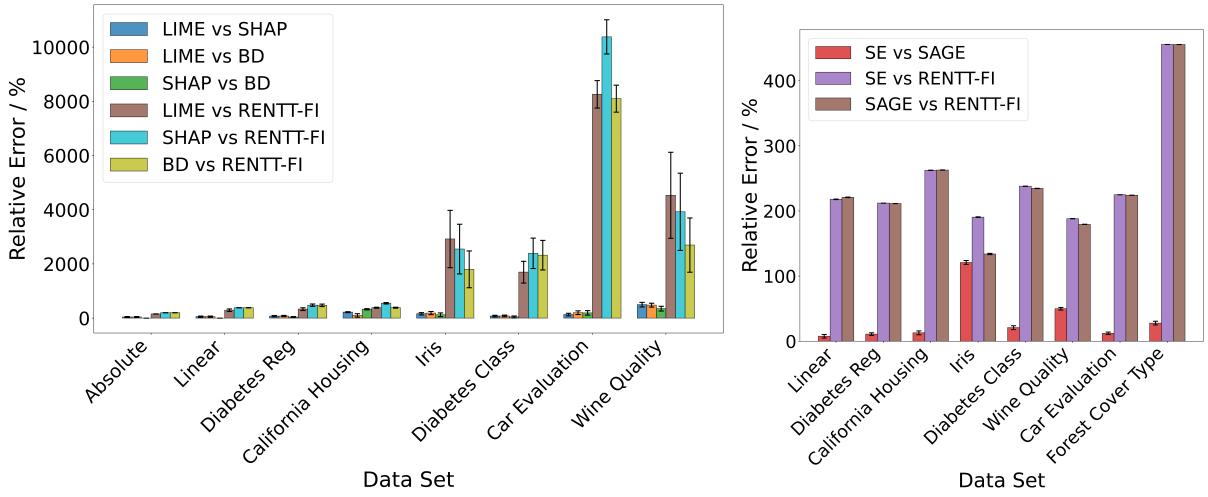
The local FI for the linear function given via Equation (25) yields largely the same findings as for the absolute function (Figure 16): SHAP, BD and RENTT-FI agree, with the offset in the RMSE between SHAP/BD and RENTT-FI created by the intercept (Figure 15). This offset dominates the RMSE. A similar trend is observed with LIME, as LIME and SHAP/BD are closer together with an RMSE below 0.11, whereas LIME and RENTT-FI yield RMSE-values between 0.11 and 0.34. For this data set, the differences between the evaluation methods become interesting: The RMSE (Figure 16), Krippendorff's α ordinal-scaled (Figure 17(a)) and interval-scaled (Figure 17(b)) show that LIME, SHAP and BD mostly agree in their ranking of the features, with the value agreement being

slightly worse. In contrast, RENTT-FI seems to provide quite different FI-rankings and values with neither clear structural agreement or disagreement. The explanation for this can be found once again within the intercept differing between the FI methods (Figure 15). The higher intercept of the FI methods leads to lower FI-values, since all methods roughly approximate the NN-output via the sum of the FIs and the intercept (Figure 15). This results in lower agreement in the interval-scaled α-values, while not being reflected in the ordinal-scaled α-values.

6.4.3 Differences between Existing Methods and RENTT-FI across Different Data Sets. To evaluate whether existing FI methods conform to ground truth explanations, SHAP, LIME and BD can be quantitatively compared to RENTT-FI across multiple data sets. Figure 18 as well as Tables 10 and 11 in Appendix E present the RMSE and RE between the different local methods for regression and classification tasks, respectively. For the local methods, it can be observed that for the regression tasks SHAP and BD produce the most similar results on the smallest data sets, with an RE of $(5.0 \pm 8.0) \cdot 10^{-6}$ % for the absolute function, 0.04 ± 1.0 % for the linear one and 37 ± 21% for Diabetes Reg. As data set complexity increases (California Housing), LIME and BD become the closest pair, with REs ranging around 100 %. Notably, RENTT-FI consistently produces substantially different values compared to all approximation methods, with REs between $(153 \pm 3) - (549 \pm 27)$ %.

In the classification tasks, all local methods generate significantly different results (lowest for SHAP/BD on Diabetes Class with 44 ± 37 %), but a consistent pattern emerges: LIME, SHAP, and BD are substantially more similar to each other than any of them is to RENTT-FI. The REs between all methods but RENTT-FI range from $(44 \pm 37) - (496 \pm 84)$ %, while errors between these methods and and RENTT-FI reach $(1,696 \pm 401) - (10,379 \pm 632)$ %. These differences are highly data set-dependent, with the Car Evaluation data set showing particularly large discrepancies for RENTT-FI (RE of $(8099 \pm 497) - (10,379 \pm 632)$ %).

For all used data sets, visualizations of Krippendorff's α and the RMSE, similar to Figures 16 and 17, can be found in Appendix E. They mainly consolidate the previous findings of the agreement between FI-methods being data set dependent with a trend towards SHAP and BD agreeing the most.



(a) Local FI methods.

(b) Global FI methods.

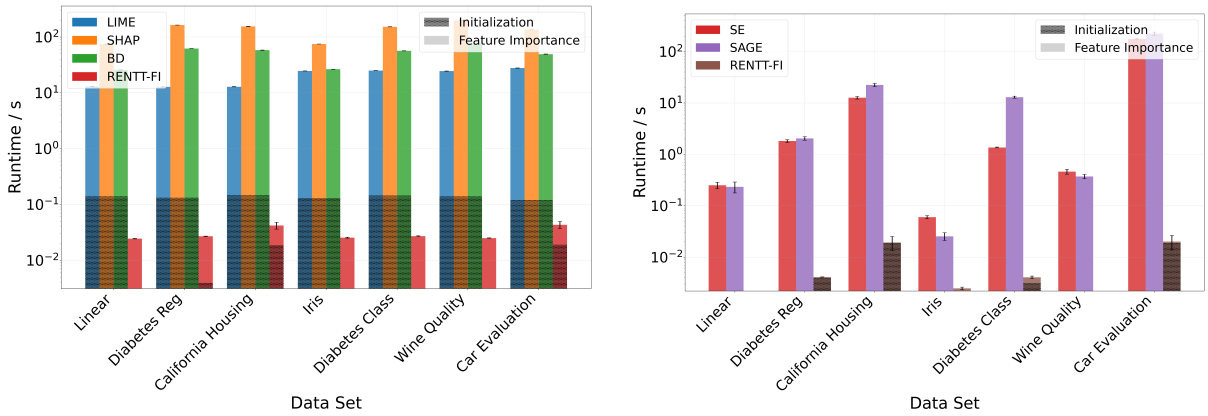
Fig. 18. Comparative analysis of RE with standard deviation for local and global FI methods across all data sets where FI calculations were possible.

For the global FI methods, SE and SAGE produce values that are orders of magnitude smaller than RENTT-FI (typically 0.001-0.03 vs 0.1-7.5 in Tables 4 and 5). Visualizations of the FI values, similar to Figure 14, can be found in Appendix E, in Figures 22 and 25 for the regression tasks and Figures 28, 31, 34, 37, 40 and 41 for the classification tasks. While SE and SAGE show reasonable to high agreement in both sign and relative magnitude, meaningful systematic (dis)agreement ($|\alpha| > 0.7$) with RENTT-FI values (Table 1) depends on the data set in question. For the Linear, Diabetes Reg, Iris, Diabetes Class and Wine Quality, all methods agree in their feature ranking (ordinal $|\alpha| > 0.75$), but not in their FI values (interval $|\alpha| < 0.6$). The RE in Figure 18(b) follows the same pattern of higher differences between RENTT-FI and SAGE/SE. For the global methods, the RE does not consistently increase or decrease between classification and regression tasks. For the most complex data set, the Forest Cover Type, the RE is significantly higher (455.3 ± 0.0006% between RENTT-FI and SE/SAGE) than for the other data sets (at most 262.9 ± 0.2% between RENTT-FI and SE/SAGE for California Housing).

These results demonstrate that while existing approximation methods show internal consistency patterns (with BD and SHAP aligning on small data sets), all diverge substantially from the ground truth explanations provided by RENTT-FI. This divergence seems to increase with data set complexity.

Table 1. Pairwise Krippendorff's α reliability coefficients for global FI methods.

Data Set	**Comparison**	α **(Ordinal)**	α **(Interval)**
Linear	SE vs SAGE	1.000	0.998
	SE vs RENTT-FI	1.000	0.112
	SAGE vs RENTT-FI	1.000	0.104
Diabetes Reg	SE vs SAGE	0.988	0.995
	SE vs RENTT-FI	0.878	0.093
	SAGE vs RENTT-FI	0.855	0.097
California Housing	SE vs SAGE	0.978	0.996
	SE vs RENTT-FI	0.196	0.022
	SAGE vs RENTT-FI	0.107	0.022
Iris	SE vs SAGE	1.000	0.652
	SE vs RENTT-FI	0.825	0.332
	SAGE vs RENTT-FI	0.825	0.592
Diabetes Class	SE vs SAGE	0.978	0.989
	SE vs RENTT-FI	0.978	0.082
	SAGE vs RENTT-FI	0.955	0.098
Wine Quality	SE vs SAGE	0.971	0.891
	SE vs RENTT-FI	0.829	-0.503
	SAGE vs RENTT-FI	0.769	-0.452
Car Evaluation	SE vs SAGE	1.000	0.987
	SE vs RENTT-FI	0.005	0.037
	SAGE vs RENTT-FI	0.005	0.039
Forest Cover Type	SE vs SAGE	0.965	0.994
	SE vs RENTT-FI	-0.410	0.010
	SAGE vs RENTT-FI	-0.437	0.010



(a) Local FI methods comparison.

(b) Global FI methods comparison.

Fig. 19. Runtime comparison of FI methods. The hatched area indicates the initialization of the method—explainer initialization or in case of RENTT-FI the transformation of the NN into DT.

6.4.4 Performance Comparison between existing Methods and RENTT-FI. In this section, we explore the effectiveness of our FI calculations, comparing its runtime against established methods such as LIME, SHAP, and others. By providing both global and local FI insights, we aim to illustrate how RENTT-FI enhances interpretability without sacrificing performance.

The runtime comparison presented in Figure 19 demonstrates the superior computational efficiency of RENTT-FI across all evaluated data sets and both local and global FI methods. As illustrated in Figure 19(a), RENTT-FI consistently achieves the fastest execution times for local explanations, producing these faster by several orders of magnitude compared to LIME, SHAP and BD. RENTT-FI achieves runtime improvements of approximately 300-500× faster than LIME, 1,800-7,500× faster than SHAP, and 600-2,900× faster than BD. Remarkably, RENTT-FI maintains sub-millisecond execution times (0.02-0.04 seconds) regardless of data set complexity, while traditional methods require tens to hundreds of seconds.

Similarly, Figure 19(b) reveals that RENTT-FI maintains its computational advantage for global FI calculations, substantially outperforming SE and SAGE methods. The consistent performance across diverse data sets validates the scalability and robustness of our approach. The detailed numerical results supporting these visualizations are provided in Appendix F (Tables 13 and 14).

6.5 Discussion of the Results

This chapter presented a comprehensive evaluation of FI methods through systematic experiments comparing RENTT-FI with established explanation methods. The analysis revealed limitations in current XAI approaches that have significant implications for practical applications.

Most methods failed the implementation-invariance test, producing significantly different FI values for functionally equivalent models (NN and DT), especially in classification tasks (Section 6.4.1). These findings demonstrate that while existing methods exhibit internal consistency patterns under specific conditions, they systematically fail to capture the true FI as determined by the network's mathematical structure (Section 6.4.2). The divergence from ground truth explanations reveals fundamental limitations in current XAI approaches that cannot be resolved through methodological refinements alone, highlighting the critical need for mathematically grounded explanation techniques like RENTT-FI.

For local methods, it can be observed that SHAP and BD showed strong agreement on smaller data sets (Absolute, Linear, Diabetes Reg), LIME and BD became the most similar methods on larger data sets (California Housing), but with rather high RE (Section 6.4.3). All methods diverged substantially from RENTT-FI, with REs values consistently exceeding 150 % across data sets and reaching up to 10, 400 % in classification tasks. This divergence increased with data set complexity. For the global explanation methods, SE and SAGE produced values orders of magnitude smaller than RENTT-FI with no meaningful overlap or common structure in actual importance values.

One important consideration is the computational cost distribution for different explanation methods (Section 6.4.4). For RENTT-FI, the transformation process represents the most computationally expensive component, depending on network size with minimal cost for small networks. Subsequently, calculating Feature Effects for each sample and determining feature contributions is relatively inexpensive, with costs scaling primarily with the number of features. As such, the relative computational burden of RENTT-FI decreases when explaining many samples of a data set. In contrast, methods such as LIME, SHAP, and BD require an initialization phase where the explanation infrastructure is prepared—including creating model wrappers and storing reference data for baseline comparisons. While this initialization cost varies by implementation, the subsequent FI calculations represent the more computationally expensive phase for these methods. Our runtime analysis demonstrate a significant improvement in FI methodology: RENTT-FI reduces the computational effort while maintaining correctness. RENTT-FI provides explanability insights with lower computational overhead compared to existing approaches.

7 Discussion and Limitations

In this section, we discuss the current limitations of our methodology and outline promising directions for future development.

The transformation's exactness of RENTT fundamentally depends on Assumption 1, which restricts the approach to piecewise linear activation functions when 100 % correctness is an essential requirement. Some commonly used activation functions in modern deep learning, such as sigmoid or tanh, cannot be directly transformed without approximation. While piecewise linear approximation is possible, this creates a trade-off between correctness and computational cost—more linear segments improve approximation quality but increase the tree size exponentially.

Memory complexity represents the most critical bottleneck for practical applications of RENTT, especially when dealing with larger network architectures. Despite significant improvements over existing methods like ECDT—with memory scaling at $O(x^{1.7})$ to $O(x^{1.9})$ versus $O(x^{8.6})$ to $O(x^{8.7})$—absolute memory requirements remain substantial. Networks with 10^4 hidden neurons require 31-630 GB of RAM, limiting applicability on standard hardware. The choice of activation functions critically impacts memory efficiency, as even inherently piecewise linear functions with more than two linear regions increase computational complexity. According to Equation (8), the number of decision tree nodes scales with the number of linear regions r_i of the activation function in the layer i (as specified in Assumption 3.1.3) as an exponent. For optimal performance, ReLU-family activation functions (ReLU, Leaky ReLU, etc.) should be used, which have only two linear regions. This approach fortunately aligns with common practices in modern deep learning where ReLU and its variants dominate. When other activation functions are necessary, the number of approximation segments should be balanced against available computational resources carefully. To overcome this limitation while enhancing runtime efficiency, future work should extend beyond the pruning techniques used in our approach by integrating parallelization strategies for distributed computation, optimizing data structures for memory efficiency, and developing enhanced model serialization methods for effective resource management. By integrating these improvements, the transformation framework would be capable of processing larger datasets and more complex models with substantially reduced execution time and memory overhead, thereby achieving better scalability.

We noticed for the FI calculations that RENTT-FI has lower computational costs on smaller to medium-sized networks compared to existing approaches. An important research question emerges regarding the NN size and sample number threshold beyond which RENTT-FI's transformation costs surpass those of conventional FI methods to determine the optimal NN size ranges and sample numbers for RENTT-FI deployment.

To reduce runtime and memory complexity, we use ante-hoc pruning of the DTs. This pruning removes activation patterns that are not used throughout the relevant data set (here training and testing data). For these data sets, the transformation with pruning is exact, resulting in the exact same predictions for all data samples. A key limitation arises when unseen data samples during deployment trigger activation patterns that were inactive during training. These patterns, having been pruned from the DT, create missing pathways that prevent proper inference. While such pathways could be dynamically added to the DT, this would necessitate extracting weights from the original NN, which must therefore also be retained in memory. To prevent such issues, a more complex and less efficient pruning could be used, e.g., by only removing activation patterns that are impossible to reach. Such a pruning approach would allow for perfect exactness even for unseen data samples, but would increase memory and runtime complexity significantly.

Currently, the implementation of this transformation is limited to FcNNs. Expanding the implementation to include CNNs and RNNs as described in Remark 8 will extend the applicability of the approach to a wider range of neural network architectures, ensuring that diverse models can benefit from enhanced interpretability and explainability.

RENTT transforms NNs into equivalent DTs. Ideally, this would result in directly interpretable models. But due to the resulting decision trees being multivariate, they are too complex to understand. As such, further simplifications are necessary to use these trees as explanations. We chose to represent the DTs via FI, as this was computationally efficient, a common explanation type and did not result in a loss of correctness. Further explanation types enabled by the DTs (see Section 5) could further improve the interpretability of NNs.

As commented on throughout this paper, some problems arise when comparing different FI methods and XAI methods in general. While RENTT-FI separates feature contribution and feature effect clearly, this distinction is often not as clear for other approaches. Especially for the global methods, it is not specified whether their results should be interpreted as effect or contribution. We chose to compare the feature contribution of RENTT-FI to SE and SAGE, to maintain consistency with the local methods. Depending on the specific question a user is interested in, the feature effect might provide a clearer interpretation for the global case.

As an extension to the existing work of evaluating XAI methods via RENTT-FI, metrics for the correctness of XAI could also be evaluated using RENTT-FI. Previously, such metrics have been shown to disagree systematically (Tomsett et al., 2019). With RENTT-FI, it can be assessed which correctness metric best reflects the ground truth FI. In cases where RENTT-FI is not feasible due to runtime or memory complexity, the metric that best agreed with RENTT-FI for simpler models or data sets could be used to choose between feasible XAI methods.

While theoretical benefits for XAI methods are desirable, explanations are directed towards humans. As such, their main goals depend on downstream impact, e.g, by empowering humans to spot incorrect model decisions or provide new modes of scientific discovery. In general, many XAI papers lack user studies to evaluate their proposed methods (Suh et al., 2025). For RENTT-FI, a user study could provide interesting insights into how the created explanations are perceived and whether their theoretical soundness create benefits for downstream goals.

The local linear models underlying our transformation represent an underutilized resource for deeper insights into neural network behavior. While our current applications focus primarily on FI calculation, these models could enable broader explainability applications that remain unexplored, such as counterfactual analysis to generate "what-if" scenarios. Further exploration of the local linear models used for this transformation could provide deeper insights into the decision-making processes of neural networks. By examining these models in detail, we can enhance explainability, offering more granular perspectives on how decisions are made within the

network. Additionally, local linear models have potential applications beyond XAI, such as improving robustness. Understanding decision boundaries and gradients near these boundaries could be crucial for developing models that are resistant to adversarial attacks and other perturbations.

Within NNs, usually different activation patterns are not tracked. Consequently, it is difficult to specify whether a model is sufficiently trained or whether all input regions contain adequate data samples to ensure robust decisions. The RENTT-DTs automatically track the number of data samples corresponding to each activation pattern. This enables the identification of underutilized and thus, possibly undertrained activation patterns, for which more data samples would improve decision robustness. In the same vein, the pruning method applied to the decision trees could be utilized: If pruning yields a simpler representation of the NNs, this representation could be used for inference to decrease runtime and memory complexity. Our experiments reveal that non-linear neural networks sometimes defaulted to a linear representation in the DTs, indicating that the used models might have been too complex for the data set in question.

8 Conclusion

The conversion of NNs into DTs offers a practical solution for improving model interpretability and explanability. By transforming NNs including FcNNs, CNNs with convolution and pooling layers and RNNs with general activation functions into multivariate DTs, this approach maintains the original network's functionalities while making the decision-making process more understandable. This allows to circumvent the often stated tradeoff between accuracy and interpretability. The linearization of NNs, especially with piecewise linear activation functions, simplifies network operations into linear transformations, enabling their mapping to DT structures. To address scalability challenges, efficient implementation techniques and ad hoc pruning methods are used to manage large networks. These strategies reduce memory consumption and runtime, making the transformation suitable for large-scale applications. This method effectively enhances explainability in NN architectures by incorporating feature importance calculations globally, regionally, and locally. This provides valuable insights into the influence of input features on model decisions, aiding in understanding model behavior. This allows for downstream applications such as ensuring fairness of models by showing the influence of sensitive attributes such as age or gender. In conclusion, transforming NNs into DTs allows performance and interpretability, making it easier to analyze and understand complex models. This method contributes to the goal of making NNs more accessible and reliable for diverse applications, providing essential tools for diagnostics and feature importance analysis. As NNs are increasingly utilized in critical domains, the ability to interpret and trust their decisions is vital for their effective deployment.

Acknowledgments

This paper is funded in parts by the German Research Foundation (DFG) Priority Programme (SPP 2422) "Data-Driven Process Modelling in Forming Technology" (Subproject "Transparent AI-supported process modeling in drop forging" 520195047)

References

J. Adebayo, J. Gilmer, M. Muelly, I. Goodfellow, M. Hardt, and B. Kim. 2018. "Sanity checks for saliency maps." In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (NIPS'18). Curran Associates Inc., Montréal, Canada, 9525–9536.

C. Aytekin. 2022. *Neural networks are decision trees*. https://arxiv.org/abs/2210.05189. arXiv preprint arXiv:2210.05189. (2022).

R. Balestriero and R. Baraniuk. 2020. "Mad Max: Affine Spline Insights Into Deep Learning." In: *Proceedings of the IEEE*, 1–24. doi:10.1109/JPROC.2020.3042100.

H. Baniecki, W. Kretowicz, P. Piatyszek, J. Wisniewski, and P. Biecek. 2021. "dalex: Responsible Machine Learning with Interactive Explainability and Fairness in Python." *Journal of Machine Learning Research*, 22, 214, 1–7. http://jmlr.org/papers/v22/20-1473.html.

D. Bhaumik, D. Dey, and S. Kayal. 2022. "A Framework for Auditing Multilevel Models using Explainability Methods." *International Conference on AI Research*, 4, 12–21. doi:10.34190/icair.4.1.874.

J. Blackard. 1998. *Covertype*. https://archive.ics.uci.edu/ml/datasets/covertype. UCI Machine Learning Repository, DOI: 10.24432/C50K5N. (1998).

M. Bohanec. 1988. *Car Evaluation*. https://archive.ics.uci.edu/ml/datasets/car. UCI Machine Learning Repository, DOI: 10.24432/C5JP48. (1988).

G. van den Broeck, A. Lykov, M. Schleich, and D. Suciu. 2022. "On the Tractability of SHAP Explanations." *Journal of Artificial Intelligence Research*, 74, 851–886. doi:10.1613/jair.1.13283.

S. Castro. 2017. *Fast Krippendorff: Fast computation of Krippendorff's alpha agreement measure*. https://github.com/pln-fing-udelar/fast-krippendorff. GitHub repository. (2017).

L. Chu, X. Hu, J. Hu, L. Wang, and J. Pei. 2018. "Exact and Consistent Interpretation for Piecewise Linear Neural Networks." In: *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, New York, NY, USA, 1244–1253. doi:10.1145/3219819.3220063.

I. Covert, S. Lundberg, and S.-I. Lee. 2021. "Explaining by Removing: A Unified Framework for Model Explanation." *Journal of Machine Learning Research*, 22, 209, 1–90. http://jmlr.org/papers/v22/20-1316.html.

I. C. Covert, S. Lundberg, and S.-I. Lee. 2020a. *SAGE*. https://github.com/iancovert/sage. (2020).

I. C. Covert, S. Lundberg, and S.-I. Lee. 2020b. "Understanding global feature contributions with additive importance measures." In: *Proceedings of the 34th International Conference on Neural Information Processing Systems* (NIPS '20) Article 1444, 12 pages.

M. T. Dzindolet, S. A. Peterson, R. A. Pomranky, L. G. Pierce, and H. P. Beck. 1998. "On quasi-Monte Carlo integrations." *Mathematics and Computers in Simulation*, 47, 2, 103–112. doi:10.1016/S0378-4754(98)00096-2.

M. T. Dzindolet, S. A. Peterson, R. A. Pomranky, L. G. Pierce, and H. P. Beck. 2003. "The role of trust in automation reliance." *International Journal of Human-Computer Studies*, 58, 6, 697–718. doi:10.1016/S1071-5819(03)00038-7.

L. C. Evans. 2018. *Measure theory and fine properties of functions*. Routledge.

B. Fresz, L. Lörcher, and M. Huber. 2024. "Classification Metrics for Image Explanations: Towards Building Reliable XAI-Evaluations." In: *Proceedings of the 2024 ACM Conference on Fairness, Accountability, and Transparency*. Association for Computing Machinery, New York, NY, USA, 1–19. doi:10.1145/3630106.3658537.

I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep Learning*. http://www.deeplearningbook.org. MIT Press.

T. Ishigami and T. Homma. 1990. "An importance quantification technique in uncertainty analysis for computer models." In: *[1990] Proceedings. First International Symposium on Uncertainty Modeling and Analysis*, 398–403. doi:10.1109/ISUMA.1990.151285.

J. S. Kim, G. Plumb, and A. Talwalkar. 2022. *Sanity Simulations for Saliency Methods*. (2022). https://arxiv.org/abs/2105.06506 arXiv: 2105.06506 (cs.LG).

P.-J. Kindermans, S. Hooker, J. Adebayo, M. Alber, K. T. Schütt, S. Dähne, D. Erhan, and B. Kim. 2017. *The (Un)reliability of saliency methods*. (2017). https://arxiv.org/abs/1711.00867 arXiv: 1711.00867 (stat.ML).

K. Krippendorff. 2004. "Reliability in Content Analysis." *Human Communication Research*, 30, 3, 411–433. doi:10.1111/j.1468-2958.2004.tb00738.x.

I. E. Kumar, S. Venkatasubramanian, C. Scheidegger, and S. Friedler. July 2020. "Problems with Shapley-value-based explanations as feature importance measures." In: *Proceedings of the 37th International Conference on Machine Learning* (Proceedings of Machine Learning Research). Ed. by H. D. III and A. Singh. Vol. 119. PMLR, (July 2020), 5491–5500. https://proceedings.mlr.press/v119/kumar20e.html.

H. Liu, W. Chen, and A. Sudjianto. 2005. "Relative Entropy Based Method for Probabilistic Sensitivity Analysis in Engineering Design." *Journal of Mechanical Design*, 128, 2, 326–336. doi:10.1115/1.2159025.

S. M. Lundberg and S.-I. Lee. 2017. "A Unified Approach to Interpreting Model Predictions." In: *Advances in Neural Information Processing Systems*. Vol. 30. https://proceedings.neurips.cc/paper_files/paper/2017/file/8a20a8621978632d76c43dfd28b67767-Paper.pdf.

S. Ma, Y. Lei, X. Wang, C. Zheng, C. Shi, M. Yin, and X. Ma. 2023. *Who Should I Trust: AI or Myself? Leveraging Human and AI Correctness Likelihood to Promote Appropriate Trust in AI-Assisted Decision-Making*. (2023). https://arxiv.org/abs/2301.05809 arXiv: 2301.05809 (cs.HC).

F. A. Mala and R. Ali. 2022. "The Big-O of Mathematics and Computer Science." *Journal of Applied Mathematics and Computation*, 6, 1, 1–3. doi:10.26855/jamc.2022.03.001.

S. Mangalathu, S.-H. Hwang, and J.-S. Jeon. 2020. "Failure mode and effects analysis of RC members based on machine-learning-based SHapley Additive exPlanations (SHAP) approach." *Engineering Structures*, 219, 110927. doi:10.1016/j.engstruct.2020.110927.

L. Merrick and A. Taly. 2020. "The Explanation Game: Explaining Machine Learning Models Using Shapley Values." In: *Machine Learning and Knowledge Extraction*. Ed. by A. Holzinger, P. Kieseberg, A. M. Tjoa, and E. Weippl. Springer International Publishing, Cham, 17–38. ISBN: 978-3-030-57321-8.

C. Molnar, G. König, J. Herbinger, T. Freiesleben, S. Dandl, C. A. Scholbeck, G. Casalicchio, M. Grosse-Wentrup, and B. Bischl. 2022. "General Pitfalls of Model-Agnostic Interpretation Methods for Machine Learning Models." In: *xxAI - Beyond Explainable AI: International Workshop, Held in Conjunction with ICML 2020, July 18, 2020, Vienna, Austria, Revised and Extended Papers*. Springer International Publishing, Cham, 39–68. ISBN: 978-3-031-04083-2. doi:10.1007/978-3-031-04083-2_4.

E. Mosca, F. Szigeti, S. Tragianni, D. Gallagher, and G. Groh. 2022. "SHAP-Based Explanation Methods: A Review for NLP Interpretability." In: *Proceedings of the 29th International Conference on Computational Linguistics*, 4593–4603. https://aclanthology.org/2022.coling-1.406.pdf.

M. Nauta, J. Trienes, S. Pathak, E. Nguyen, M. Peters, Y. Schmitt, J. Schlötterer, M. van Keulen, and C. Seifert. 2023. "From Anecdotal Evidence to Quantitative Evaluation Methods: A Systematic Review on Evaluating Explainable AI." *ACM Comput. Surv.*, 55, 13s, Article 295, 42 pages. doi:10.1145/3583558.

D. T. Nguyen, K. E. Kasmarik, and H. A. Abbass. 2020. *Towards Interpretable ANNs: An Exact Transformation to Multi-Class Multivariate Decision Trees*. https://arxiv.org/abs/2003.04675. arXiv preprint arXiv:2003.04675. (2020).

A. B. Owen. 2014. "Sobol' Indices and Shapley Value." *SIAM/ASA Journal on Uncertainty Quantification*, 2, 1, 245–251. doi:10.1137/130936233.

Pinecone. 2023. *Image Search with ImageNet*. https://www.pinecone.io/learn/series/image-search/imagenet/. Accessed: 2025-05-26. (2023).

M. Raghu, B. Poole, J. Kleinberg, S. Ganguli, and J. Sohl-Dickstein. 2017. "On the Expressive Power of Deep Neural Networks." In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. PMLR, 2847–2854. https://proceedings.mlr.press/v70/raghu17a.html.

M. T. Ribeiro, S. Singh, and C. 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* (KDD '16). Association for Computing Machinery, New York, NY, USA, 1135–1144. doi:10.1145/2939672.2939778.

C. Rudin. 2019. "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead." *Nature Machine Intelligence*, 1, 5, 206–215. doi:10.1038/s42256-019-0048-x.

D. E. Rumelhart, G. E. Hinton, and R. J. Williams. 1986. "Learning representations by back-propagating errors." *Nature*, 323, 533–536. https://www.nature.com/articles/323533a0.

M. Schröder, A. Zamanian, and N. Ahmidi. 2023a. "Post-hoc Saliency Methods Fail to Capture Latent Feature Importance in Time Series Data." In: *Trustworthy Machine Learning for Healthcare: First International Workshop, TML4H 2023, Virtual Event, May 4, 2023, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 106–121. doi:10.1007/978-3-031-39539-0_10.

M. Schröder, A. Zamanian, and N. Ahmidi. 2023b. "What about the Latent Space? The Need for Latent Feature Saliency Detection in Deep Time Series Classification." *Machine Learning and Knowledge Extraction*, 5, 2, 539–559. doi:10.3390/make5020032.

D. Slack, S. Hilgard, E. Jia, S. Singh, and H. Lakkaraju. 2020. "Fooling lime and shap: Adversarial attacks on post hoc explanation methods." In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society* (AIES '20). Ed. by A. Markham, J. Powles, T. Walsh, and A. L. Washington. ACM, New York, NY, USA, 180–186.

J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. 1988. "Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus." *Proceedings of the Annual Symposium on Computer Applications in Medical Care*, 10, 261–265.

M. Staniak and P. Biecek. 2018. "Explanations of Model Predictions with live and breakDown Packages." *The R Journal*, 10, 395–409, 2.

A. Sudjianto, W. Knauth, R. Singh, Z. Yang, and A. Zhang. 2020. *Unwrapping The Black Box of Deep ReLU Networks: Interpretability, Diagnostics, and Simplification*. https://arxiv.org/abs/2011.04041. (2020). doi:10.48550/ARXIV.2011.04041.

A. Suh, I. Hurley, N. Smith, and H. C. Siu. 2025. *Fewer Than 1% of Explainable AI Papers Validate Explainability with Humans*. (2025). https://arxiv.org/abs/2503.16507 arXiv: 2503.16507 [cs.HC].

M. Sundararajan and A. Najmi. 2020. "The many Shapley values for model explanation." In: *Proceedings of the 37th International Conference on Machine Learning* (ICML'20) Article 859. JMLR.org, 10 pages.

R. Tomsett, D. Harborne, S. Chakraborty, P. Gurram, and A. Preece. 2019. *Sanity Checks for Saliency Metrics*. (2019). arXiv: 1912.01451 [cs.LG].

J. Tritscher, M. Wolf, A. Hotho, and D. Schlör. 2023. "Evaluating Feature Relevance XAI in Network Intrusion Detection." In: *Explainable Artificial Intelligence. First World Conference, xAI 2023, Lisbon, Portugal, July 26–28, 2023, Proceedings, Part I*. Communications in Computer and Information Science. Vol. 1901. Ed. by L. Longo. Springer Nature Switzerland, Cham, 483–497. doi:10.1007/978-3-031-44064-9_25.

Z. Wang, R. Balestriero, and R. G. Baraniuk. 2019. "A max-affine spline perspective of recurrent Neural Networks." In: *ICLR 2019*.

G. Yona and D. Greenfeld. 2021. "Revisiting Sanity Checks for Saliency Maps." *CoRR*, abs/2110.14297. https://arxiv.org/abs/2110.14297 arXiv: 2110.14297.

Y. Zhang, Q. V. Liao, and R. K. E. Bellamy. 2020. "Effect of confidence and explanation on accuracy and trust calibration in AI-assisted decision making." In: *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency* (FAT* '20). ACM, 295–305. doi:10.1145/3351095.3372852.

A Data Sets

For the different experiments, we used a set of common data sets. Where possible and not stated otherwise, the data sets from scikit-learn were used. For regression tasks, we used these data sets:

- The Diabetes Reg data set contains data collected from 442 patients, with each patient having ten features related to diabetes progression one year after baseline. The features include Age, Sex, Body Mass Index, Average Blood Pressure and six blood serum measurements. From these inputs, the disease progression within a year should be predicted as a continuous value.
- The California Housing Data Set contains information about housing prices in California, based on data collected from various block groups in the state. Overall, 20,640 samples with eight features are present, with the features Median Income (MedInc), House Age, Latitude, Longitude, Population (all related to one block group), Average Number of Rooms (AveRooms), Average Number of Bedrooms (AveBedrms), Average Number of Occupants (AveOccup). From these, the median house value within a block group should be predicted.

Since RENTT-FI can also be used for classification tasks, further data sets were included for evaluation:

- The Iris Data Set consists of 150 samples of iris flowers, each belonging to one of three species: Iris setosa, Iris versicolor, and Iris virginica. The data set includes four features measured for each sample: sepal length, sepal width, petal length, and petal width, all measured in centimeters. Based on these features, the species of flower should be predicted.
- A number of different Diabetes data sets exist, also for classification. The one used here (called Diabetes Class) contains 768 samples with eight features such as Number of Pregnancies, Plasma Glucose Concentration, Body Mass Index and Age. The task is a binary classification of whether a patient will be tested positive for diabetes. The version of kaggle is used[2], which is a subset of an older data set created by (Smith et al., 1988).
- The Wine Quality Data Set consists of 178 samples of wine, each described by 13 features derived from chemical analyses, such as Alcohol Content, Ash and Magnesium. Based on these features, the samples can be categorized into three different wine quality types, low, medium, and high.
- The Car Evaluation Data Set contains 1,728 data samples of cars which are sorted into four categories regarding their acceptability: unacceptable, acceptable, good and very good. For this, six features can be used, which contain information about the price of the car, maintenance costs, number of doors, person capacity, luggage boot size and a safety rating. Here, the version by (Bohanec, 1988) is used.
- Since RENTT is constructed to also work for larger and more complex NN, the Forest Cover Type data set is also used, which requires a larger NN than the other data sets to achieve acceptable accuracy. It contains information on the types of forest cover for 30x30m cells in the Rocky Mountain region of the United States (Blackard, 1998). It includes 581,012 samples with 54 features describing various properties of the areas, such as elevation, distances to hydrology and soil type. With these, one of seven classes of forest cover type should be predicted, with classes such as spruce-fir, lodgepole pine and ponderosa pine.

B Performance of the Neural Network Models on the Data Sets

The results for the performance of the NN are shown in Table 2 for the regression performance and Table 3 for the classification performance. Both tables show that there are no differences in performance between NN and DT, which provides evidence that the theoretical equivalence of both models also holds in practice.

[2] https://www.kaggle.com/datasets/mathchi/diabetes-data-set

Table 2. Number of neurons in the hidden layer of the NN, performance of NN and DT on the regression data sets and number of nodes of the pruned decision trees.

Data Set	Neurons in Hidden Layers	NN MSE	DT MSE	Number of Nodes
Absolute	8, 4	$3.43 \cdot 10^{-10}$	$3.43 \cdot 10^{-10}$	39
Linear	8, 4	$2.53 \cdot 10^{-7}$	$2.53 \cdot 10^{-7}$	48
Diabetes Reg	8, 4	0.03	0.03	201
California Housing	8, 4	0.015	0.015	159

Table 3. Number of neurons in the hidden layer of the NN, performance of NN and DT on the classification data sets and number of nodes of the pruned decision trees.

Data Set	Neurons in Hidden Layers	NN Acc/ %	DT Acc/ %	Number of Nodes
Iris	8, 4	77	77	114
Diabetes Class	8, 4	75	75	169
Wine Quality	8, 4	92	92	68
Car Evaluation	8, 4	94	94	398
Forest Cover Type	64, 32	88	88	7, 638, 684

C Global Feature Importance Values

In Tables 4 and 5, the FI values for the global explanation methods are presented, for regression tasks in Table 4 and for classification tasks in Table 5. Both tables contain the values for the global methods SE and SHAP both on a NN and on an equivalent DT generated via RENTT and the RENTT-FI results.

Table 4. FI results for the global methods between NN and DT on regression data.

Data Set	Feature	SE DT	SE NN	SAGE DT	SAGE NN	RENTT-FI
linear	x_1	0.055 ± 0.001	0.055 ± 0.001	0.055 ± 0.001	0.053 ± 0.001	0.39
	x_2	0.0035 ± 0.0005	0.0045 ± 0.0005	0.0025 ± 0.0005	0.0029 ± 0.0004	0.1
	x_3	$(0.3 \pm 2.5) \cdot 10^{-5}$	$(2.8 \pm 2.5) \cdot 10^{-5}$	$(0.05 \pm 2.5) \cdot 10^{-5}$	$(2.0 \pm 2.1) \cdot 10^{-5}$	0.005
Diabetes Reg	age	$(2.0 \pm 0.6) \cdot 10^{-4}$	$(2.5 \pm 0.5) \cdot 10^{-4}$	$(1.2 \pm 0.7) \cdot 10^{-4}$	$(2.1 \pm 0.8) \cdot 10^{-4}$	−0.052
	sex	$(1.3 \pm 1.2) \cdot 10^{-4}$	$(1.0 \pm 1.2) \cdot 10^{-4}$	$(2.4 \pm 1.5) \cdot 10^{-4}$	$(3.5 \pm 1.6) \cdot 10^{-4}$	−0.085
	bmi	0.0085 ± 0.0002	0.0087 ± 0.0002	0.0098 ± 0.0003	0.0096 ± 0.0003	0.12
	bp	0.0056 ± 0.0002	0.0055 ± 0.0002	0.0054 ± 0.0002	0.0054 ± 0.0002	0.12
	s1	$(-6.1 \pm 0.4) \cdot 10^{-4}$	$(-5.2 \pm 0.4) \cdot 10^{-4}$	$(-5.5 \pm 0.5) \cdot 10^{-4}$	$(-5.9 \pm 0.5) \cdot 10^{-4}$	−0.037
	s2	$(-6.6 \pm 0.7) \cdot 10^{-4}$	$(-6.2 \pm 0.8) \cdot 10^{-4}$	$(-9.9 \pm 0.9) \cdot 10^{-4}$	$(-9.5 \pm 1.1) \cdot 10^{-4}$	−0.069
	s3	0.0017 ± 0.0001	0.0019 ± 0.0001	0.0017 ± 0.0001	0.0015 ± 0.0001	−0.044
	s4	0.0036 ± 0.0001	0.0036 ± 0.0001	0.0034 ± 0.0001	0.0034 ± 0.0001	0.051
	s5	0.0101 ± 0.0003	0.0095 ± 0.0002	0.0096 ± 0.0002	0.0098 ± 0.0003	0.19
	s6	0.0014 ± 0.0001	0.0014 ± 0.0001	0.0013 ± 0.0001	0.0014 ± 0.0001	0.045
California Housing	MedInc	0.0268 ± 0.0002	0.0263 ± 0.0002	0.0270 ± 0.0002	0.0266 ± 0.0002	0.28
	HouseAge	0.00129 ± 0.00005	0.00137 ± 0.00005	$(9.6 \pm 0.5) \cdot 10^{-4}$	$(8.7 \pm 0.5) \cdot 10^{-4}$	0.11
	AveRooms	0.0054 ± 0.0008	0.0076 ± 0.0009	0.0078 ± 0.0009	0.0073 ± 0.0009	−0.073
	AveBedrms	-0.0069 ± 0.0008	-0.0092 ± 0.0009	-0.0101 ± 0.0009	-0.0094 ± 0.0009	0.079
	Population	$(-1.4 \pm 0.3) \cdot 10^{-4}$	$(-1.7 \pm 0.3) \cdot 10^{-4}$	$(-1.8 \pm 0.2) \cdot 10^{-4}$	$(-2.1 \pm 0.2) \cdot 10^{-4}$	0.014
	AveOccup	0.0032 ± 0.0001	0.0033 ± 0.0001	0.0025 ± 0.0001	0.0025 ± 0.0001	−0.15
	Latitude	0.0107 ± 0.0003	0.0117 ± 0.0003	0.0079 ± 0.0003	0.0094 ± 0.0003	−0.36
	Longitude	$(1.2 \pm 3.3) \cdot 10^{-4}$	$(-5.6 \pm 2.9) \cdot 10^{-4}$	0.0014 ± 0.0003	$(3.6 \pm 3.0) \cdot 10^{-4}$	−0.66

Table 5. FI results for the global methods between NN and DT on classification data.

Data Set	Feature	SE DT	SE NN	SAGE DT	SAGE NN	RENTT-FI
Iris	sepal length	-0.0052 ± 0.0004	-0.0054 ± 0.0003	-0.013 ± 0.001	-0.013 ± 0.001	-0.052
	sepal width	0.030 ± 0.002	0.029 ± 0.002	0.014 ± 0.006	0.029 ± 0.005	-0.62
	petal length	0.040 ± 0.001	0.040 ± 0.001	0.101 ± 0.003	0.097 ± 0.003	0.17
	petal width	0.186 ± 0.004	0.201 ± 0.004	0.46 ± 0.01	0.46 ± 0.01	0.89
Diabetes Class	Pregnancies	0.0040 ± 0.0004	0.0045 ± 0.0003	0.0051 ± 0.0004	0.0056 ± 0.0004	0.095
	Glucose	0.108 ± 0.003	0.104 ± 0.003	0.112 ± 0.003	0.118 ± 0.003	1.37
	Blood Pressure	-0.0046 ± 0.0006	-0.0042 ± 0.0007	-0.0069 ± 0.0007	-0.0065 ± 0.0007	-0.61
	SkinThickness	0.0057 ± 0.0003	0.0058 ± 0.0004	0.0054 ± 0.0004	0.0050 ± 0.0004	0.17
	Insulin	-0.0014 ± 0.0005	-0.0014 ± 0.0005	$(-2.0 \pm 0.5) \cdot 10^{-4}$	-0.0016 ± 0.0005	-0.13
	BMI	0.033 ± 0.001	0.033 ± 0.001	0.037 ± 0.002	0.039 ± 0.002	0.59
	Pedigree	0.0099 ± 0.0008	0.0100 ± 0.0008	0.0099 ± 0.0009	0.0105 ± 0.0009	0.21
	Age	0.042 ± 0.002	0.044 ± 0.002	0.041 ± 0.002	0.040 ± 0.002	0.35
Wine Quality	alcohol	$(-8.1 \pm 3.5) \cdot 10^{-4}$	$(-7.2 \pm 3.2) \cdot 10^{-4}$	-0.0059 ± 0.0006	-0.0048 ± 0.0004	0.31
	malic acid	0.024 ± 0.001	0.022 ± 0.001	0.022 ± 0.002	0.025 ± 0.001	0.38
	ash	0.0026 ± 0.0007	0.0063 ± 0.0009	-0.018 ± 0.002	-0.014 ± 0.001	0.65
	alcalinity	0.035 ± 0.002	0.037 ± 0.001	0.050 ± 0.002	0.049 ± 0.002	1.01
	magnesium	$(-4.0 \pm 3.2) \cdot 10^{-4}$	$(8.1 \pm 3.1) \cdot 10^{-4}$	0.0031 ± 0.0005	0.0027 ± 0.0004	0.22
	total phenols	0.034 ± 0.001	0.032 ± 0.001	0.039 ± 0.002	0.038 ± 0.002	0.42
	flavanoids	0.135 ± 0.003	0.125 ± 0.003	0.161 ± 0.004	0.162 ± 0.003	0.98
	nonflavanoids	0.050 ± 0.002	0.049 ± 0.002	0.035 ± 0.003	0.039 ± 0.003	0.76
	PACs	0.013 ± 0.0008	0.014 ± 0.0007	0.015 ± 0.001	0.015 ± 0.001	0.43
	color intensity	0.031 ± 0.002	0.032 ± 0.002	0.029 ± 0.002	0.032 ± 0.002	0.56
	hue	0.066 ± 0.002	0.068 ± 0.002	0.085 ± 0.003	0.085 ± 0.002	0.56
	od280/od315	0.170 ± 0.004	0.165 ± 0.004	0.200 ± 0.005	0.197 ± 0.005	1.24
	proline	0.102 ± 0.003	0.098 ± 0.003	0.190 ± 0.005	0.184 ± 0.004	1.07
Car Evaluation	buying price	0.108 ± 0.004	0.103 ± 0.004	0.124 ± 0.004	0.124 ± 0.004	6.43
	maintenance	0.094 ± 0.003	0.085 ± 0.003	0.100 ± 0.004	0.101 ± 0.004	5.63
	doors	0.0023 ± 0.0005	0.0016 ± 0.0005	0.0042 ± 0.0008	0.0034 ± 0.0009	-0.67
	persons	0.183 ± 0.005	0.187 ± 0.005	0.189 ± 0.005	0.175 ± 0.005	2.02
	lug boot	0.028 ± 0.002	0.028 ± 0.002	0.041 ± 0.002	0.039 ± 0.002	-2.14
	safety	0.216 ± 0.005	0.218 ± 0.005	0.225 ± 0.005	0.226 ± 0.005	-3.47

Table 6. FI results for the global methods between NN and DT on classification data.

Data Set	Feature	SE DT	SE NN	SAGE DT	SAGE NN	RENTT-FI
Forest Cover Type	Elevation	$(9.8 \pm 6.6) \cdot 10^{-4}$	-0.0011 ± 0.0004	-0.52 ± 0.02	0.001 ± 0.001	183.7
	Aspect	$(6.3 \pm 6.8) \cdot 10^{-4}$	0.002 ± 0.001	-0.44 ± 0.02	0.004 ± 0.001	-10.2
	Slope	-0.022 ± 0.003	0.017 ± 0.003	-0.37 ± 0.02	0.022 ± 0.003	8.1
	Hor. Dist. Hydrology	0.006 ± 0.001	-0.001 ± 0.001	-0.51 ± 0.02	0.004 ± 0.002	-19.3
	Vert. Dist. Hydrology	-0.005 ± 0.003	0.019 ± 0.004	-0.24 ± 0.02	0.030 ± 0.003	25.2
	Hor. Dist. Roadways	0.011 ± 0.003	0.0037 ± 0.001	-0.80 ± 0.03	0.002 ± 0.001	-51.3
	Hillshade 9am	$(-9.1 \pm 2.4) \cdot 10^{-4}$	0.006 ± 0.003	0.015 ± 0.003	0.0027 ± 0.0007	58.1
	Hillshade Noon	$(10.0 \pm 0.6) \cdot 10^{-4}$	$(-2.4 \pm 0.54) \cdot 10^{-4}$	-0.39 ± 0.02	$(-2.0 \pm 0.5) \cdot 10^{-4}$	-37.3
	Hillshade 3pm	0.066 ± 0.004	0.026 ± 0.004	-0.80 ± 0.03	0.025 ± 0.004	20.1
	Hor. Dist. Fire Points	0.004 ± 0.003	0.010 ± 0.002	-0.73 ± 0.03	0.011 ± 0.002	-36.9
	Wilderness Area 1	0.012 ± 0.001	0.017 ± 0.003	-0.56 ± 0.02	0.025 ± 0.003	-38.9
	Wilderness Area 2	0.042 ± 0.001	0.060 ± 0.004	-0.43 ± 0.02	0.068 ± 0.004	-6.0
	Wilderness Area 3	0.0037 ± 0.0005	0.0081 ± 0.002	-0.066 ± 0.009	0.012 ± 0.002	-29.1
	Wilderness Area 4	-0.0053 ± 0.0006	0.022 ± 0.004	0.006 ± 0.008	0.029 ± 0.004	-3.4
	Soil Type 1	-0.013 ± 0.002	-0.011 ± 0.004	-0.25 ± 0.02	-0.016 ± 0.004	-1.3
	Soil Type 2	-0.005 ± 0.002	0.011 ± 0.004	-0.19 ± 0.02	0.020 ± 0.003	-1.9
	Soil Type 3	0.121 ± 0.003	0.047 ± 0.004	-0.92 ± 0.03	0.053 ± 0.005	-0.8
	Soil Type 4	-0.005 ± 0.002	0.007 ± 0.002	-0.39 ± 0.02	0.010 ± 0.002	-1.6
	Soil Type 5	$(1.0 \pm 6.1) \cdot 10^{-4}$	-0.0057 ± 0.001	-0.34 ± 0.02	-0.011 ± 0.002	-1.6
	Soil Type 6	-0.0011 ± 0.0004	0.0085 ± 0.002	-0.06 ± 0.01	0.014 ± 0.002	-1.7
	Soil Type 7	$(-8.9 \pm 6.9) \cdot 10^{-4}$	0.0035 ± 0.002	-0.41 ± 0.02	0.005 ± 0.002	0.0
	Soil Type 8	0.0 ± 0.0	$(-3.8 \pm 2.0) \cdot 10^{-4}$	0.0 ± 0.0	$(-3.0 \pm 1.0) \cdot 10^{-4}$	-0.19
	Soil Type 9	$(2.8 \pm 5.9) \cdot 10^{-4}$	-0.007 ± 0.002	-0.21 ± 0.02	-0.004 ± 0.001	-0.55
	Soil Type 10	0.011 ± 0.0009	0.0042 ± 0.001	-0.14 ± 0.01	0.009 ± 0.002	-9.9
	Soil Type 11	-0.0031 ± 0.0006	0.0017 ± 0.0005	-0.23 ± 0.01	0.0020 ± 0.0004	-2.4
	Soil Type 12	0.086 ± 0.002	0.052 ± 0.004	-0.44 ± 0.02	0.045 ± 0.005	-10.2
	Soil Type 13	0.007 ± 0.001	0.008 ± 0.002	-0.066 ± 0.01	0.010 ± 0.002	-7.9
	Soil Type 14	0.0056 ± 0.0006	$(-1.5 \pm 10) \cdot 10^{-4}$	-0.49 ± 0.02	$(-3.0 \pm 10) \cdot 10^{-4}$	-0.022

Table 7. FI results for the global methods between NN and DT on classification data.

Data Set	Feature	SE DT	SE NN	SAGE DT	SAGE NN	RENTT-FI
Forest Cover Type	Soil Type 15	-0.0015 ± 0.0004	0.001 ± 0.001	-0.05 ± 0.01	0.004 ± 0.002	0.0
	Soil Type 16	$(2.6 \pm 0.2) \cdot 10^{-4}$	$(-5.1 \pm 0.71) \cdot 10^{-4}$	-0.37 ± 0.02	$(-6.0 \pm 0.70) \cdot 10^{-4}$	-0.51
	Soil Type 17	$(1.20 \pm 0.05) \cdot 10^{-4}$	$(-5.1 \pm 5.0) \cdot 10^{-5}$	-0.16 ± 0.01	$(0.6 \pm 6.0) \cdot 10^{-5}$	-0.41
	Soil Type 18	$(0.8 \pm 4.0) \cdot 10^{-4}$	$(6.6 \pm 9.0) \cdot 10^{-4}$	0.0034 ± 0.001	$(-7.0 \pm 6.0) \cdot 10^{-4}$	0.47
	Soil Type 19	0.007 ± 0.001	-0.0072 ± 0.003	-0.38 ± 0.02	-0.007 ± 0.003	-0.80
	Soil Type 20	$(5.7 \pm 0.24) \cdot 10^{-4}$	$(-7.3 \pm 1.6) \cdot 10^{-4}$	-0.096 ± 0.009	$(-6.0 \pm 0.80) \cdot 10^{-4}$	-4.6
	Soil Type 21	$(0.59 \pm 6.2) \cdot 10^{-4}$	0.010 ± 0.003	-0.27 ± 0.02	0.006 ± 0.002	-0.27
	Soil Type 22	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	-13.9
	Soil Type 23	0.0025 ± 0.0006	0.008 ± 0.001	-0.074 ± 0.01	0.010 ± 0.001	-19.9
	Soil Type 24	-0.016 ± 0.003	0.040 ± 0.004	-0.11 ± 0.03	0.046 ± 0.004	-8.5
	Soil Type 25	0.0033 ± 0.001	0.014 ± 0.002	-0.089 ± 0.01	0.014 ± 0.002	-0.38
	Soil Type 26	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	15.9
	Soil Type 27	-0.0011 ± 0.0003	-0.0026 ± 0.003	0.16 ± 0.01	-0.001 ± 0.003	-0.26
	Soil Type 28	-0.001 ± 0.001	0.028 ± 0.003	-0.32 ± 0.02	0.023 ± 0.003	-0.73
	Soil Type 29	0.0076 ± 0.0008	0.001 ± 0.001	-0.63 ± 0.02	0.004 ± 0.002	-20.9
	Soil Type 30	0.0 ± 0.0	$(1.7 \pm 3.0) \cdot 10^{-4}$	0.0 ± 0.0	$(-9.0 \pm 4.3) \cdot 10^{-5}$	-6.76
	Soil Type 31	0.043 ± 0.002	0.338 ± 0.008	-0.83 ± 0.03	0.316 ± 0.008	-7.9
	Soil Type 32	0.0022 ± 0.0006	0.019 ± 0.002	-0.16 ± 0.01	0.029 ± 0.003	-23.9
	Soil Type 33	0.0019 ± 0.0002	0.0099 ± 0.001	-0.032 ± 0.006	0.0098 ± 0.002	-14.0
	Soil Type 34	0.043 ± 0.003	0.019 ± 0.003	-0.61 ± 0.02	0.022 ± 0.003	-0.45
	Soil Type 35	$(3.0 \pm 1.2) \cdot 10^{-4}$	-0.004 ± 0.002	-0.026 ± 0.004	-0.002 ± 0.001	0.088
	Soil Type 36	0.0 ± 0.0	$(5.0 \pm 3.0) \cdot 10^{-5}$	0.0 ± 0.0	$(-9.0 \pm 3.0) \cdot 10^{-5}$	0.0
	Soil Type 37	$(-6.7 \pm 5.5) \cdot 10^{-4}$	-0.0038 ± 0.001	-0.27 ± 0.02	-0.004 ± 0.002	-0.011
	Soil Type 38	-0.011 ± 0.003	0.012 ± 0.004	-0.094 ± 0.02	0.012 ± 0.003	-3.1
	Soil Type 39	0.029 ± 0.004	0.026 ± 0.004	-0.57 ± 0.03	0.022 ± 0.004	1.08
	Soil Type 40	0.166 ± 0.004	0.064 ± 0.005	-0.83 ± 0.02	0.058 ± 0.005	-0.67

D Feature Importance Results between Neural Networks and Decision Tree for Local and Global Methods

This section provides detailed quantitative comparisons of feature importance values between neural networks and their equivalent decision trees. The analysis uses Root Mean Squared Error (RMSE) and Relative Error (RE) to quantify the differences between the methods. Figures 20 and 21 underpin the results of Section 6.4.1: The local FI methods show significantly higher RE values than the global methods, while these still have non-negligible REs (Figures 20 and 21(b)). Additionally, the RE is in general lower for the regression tasks than for the classification tasks (Figure 21(a)). The results in Tables 8 and 9 further support the findings in Section 6.4.1 by demonstrating that LIME, SHAP, BD, SE, and SAGE produce similar but not identical feature importance values when applied to neural networks versus decision trees.

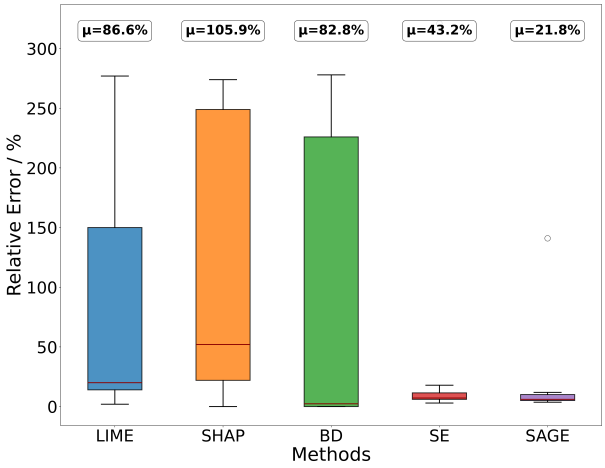
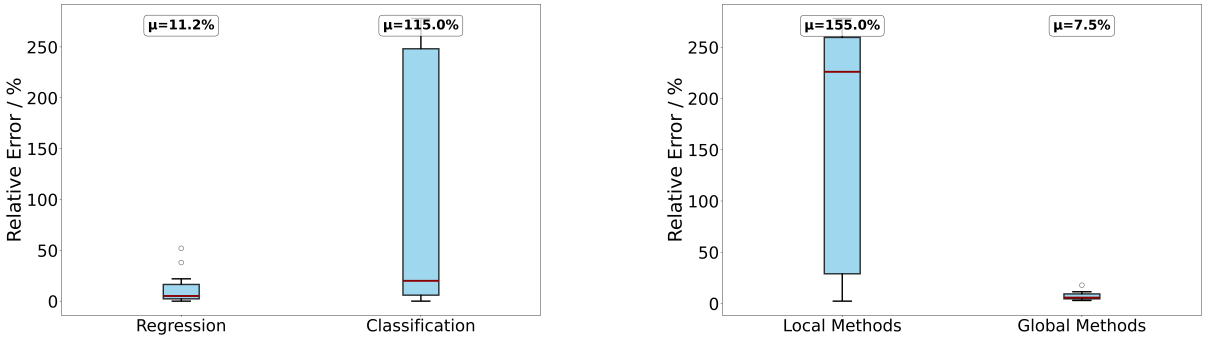


Fig. 20. Comparison of RE for FI methods when applied to functionally equivalent NN and DT across multiple data sets.



(a) Regression vs classification tasks, mean across all global and local methods.

(b) Local vs global FI methods, mean across all tasks.

Fig. 21. Boxplot summary comparing mean RE for FI methods when applied to functionally equivalent NN and DT models across all methods and data sets.

Table 8. RMSE and RE between NN and DT for local methods NN vs DT

Data Set	LIME NN vs DT		SHAP NN vs DT		BD NN vs DT	
	RMSE	RE / %	RMSE	RE / %	RMSE	RE / %
Absolute	0.008 ± 0.006	2 ± 2	$(3 \pm 2) \cdot 10^{-7}$	$(1.4 \pm 0.8) \cdot 10^{-4}$	$(3 \pm 2) \cdot 10^{-7}$	$(1.4 \pm 0.8) \cdot 10^{-4}$
Linear	0.02 ± 0.06	20 ± 60	0.002 ± 0.0	2.3 ± 0.0	0.002 ± 0.0	2.3 ± 0.0
Diabetes Reg	0.005 ± 0.001	12 ± 3	0.006 ± 0.004	22 ± 15	$(3 \pm 1) \cdot 10^{-8}$	$(10 \pm 4) \cdot 10^{-5}$
California Housing	0.05 ± 0.04	38 ± 34	0.04 ± 0.16	52 ± 184	0.005 ± 0.002	4 ± 2
Iris	0.06 ± 0.02	248 ± 83	0.07 ± 0.03	270 ± 100	0.09 ± 0.03	226 ± 76
Diabetes Class	0.010 ± 0.003	14 ± 4	0.01 ± 0.02	26 ± 29	$(3 \pm 1) \cdot 10^{-7}$	$(5 \pm 2) \cdot 10^{-4}$
Wine Quality	0.03 ± 0.01	150 ± 60	0.06 ± 0.02	249 ± 76	0.08 ± 0.02	225 ± 55
Car Evaluation	0.20 ± 0.04	277 ± 59	0.16 ± 0.04	274 ± 66	0.20 ± 0.07	278 ± 90
Forest Cover Type	-	-	-	-	-	-

Table 9. RMSE and RE between DT and NN FI values for SE and SAGE methods.

Data Set	SE NN vs DT		SAGE NN vs DT	
	RMSE	RE / %	RMSE	RE / %
Linear	$(6.0 \pm 1.0) \cdot 10^{-4}$	3.0 ± 2.7	0.0012 ± 0.001	6.1 ± 2.7
Diabetes Reg	$(2.0 \pm 0.0) \cdot 10^{-4}$	6.6 ± 1.9	$(1.0 \pm 0.0) \cdot 10^{-4}$	3.7 ± 2.2
California Housing	0.0012 ± 0.000	17.9 ± 3.4	$(7.0 \pm 0.0) \cdot 10^{-4}$	10.1 ± 3.3
Iris	0.0075 ± 0.002	11.5 ± 2.5	0.0078 ± 0.004	5.3 ± 2.9
Diabetes Class	0.0016 ± 0.001	6.1 ± 2.7	0.0023 ± 0.001	8.6 ± 2.8
Wine Quality	0.0037 ± 0.001	7.2 ± 1.5	0.0028 ± 0.001	4.2 ± 1.6
Car Evaluation	0.0046 ± 0.002	4.4 ± 2.0	0.0058 ± 0.002	5.1 ± 1.9
Forest Cover Type	0.0463 ± 0.001	321 ± 10	0.4221 ± 0.002	141 ± 1

E Additional Results for the Comparison of the Feature Importance Methods

This section provides comprehensive comparisons between different feature importance methods. The analysis includes both local methods (LIME, SHAP, BD, RENTT-FI) and global methods (SE, SAGE, RENTT-FI) evaluated using RMSE, relative error metrics and Krippendorff's alpha in ordinal and interval manner.

The results show that traditional methods (LIME, SHAP, BD) generally produce more similar feature importance values to each other, but differ significantly from the ground truth revealing method RENTT-FI.

Table 10. Pairwise RMSE and RE for the local FI methods for the regression tasks.

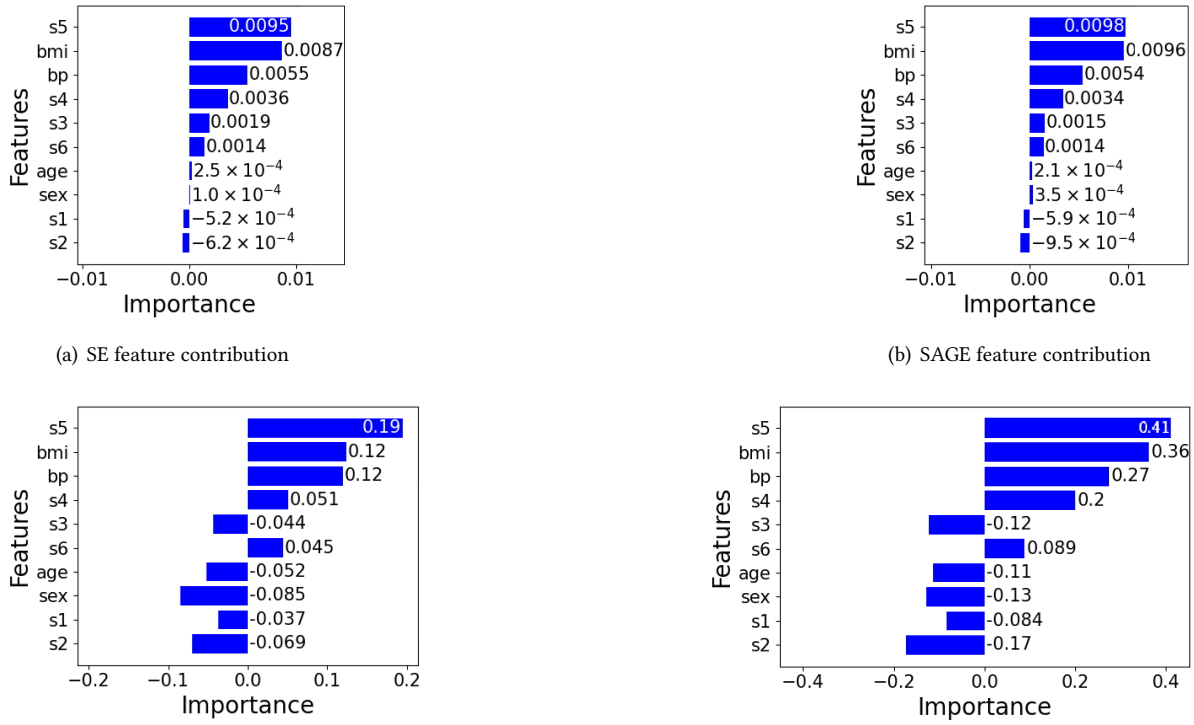
Data Set	Comparison	RMSE	RE / %
Absolute	LIME vs SHAP	0.13 ± 0.06	40 ± 18
	LIME vs BD	0.13 ± 0.06	40 ± 18
	LIME vs RENTT-FI	0.51 ± 0.01	153 ± 3
	SHAP vs BD	$(1.2 \pm 2.0) \cdot 10^{-9}$	$(5.0 \pm 8.0) \cdot 10^{-6}$
	SHAP vs RENTT-FI	0.49 ± 0.01	199 ± 4
	BD vs RENTT-FI	0.49 ± 0.01	199 ± 4
Linear	LIME vs SHAP	0.06 ± 0.03	55 ± 23
	LIME vs BD	0.06 ± 0.03	55 ± 23
	LIME vs RENTT-FI	0.33 ± 0.05	293 ± 48
	SHAP vs BD	$(3.7 \pm 82) \cdot 10^{-6}$	0.04 ± 1.0
	SHAP vs RENTT-FI	0.326 ± 0.005	383 ± 6
	BD vs RENTT-FI	0.326 ± 0.005	383 ± 6
Diabetes Reg	LIME vs SHAP	0.030 ± 0.010	73 ± 24
	LIME vs BD	0.032 ± 0.009	78 ± 23
	LIME vs RENTT-FI	0.14 ± 0.02	332 ± 51
	SHAP vs BD	0.011 ± 0.006	37 ± 21
	SHAP vs RENTT-FI	0.135 ± 0.013	478 ± 45
	BD vs RENTT-FI	0.136 ± 0.013	474 ± 45
California Housing	LIME vs SHAP	0.26 ± 0.02	221 ± 18
	LIME vs BD	0.12 ± 0.08	100 ± 63
	LIME vs RENTT-FI	0.46 ± 0.02	384 ± 19
	SHAP vs BD	0.280 ± 0.019	328 ± 22
	SHAP vs RENTT-FI	0.47 ± 0.02	549 ± 27
	BD vs RENTT-FI	0.47 ± 0.02	384 ± 18

Table 11. Pairwise RMSE and RE for the local FI methods for the classification tasks.

Data Set	Comparison	RMSE	RE / %
Iris	LIME vs SHAP	0.038 ± 0.010	160 ± 42
	LIME vs BD	0.044 ± 0.014	186 ± 58
	LIME vs RENTT-FI	0.70 ± 0.25	2921 ± 1058
	SHAP vs BD	0.034 ± 0.019	124 ± 70
	SHAP vs RENTT-FI	0.69 ± 0.25	2548 ± 917
	BD vs RENTT-FI	0.69 ± 0.26	1800 ± 678
Diabetes Class	LIME vs SHAP	0.057 ± 0.023	76 ± 30
	LIME vs BD	0.059 ± 0.024	79 ± 32
	LIME vs RENTT-FI	1.28 ± 0.30	1696 ± 401
	SHAP vs BD	0.023 ± 0.020	44 ± 37
	SHAP vs RENTT-FI	1.27 ± 0.30	2391 ± 561
	BD vs RENTT-FI	1.27 ± 0.30	2323 ± 545
Car Evaluation	LIME vs SHAP	0.095 ± 0.034	132 ± 47
	LIME vs BD	0.143 ± 0.049	200 ± 68
	LIME vs RENTT-FI	5.92 ± 0.36	8259 ± 506
	SHAP vs BD	0.110 ± 0.047	193 ± 83
	SHAP vs RENTT-FI	5.93 ± 0.36	10379 ± 632
	BD vs RENTT-FI	5.93 ± 0.36	8099 ± 497
Wine Quality	LIME vs SHAP	0.109 ± 0.018	496 ± 84
	LIME vs BD	0.105 ± 0.016	479 ± 72
	LIME vs RENTT-FI	0.99 ± 0.35	4532 ± 1587
	SHAP vs BD	0.088 ± 0.022	351 ± 88
	SHAP vs RENTT-FI	0.99 ± 0.36	3926 ± 1424
	BD vs RENTT-FI	0.98 ± 0.37	2696 ± 1002

Table 12. Pairwise RMSE and RE for the global FI methods for the regression and classification tasks.

Data Set	Comparison	RMSE	RE / %
Linear	SE vs SAGE	0.0015 ± 0.0005	7.7 ± 2.7
	SE vs RENTT-FI	0.201 ± 0.0004	218 ± 0.4
	SAGE vs RENTT-FI	0.203 ± 0.0004	221 ± 0.4
Diabetes Reg	SE vs SAGE	0.0004 ± 0.00007	11.0 ± 2
	SE vs RENTT-FI	0.0896 ± 0.00004	212.1 ± 0.1
	SAGE vs RENTT-FI	0.0894 ± 0.00005	211.4 ± 0.1
California Housing	SE vs SAGE	0.0010 ± 0.0002	13.0 ± 3
	SE vs RENTT-FI	0.293 ± 0.0002	262.5 ± 0.2
	SAGE vs RENTT-FI	0.293 ± 0.0002	262.9 ± 0.2
Iris	SE vs SAGE	0.133 ± 0.003	121 ± 3
	SE vs RENTT-FI	0.478 ± 0.001	190.6 ± 0.5
	SAGE vs RENTT-FI	0.391 ± 0.003	134 ± 1
Diabetes Class	SE vs SAGE	0.0057 ± 0.0007	20.9 ± 3
	SE vs RENTT-FI	0.555 ± 0.0005	238.0 ± 0.2
	SAGE vs RENTT-FI	0.5500 ± 0.0005	234.7 ± 0.2
Wine Quality	SE vs SAGE	0.0288 ± 0.0009	50.0 ± 2
	SE vs RENTT-FI	0.669 ± 0.0005	188.2 ± 0.2
	SAGE vs RENTT-FI	0.652 ± 0.0007	179.5 ± 0.2
Car Evaluation	SE vs SAGE	0.013 ± 0.002	12.2 ± 2
	SE vs RENTT-FI	3.93 ± 0.001	225.0 ± 0.08
	SAGE vs RENTT-FI	3.93 ± 0.002	224.2 ± 0.09
Forest Cover Type	SE vs SAGE	0.0052 ± 0.0005	27.9 ± 2.8
	SE vs RENTT-FI	30.1 ± 0.0004	455.3 ± 0.006
	SAGE vs RENTT-FI	30.1 ± 0.0004	455.3 ± 0.006



(a) SE feature contribution

(b) SAGE feature contribution

(c) RENTT-FI feature contribution

(d) RENTT-FI feature effect

Fig. 22. Global FI by SE, SAGE and RENTT on an NN trained on the Diabetes Reg data set.

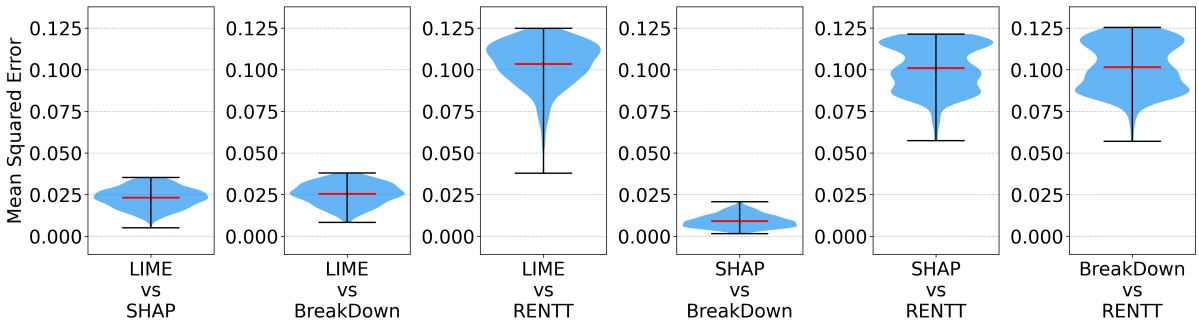
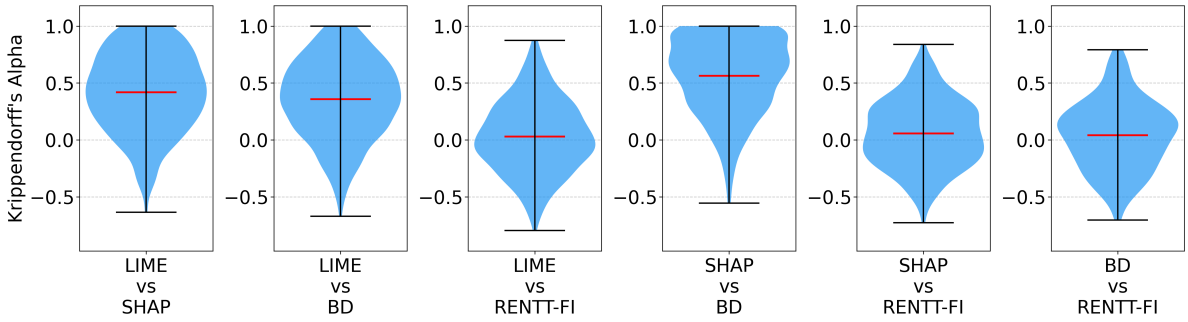
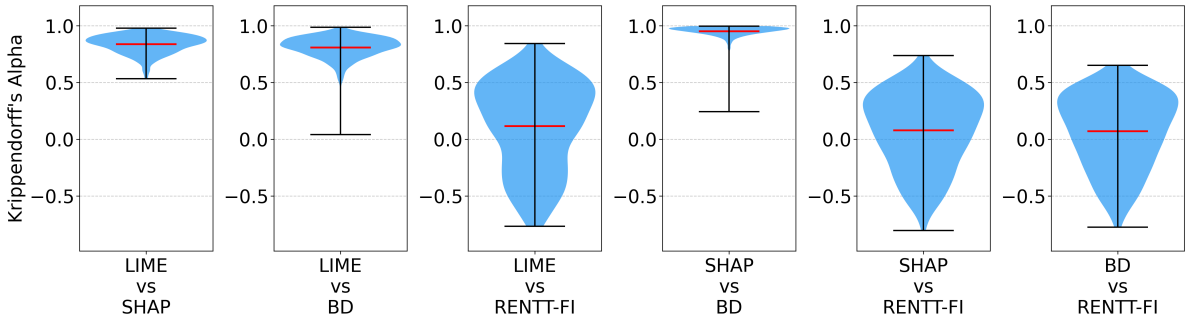


Fig. 23. Violin plot of the RMSE between the different FI-methods for the Diabetes Reg data set. Only the 95th percentile of samples is pictured.

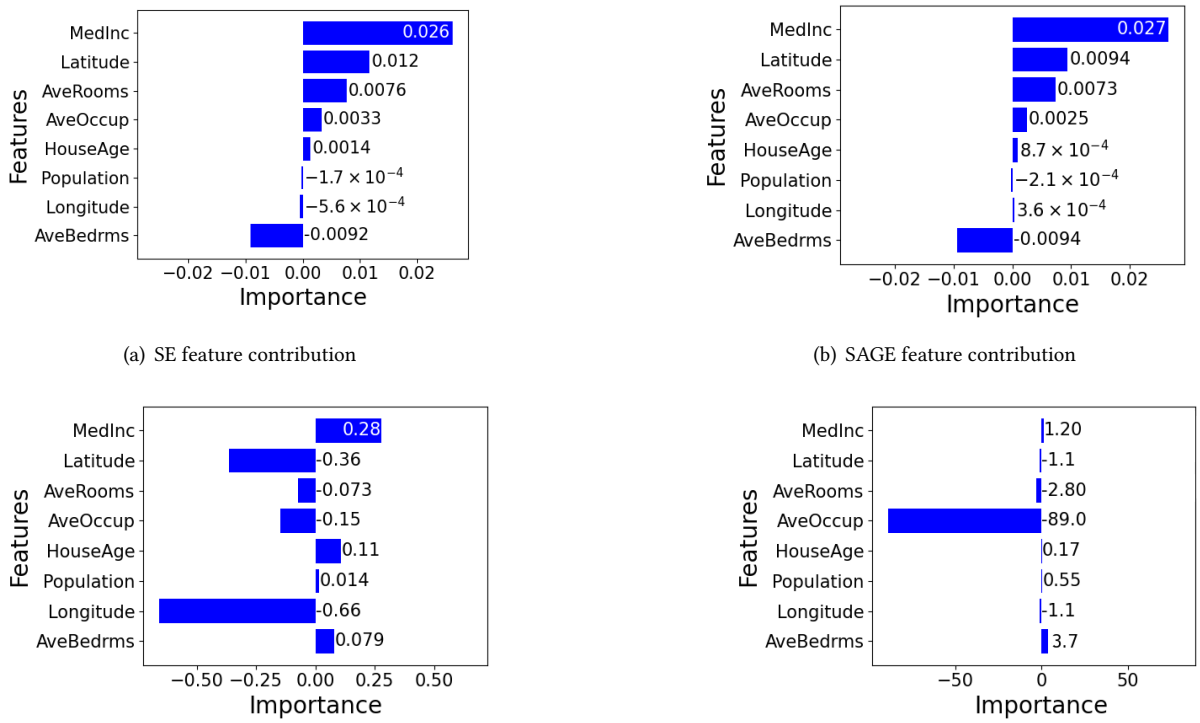


(a) Ordinal Krippendorff's α (ranking)



(b) Interval Krippendorff's α (values)

Fig. 24. Violin plots of Krippendorff's α for the Diabetes Reg data set, calculated samplewise between different FI methods. Only the 95th percentile of samples is pictured.



(a) SE feature contribution

(b) SAGE feature contribution

(c) RENTT-FI feature contribution

(d) RENTT-FI feature effect

Fig. 25. Global FI by SE, SAGE and RENTT on an NN trained on the California Housing data set.

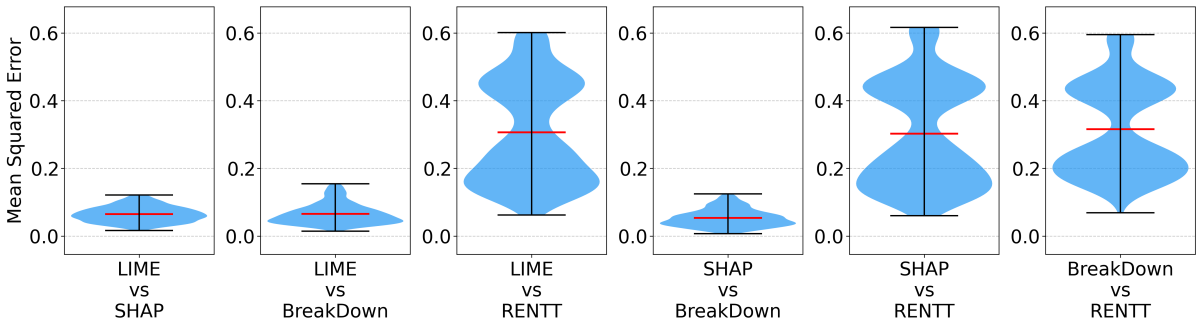
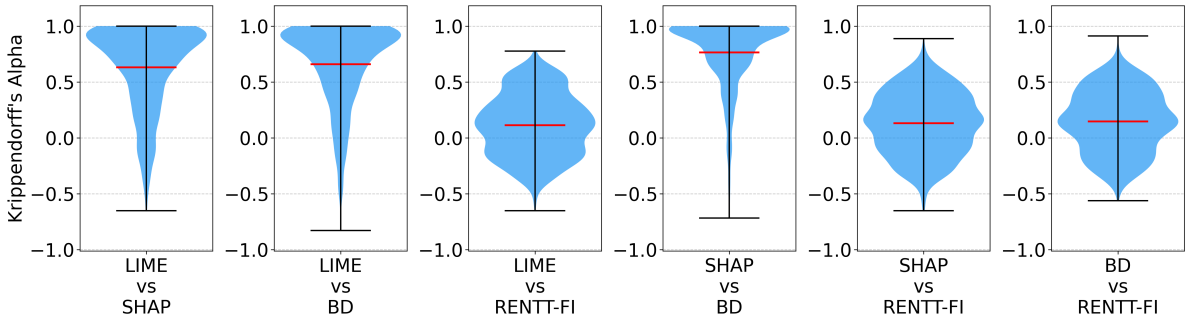
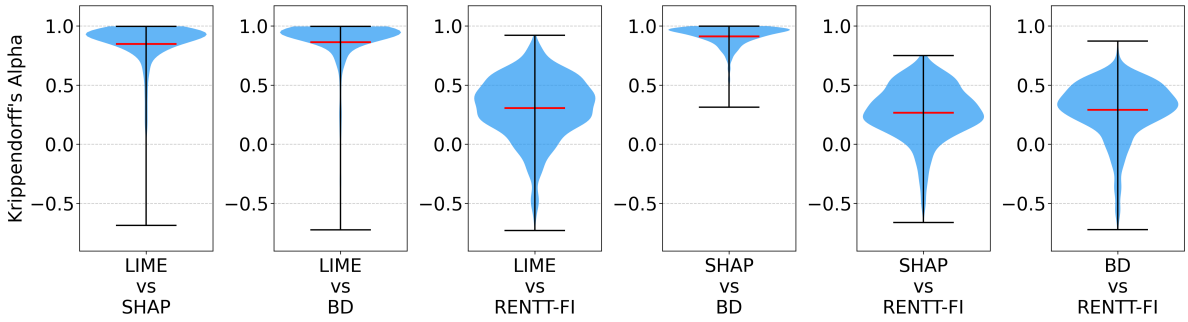


Fig. 26. Violin plot of the RMSE between the different FI-methods for the California Housing data set. Only the 95th percentile of samples is pictured.

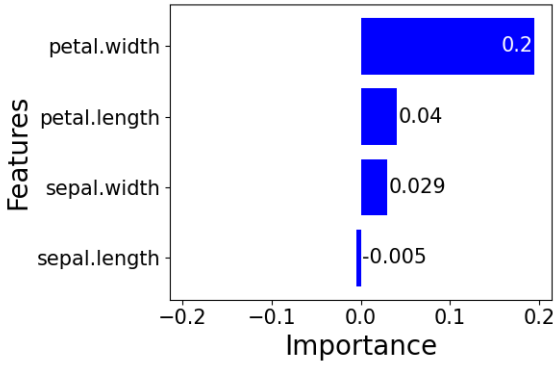


(a) Ordinal Krippendorff's α (ranking)

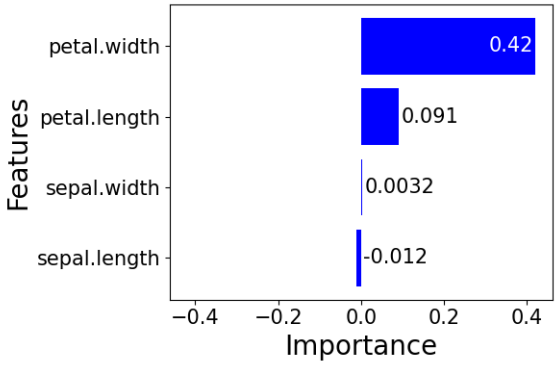


(b) Interval Krippendorff's α (values)

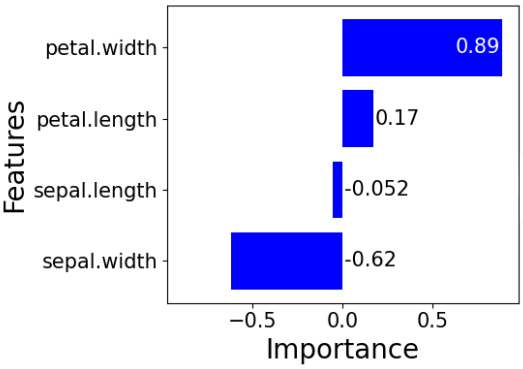
Fig. 27. Violin plots of Krippendorff's α for the California Housing data set, calculated samplewise between different FI methods. Only the 95th percentile of samples is pictured.



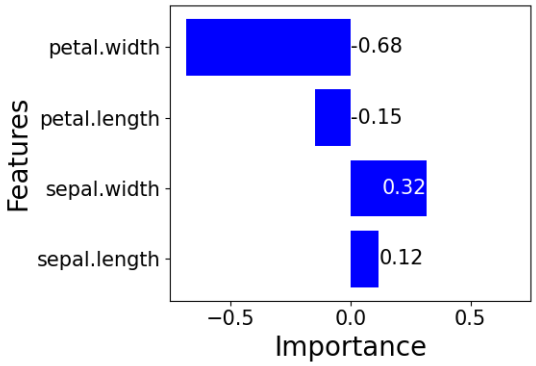
(a) SE feature contribution



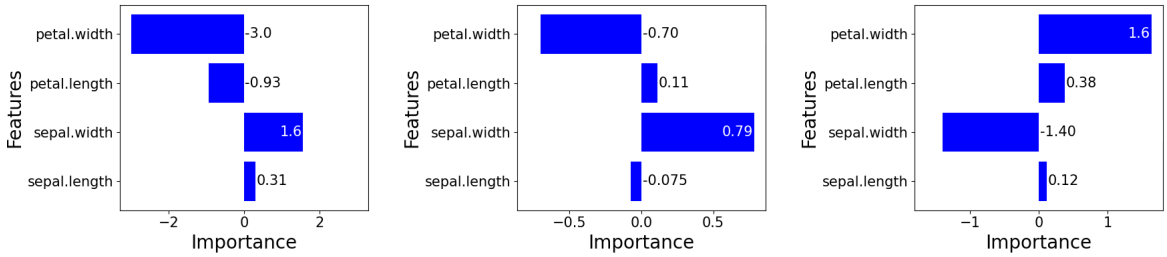
(b) SAGE feature contribution



(c) RENTT-FI feature contribution



(d) RENTT-FI feature effect



(e) RENTT-FI feature effect classwise (Classes: Setosa, Versicolour, and Virginica)

Fig. 28. Global FI by SE, SAGE and RENTT on an NN trained on the Iris data set.

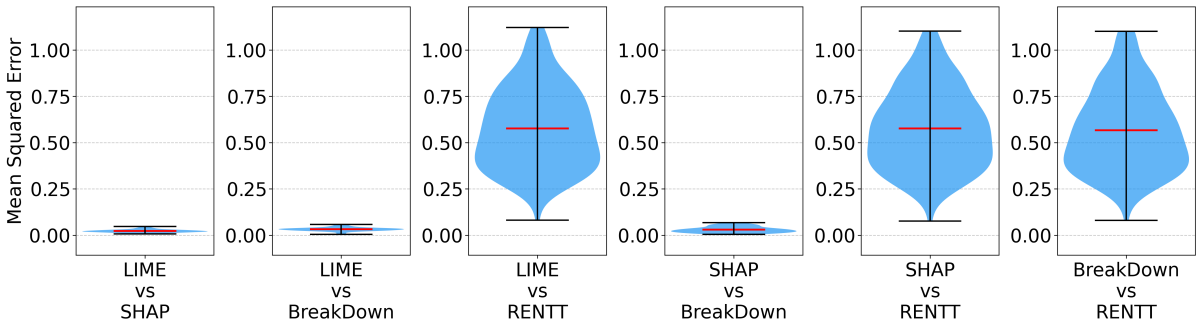
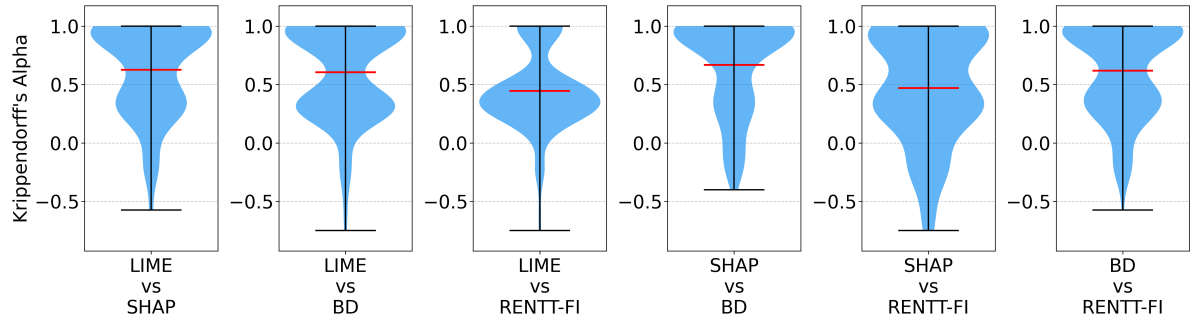
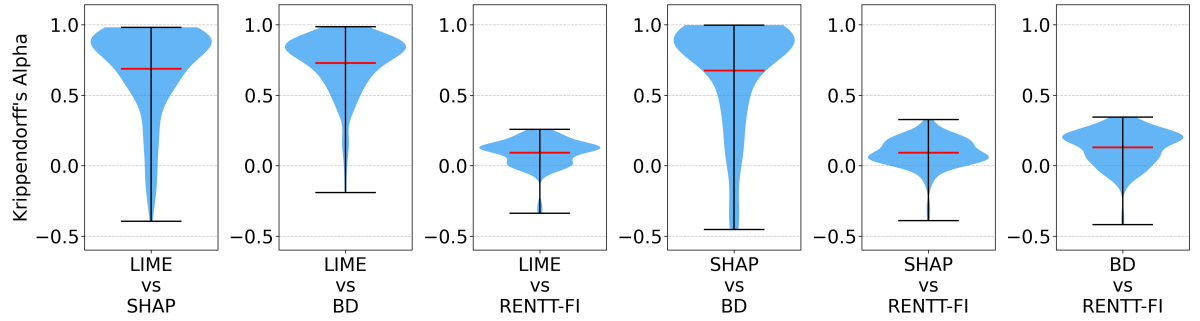


Fig. 29. Violin plot of the RMSE between the different FI-methods for the Iris data set. Only the 95th percentile of samples is pictured.

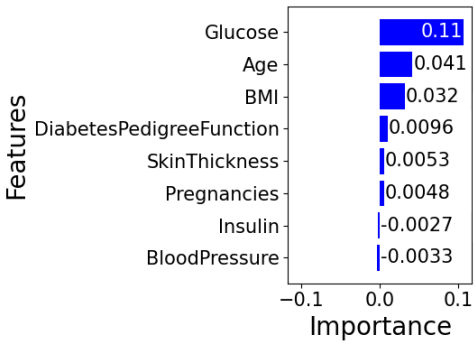


(a) Ordinal Krippendorff's α (ranking)

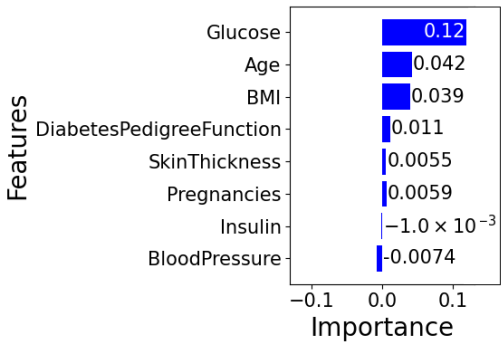


(b) Interval Krippendorff's α (values)

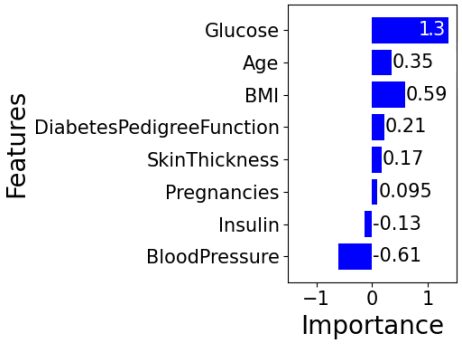
Fig. 30. Violin plots of Krippendorff's α for the Iris data set, calculated samplewise between different FI methods. Only the 95th percentile of samples is pictured.



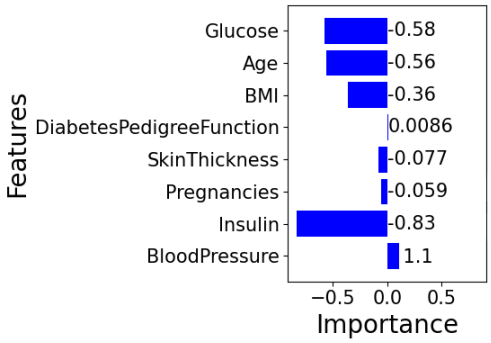
(a) SE feature contribution



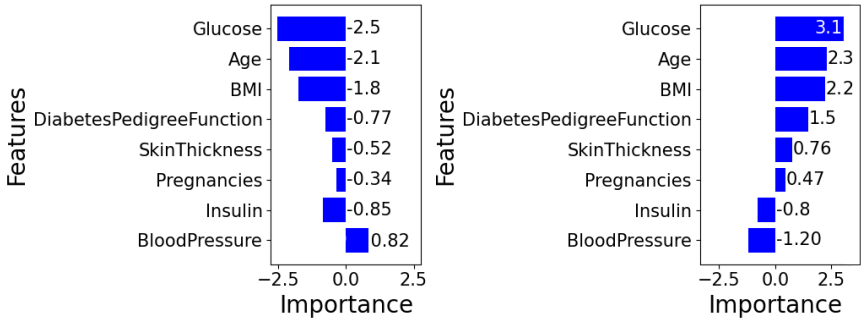
(b) SAGE feature contribution



(c) RENTT-FI feature contribution



(d) RENTT-FI feature effect



(e) RENTT-FI feature effect classwise (Classes: negative, positive)

Fig. 31. Global FI by SE, SAGE and RENTT on an NN trained on the Diabetes Class data set.

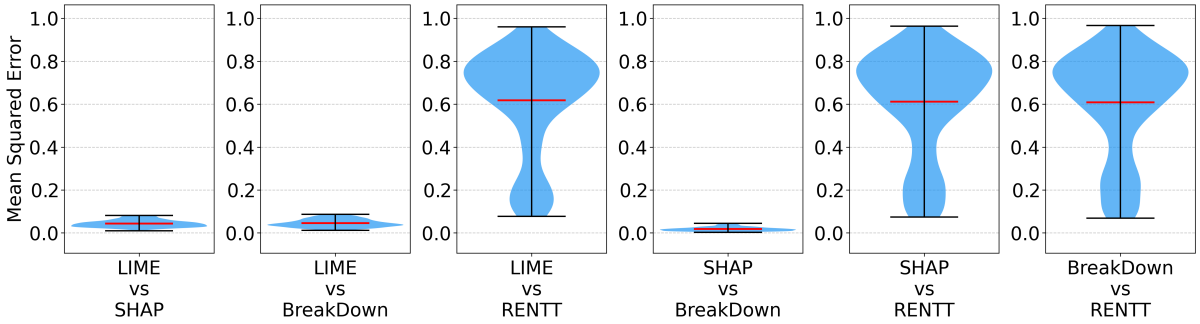
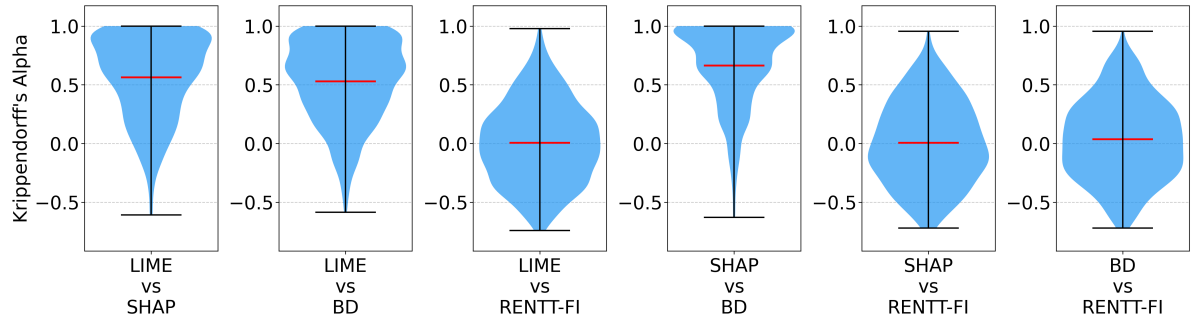
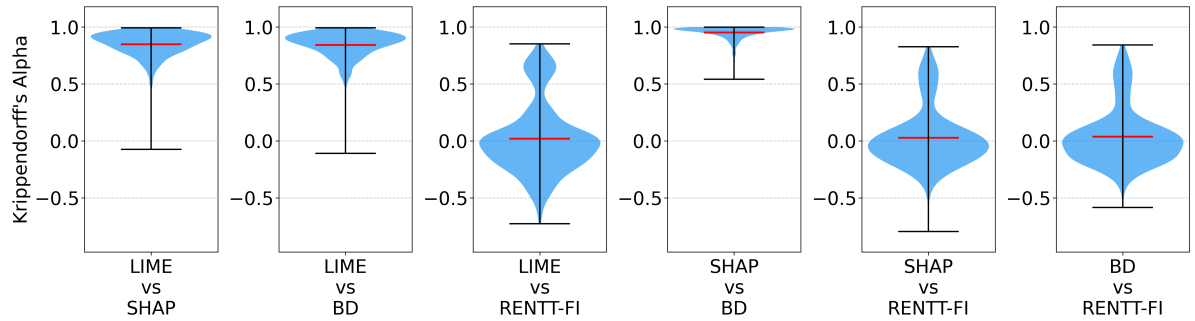


Fig. 32. Violin plot of the RMSE between the different FI-methods for the Diabetes Class data set. Only the 95th percentile of samples is pictured.

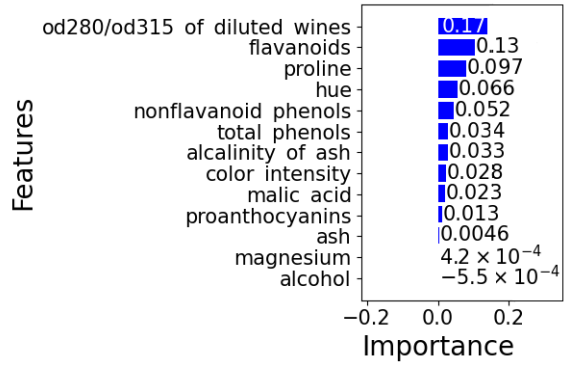


(a) Ordinal Krippendorff's α (ranking)

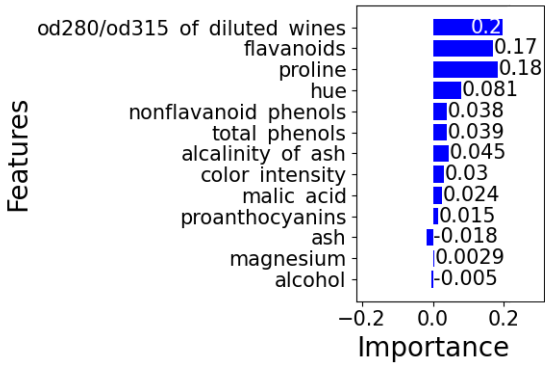


(b) Interval Krippendorff's α (values)

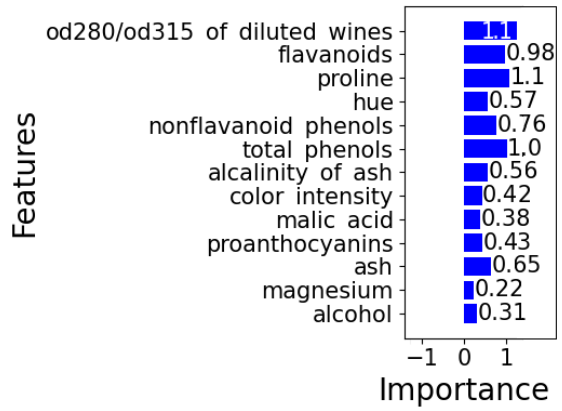
Fig. 33. Violin plots of Krippendorff's α for the Diabetes Class data set, calculated samplewise between different FI methods. Only the 95th percentile of samples is pictured.



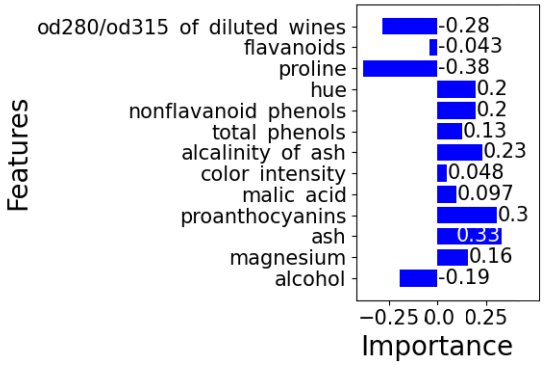
(a) SE feature contribution



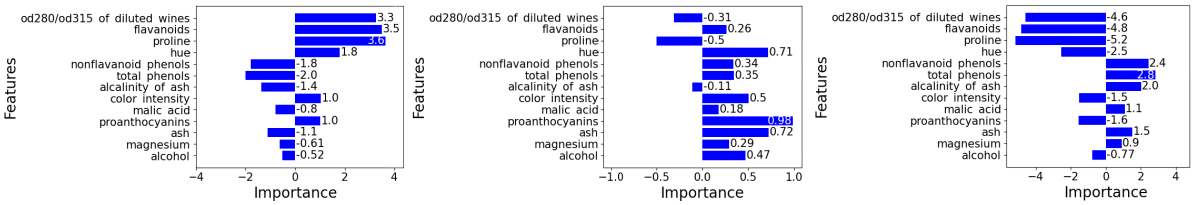
(b) SAGE feature contribution



(c) RENTT-FI feature contribution



(d) RENTT-FI feature effect



(e) RENTT-FI feature effect classwise (Classes:low, medium, high quality)

Fig. 34. Global FI by SE, SAGE and RENTT on an NN trained on the Wine Quality data set.

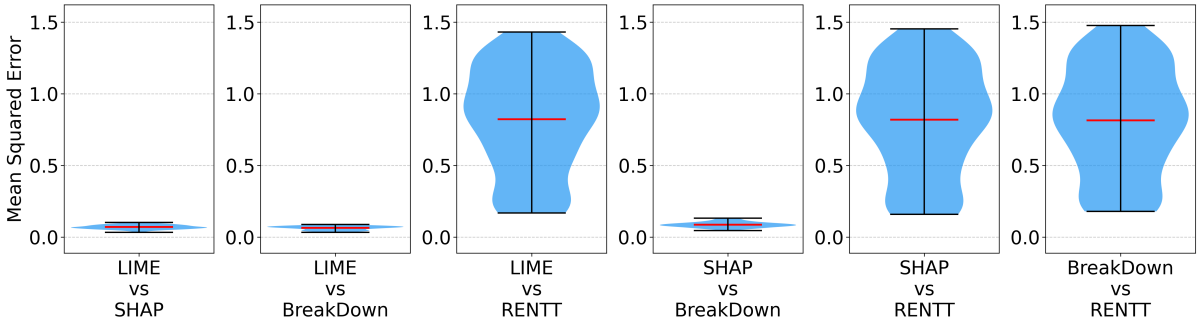
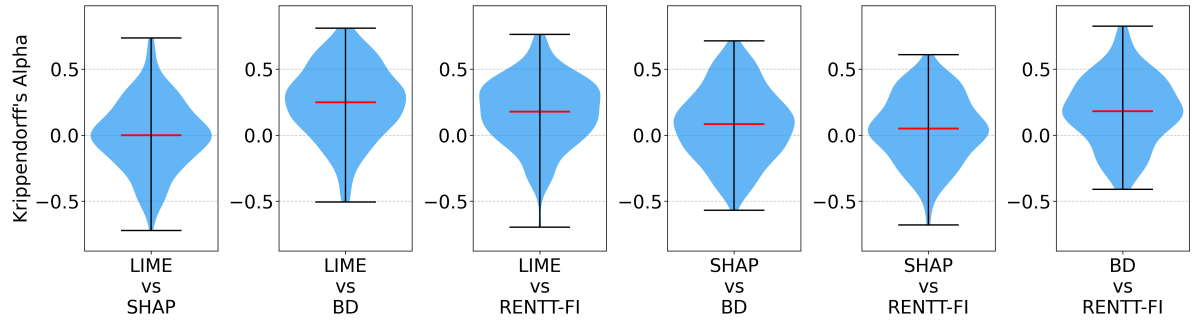
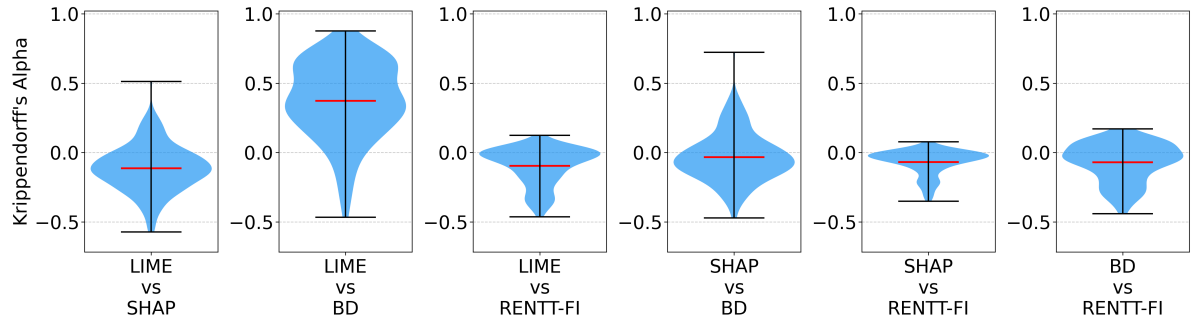


Fig. 35. Violin plot of the RMSE between the different FI-methods for the Wine Quality data set. Only the 95th percentile of samples is pictured.

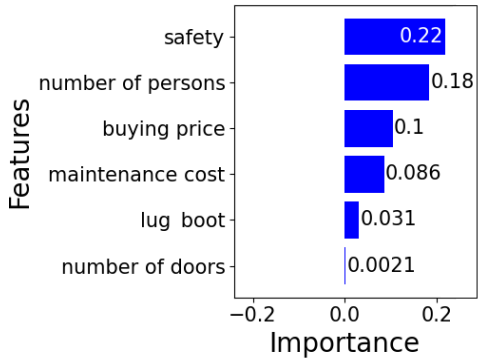


(a) Ordinal Krippendorff's α (ranking)

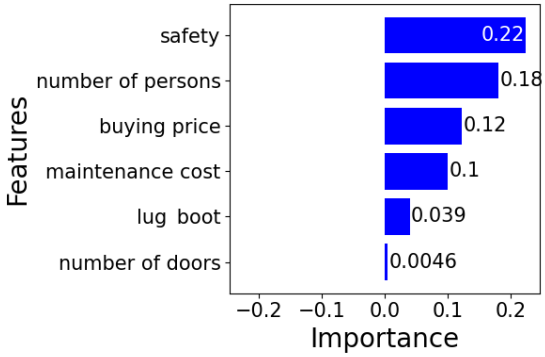


(b) Interval Krippendorff's α (values)

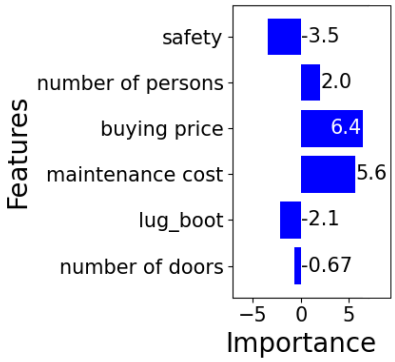
Fig. 36. Violin plots of Krippendorff's α for the Wine Quality data set, calculated samplewise between different FI methods. Only the 95th percentile of samples is pictured.



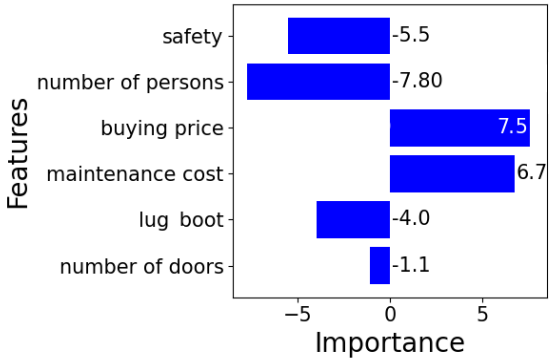
(a) SE feature contribution



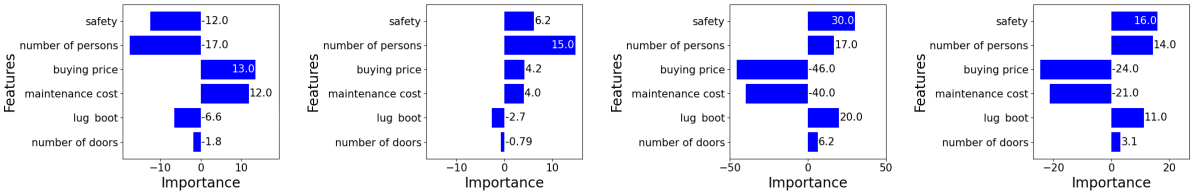
(b) SAGE feature contribution



(c) RENTT-FI feature contribution



(d) RENTT-FI feature effect



(e) RENTT-FI feature effect classwise (Classes: unacceptable, acceptable, good, very good)

Fig. 37. Global FI by SE, SAGE and RENTT on an NN trained on the Car Evaluation data set.

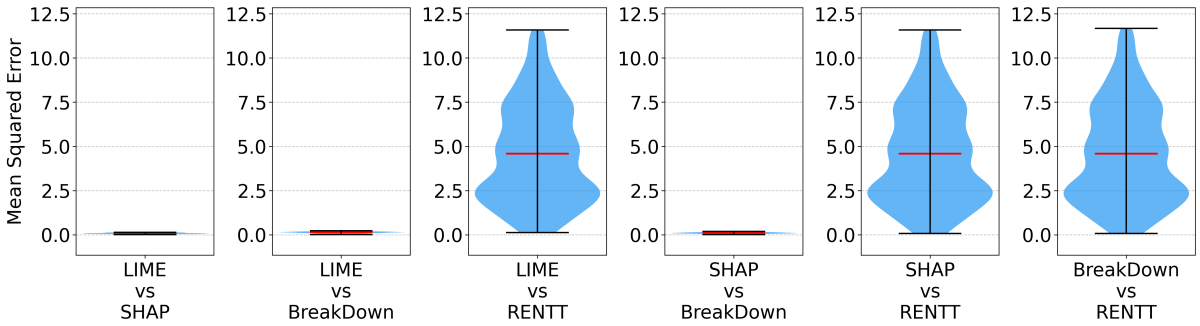
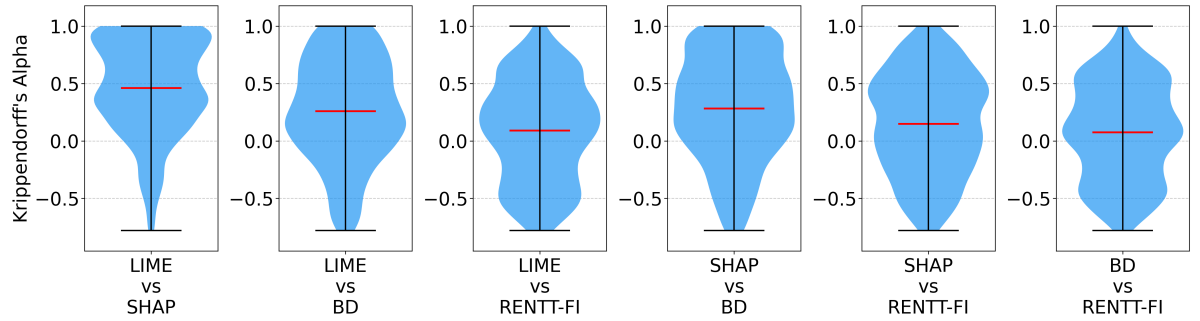
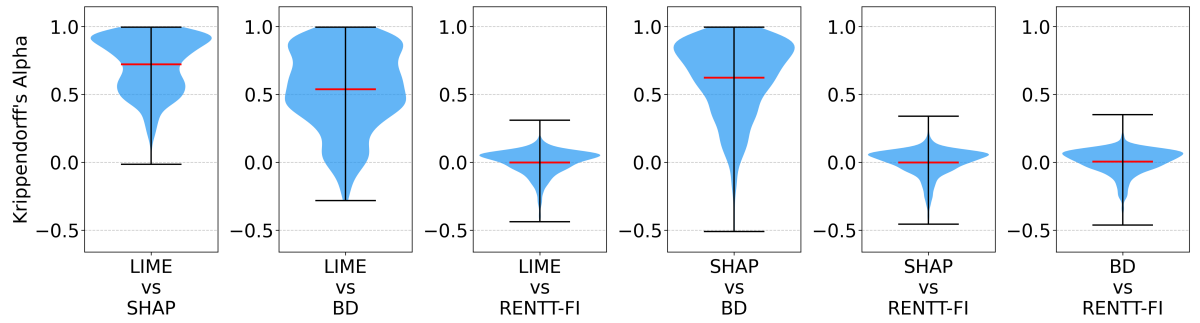


Fig. 38. Violin plot of the RMSE between the different FI-methods for the Car Evaluation data set. Only the 95th percentile of samples is pictured.

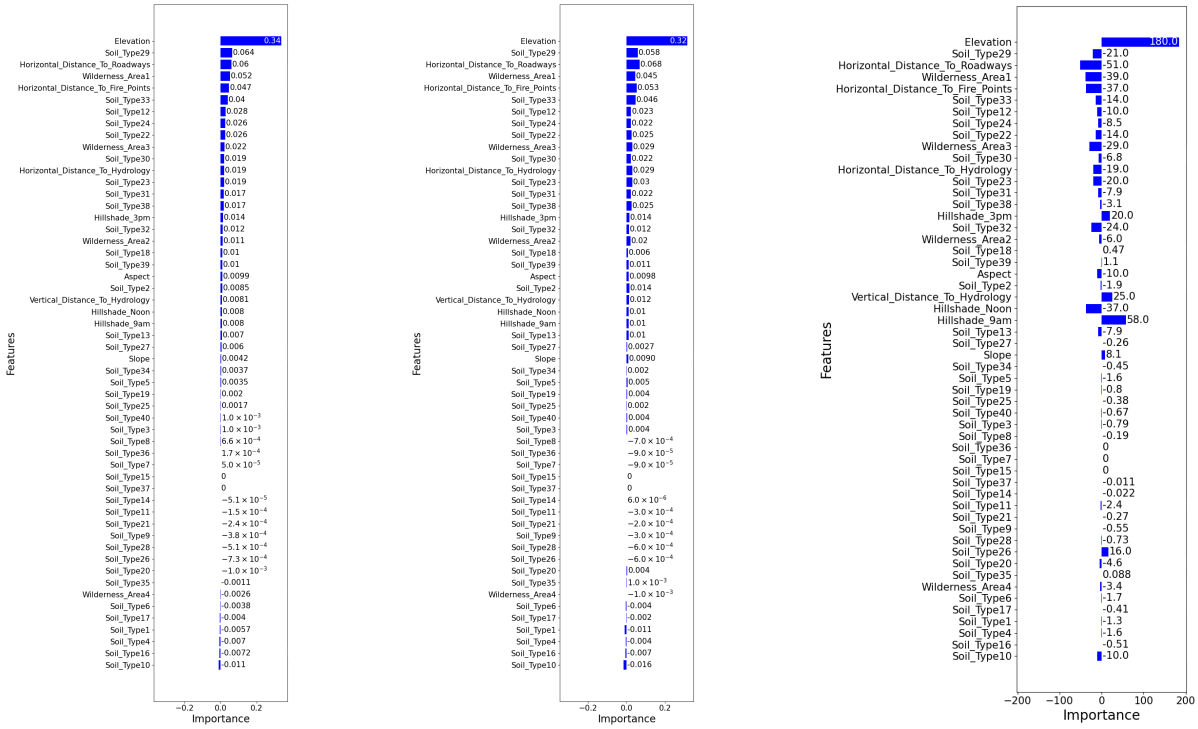


(a) Ordinal Krippendorff's α (ranking)



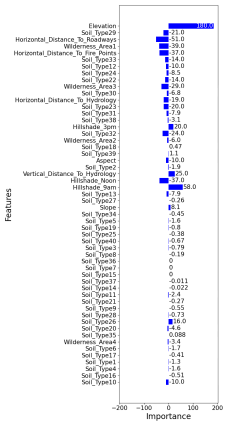
(b) Interval Krippendorff's α (values)

Fig. 39. Violin plots of Krippendorff's α for the Car Evaluation data set, calculated samplewise between different FI methods. Only the 95th percentile of samples is pictured.

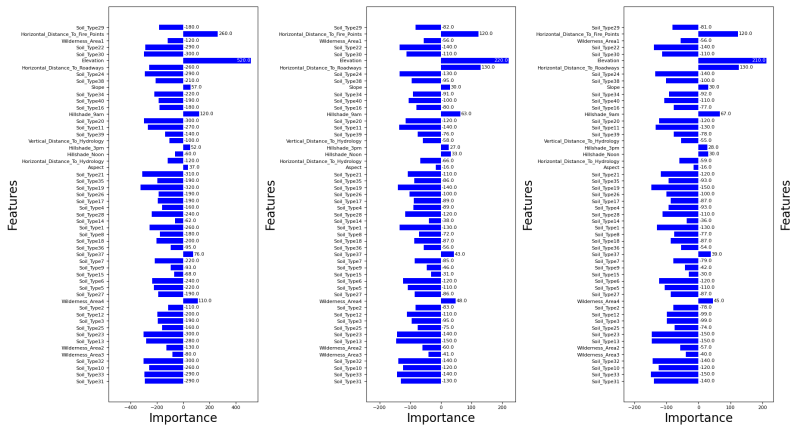


(a) SE feature contribution (b) SAGE feature contribution (c) RENTT-FI feature contribution

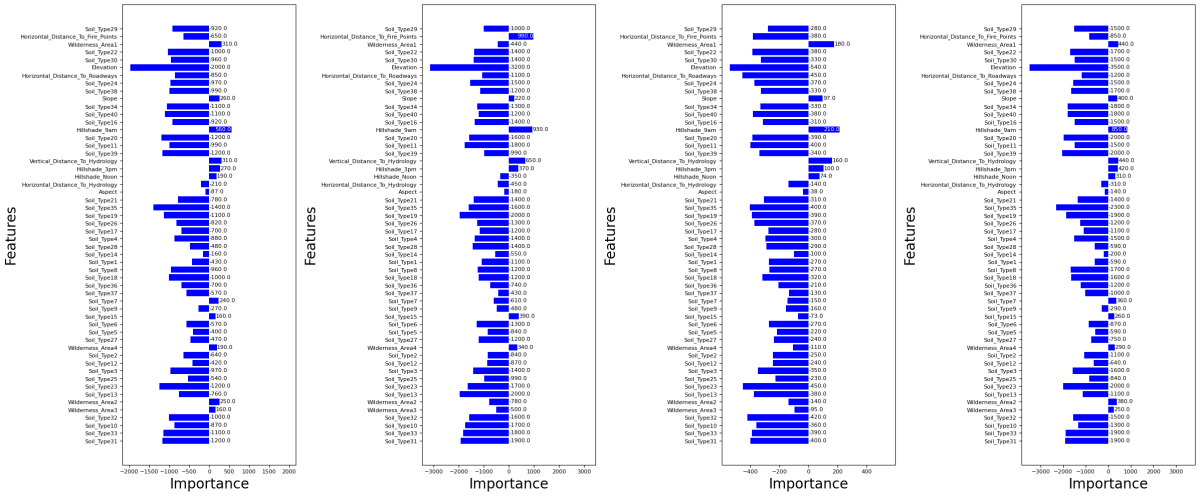
Fig. 40. Global feature contribution by SE, SAGE and RENTT on an NN trained on the Forest Cover Type data set.



(a) RENTT global feature effect



(b) RENTT feature effect for the classes Spruce/Fir, Lodgepole Pine, Ponderosa Pine



(c) RENTT feature effect for the classes Cottonwood/Willow, Aspen, Douglas-fir, Krummholz

Fig. 41. Feature effect global and classwise by RENTT on an NN trained on the Forest Cover Type data set.

F Results for the Runtime Analysis of Feature Importance Methods

This section presents the computational efficiency analysis of different feature importance methods. The runtime measurements demonstrate the significant computational advantages of RENTT-FI over traditional perturbation-based methods. The results show that RENTT-FI is orders of magnitude faster than LIME, SHAP, BD, SE, and SAGE methods across all data sets.

Table 13. Mean runtime results for local FI methods with standard deviation.

Data Set	**Method**	**Runtime / s**
Linear	LIME	13.0 ± 0.2
	SHAP	74.4 ± 0.2
	BD	25.8 ± 0.1
	RENTT-FI	0.024 ± 0.000
Diabetes Reg	LIME	12.8 ± 0.1
	SHAP	163.5 ± 0.4
	BD	62.1 ± 0.1
	RENTT-FI	0.027 ± 0.000
California Housing	LIME	12.8 ± 0.1
	SHAP	154.8 ± 1.2
	BD	58.0 ± 0.4
	RENTT-FI	0.042 ± 0.007
Iris	LIME	24.6 ± 0.1
	SHAP	74.8 ± 0.1
	BD	26.4 ± 0.1
	RENTT-FI	0.025 ± 0.001
Diabetes Class	LIME	24.9 ± 0.1
	SHAP	151.9 ± 0.6
	BD	56.4 ± 0.3
	RENTT-FI	0.027 ± 0.001
Car Evaluation	LIME	27.8 ± 0.2
	SHAP	134.8 ± 0.3
	BD	49.2 ± 0.3
	RENTT-FI	0.043 ± 0.007
Wine Quality	LIME	24.4 ± 0.1
	SHAP	198.1 ± 0.1
	BD	75.7 ± 0.1
	RENTT-FI	0.025 ± 0.000

Table 14. Mean runtime results for global FI methods with standard deviation.

Data Set	**Method**	**Runtime / s**
Linear	SE	0.250 ± 0.040
	SAGE	0.233 ± 0.063
	RENTT-FI	$(16.0 \pm 0.4) \cdot 10^{-4}$
Diabetes Reg	SE	1.8 ± 0.1
	SAGE	2.0 ± 0.2
	RENTT-FI	$(4.1e-03 \pm 4.3e-05)$
California Housing	SE	12.6 ± 0.8
	SAGE	22.4 ± 1.8
	RENTT-FI	0.019 ± 0.007
Iris	SE	0.060 ± 0.005
	SAGE	0.025 ± 0.005
	RENTT-FI	0.0025 ± 0.0002
Diabetes Class	SE	1.4 ± 0.0
	SAGE	12.9 ± 0.7
	RENTT-FI	0.0040 ± 0.0003
Car Evaluation	SE	175.2 ± 5.4
	SAGE	222.1 ± 18.6
	RENTT-FI	0.020 ± 0.007
Wine Quality	SE	0.460 ± 0.057
	SAGE	0.37 ± 0.04
	RENTT-FI	0.0017 ± 0.0002