Muppet: A Modular and Constructive Decomposition for Perturbation-based Explanation Methods

Quentin Ferré, Ismail Bachchar, Hakima Arroubat, Aziz Jedidi, Youssef Achenchabe, Antoine Bonnefoy
Euranova, 146 rue Paradis 13006 Marseille

***Abstract*—The topic of explainable AI has recently received attention driven by a growing awareness of the need for transparent and accountable AI. In this paper, we focus on local perturbation-based explanation methods, which currently represent a majority of the post-hoc explainability literature and usage. We have observed a fundamental commonality among these methods, forming the essence of our contribution. We propose a novel methodology to decompose any state-of-the-art perturbation-based explainability approach into four blocks. This decomposition framework offers a concise analysis of existing similarities between methods and accelerates the development of new ones and their variants by promoting the reuse of their common blocks. In addition, we provide Muppet: an open-source Python library that offers (i) explainable AI methods to debug and interpret ML models, (ii) standardized API based on our decomposition methodology, facilitating contribution and benchmarking, and (iii) built-in modularity design: enables the decomposition of every method into reusable modules, thus speeding up their implementation. Available at https://github.com/euranova/muppet/.**

***Index Terms*—explainable AI, perturbation-based methods, multi-modal.**

I. Introduction

Artificial Intelligence (AI) has revolutionized many domains. However, this power comes with a challenge: shifting from simple interpretable models to complex but powerful ones such as Deep Neural Networks (DNNs). As a result, their lack of transparency limits their real-world use in high-stakes domains, such as healthcare, criminal justice, and finance [1].

This growing awareness of the need for accountable and transparent AI systems, promoted by regulatory movements like the GDPR, and the converging AI Act, emphasizes the importance of responsible AI practices, and the right to meaningful explanations for AI-driven decisions.

In response to these concerns, the field of explainable AI (XAI) has gained prominence [2] to allow AI to resonate with human comprehension for developers and end users. This field follows two distinct approaches. The first is to build inherently interpretable ML models [3]. The second is to employ post-hoc XAI methods to explain the predictions of existing ML models, regardless of their complexity. The latter has gained popularity, with numerous methods proposed in the literature, as in [4]–[6]. These methods can provide insights for one prediction at a time (local explanations) or for the model's global behaviour (global explanations). Notably, most post-hoc approaches are local and model-agnostic; meaning they offer per-sample explanations without relying on the model's architecture, which makes them applicable to most ML models.

The authors acknowledge the ANR – FRANCE (French National Research Agency) for its financial support under reference ANR-20-CE23-0020 (TAU-DoS project).

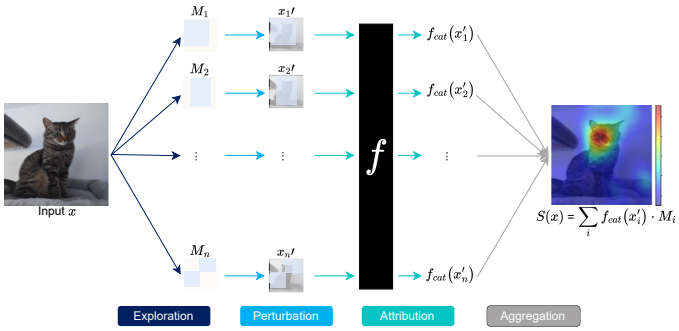

Fig. 1: Illustration of the Muppet decomposition of a perturbation-based explanation method, namely RISE [10]. It follows four steps: 1) *Exploration* It takes an input image x and generates masks M_i, 2) *Perturbation* pixels of x highlighted in blue in M_i are perturbed to get x_i', 3) *Attribution* calculates the perturbations' contribution as cat predicted class by $f_{cat}(x_i')$, then 4) *Aggregation* which compiles the perturbations' impacts using weighted sum of masks by prediction scores to form a saliency map $S(x)$.

Perturbation-based approaches represent a substantial portion of the post-hoc family of methods. Simply put, they involve altering the input of a model, such as masking part of an image or substituting words in a sentence, to observe the model's behavior on them, e.g. changes in the model's output. Several methods have been proposed in the literature for images [5], [7]–[9], and have been expanded to other data types, such as to explain time-series. The popularity of this perturbation-based approach lies in its intuitiveness, making it one of the promising approaches in XAI.

Based on our observation that the perturbation-based family of methods share common similarities in the process of generating explanations, we propose a novel methodology to decompose any perturbation-based explainability approach into four components. This decomposition serves several purposes. First, it allows a concise and easy analysis of the similarities

between perturbation-based methods, allowing their unification into a common framework of thought; this is done by examining their constitutive components, rather than the methods as a single unit. Second, it facilitates ablation studies and, more broadly, comparisons of variants and modifications of certain components. This allows users to recommend specific explainability methods for specific use cases based on the methods' components. Finally, the decomposition framework fosters the implementation of perturbation-based methods and their benchmarking by allowing the isolation of the effect of individual components.

For practical use of the proposed methodology, we also propose an open-source Python library for explaining ML models Muppet[1]: `Mu`ltiple `Pert`urbations. The library has been developed in a modular design to take advantage of the methods' decomposition. Every method is composed of four blocks (modules), which makes it easy to reuse them in the implementation of other methods. The library offers several functionalities for data scientists to debug and explain their ML models by applying ready-to-use XAI methods alongside their evaluation metrics. It also allows the XAI research community to contribute to an open-source project based on a standardized API, transparent code, and easy implementation of new decomposable methods. The latter aspects allow contributors to speed up the code implementation of their new methods, thanks to the theoretical decomposition and modular design of the library. Furthermore, the library offers benchmarking capabilities for researchers to compare their methods against available ones; facilitating the experimental part of their research.

Outline of paper. The next section II presents work related to perturbation-based explainability methods and existing XAI libraries. Section III details our proposed decomposition framework, as well as the associated Python library Muppet. Section IV highlights examples of the results of our benchmark tool to compare several explainers. Section V concludes by highlighting our main contributions and discussing directions for future work.

II. Related Work

Perturbation-based explainable AI (PXAI): In post-hoc settings, where models are already trained, feature attribution methods generate the importance score of every input feature, which quantifies how much the model relies on the feature to provide its decision [2]. These methods are commonly classified into two main categories [11]. First, model-specific approaches exploit the model's intrinsic characteristics. These include heuristic propagation mechanisms [12] and gradient-based methods [4], [13]. Second, model-agnostic approaches treat the model as a black box and hence apply to any model regardless of its complexity. Most of the proposed post-hoc methods in the literature are based on the concept of perturbing input data [5], [14]. These perturbation-based approaches generate perturbations (with altered features) of the original input and estimate feature importances based on the impact of the perturbation on the model's behavior. To exemplify, in the case of image classification, considerable changes induced by perturbing specific pixels in the input image would lead the classifier to assign the perturbed image to a different class than previously predicted.

In this work, we focus on the family of perturbation-based methods which represent the vast majority of post-hoc methods in the literature. In our framework, we encompass any explanation method that involves data perturbation. For instance, LIME [15] and SHAP [16] methods align seamlessly with our approach. The first generates perturbed inputs around the sample to explain to train a local proxy model, while the second perturbs the sample's features to generate subsets of feature combinations and their values (collisions).

Commonalities of PXAI methods: We have observed that PXAI methods share a common pattern to derive explanations. This observation aligns closely with a theoretical unification of feature-removal explanation methods proposed in [17]. They defined every method by three mapping functions corresponding to 1) how the method removes features, 2) which model behaviour the method explains, and 3) how the method summarizes the influence of each feature.

In our constructive framework, these would correspond respectively to the Perturbation, Attribution, and Aggregation steps. However, this previous work does not address the question of how to determine which feature should be perturbed, which is an essential step in every PXAI method. In practice, PXAI methods explore the input space in the neighbourhood of the input to be explained. It can be done by building masks that are used to define the regions of the input to be perturbed. We can cite two possible strategies: randomly sampling masks [10], [15] or building masks using an iterative optimization approach [7], [18].

In our decomposition, this critical aspect is assigned to the Exploration step which determines how each mask is generated. It is capital to include this Exploration step in any decomposition, as it represents one of the XAI challenges regarding data modalities. Without the Exploration concept, the mapping proposed in [17] cannot clarify the problem of how to pass information from each step of the explanation to the next. The Exploration stage also enables us to suitably address the challenge of handling different modalities (tabular, image, timeseries...) since each modality can be explored differently. Moreover, this decomposition is implemented in a Python library enabling a constructive composition of existing approaches and components, which makes it an important contribution of our work.

Explainability libraries: Numerous XAI methods have been proposed in the literature for various data modalities and tasks. However, they often lack a clean and open-source implementation, which limits their accessibility and usage. To answer this challenge, several libraries have emerged, serving as repositories for XAI methods. Based on their GitHub usage statistics, the top contenders are Captum [19], AI Explainability 360 (AIX360) [20], [21], and OmniXAI [22].

[1]Muppet library: https://github.com/euranova/muppet/

The goal of these libraries is to make as many XAI methods available as possible. However, based on our observation, none addresses the issues of code redundancy or implementation simplicity. Integrating a new method into the existing libraries necessitates a complete and non-reusable implementation, even for methods closely related to each other. This arises from not taking advantage of the inherent similarity among XAI methods and the absence of adopting a modular design that fosters the re-usability of modules across various implementations.

In response to both challenges, we introduce Muppet, an open-source Python framework designed with modularity in mind for PXAI methods. By decomposing each method into reusable components, Muppet streamlines the development of new techniques and their extensions. It offers a standardized API based on the theoretical decomposition, clean and transparent code alongside benchmarking capabilities. Muppet 's objective is to bridge the gap between research and usability in XAI by fostering efficiency, simplicity, and reproducibility.

III. MUPPET : MULTIPLE PERTURBATIONS

A. Theoretical decomposition

In this section, we present our methodology for decomposing PXAI methods. We observed that the PXAI techniques follow the same process for generating explanations for different data types and modeling tasks. This process is mainly based on the concept of perturbing (modifying) the original data and observing the model's behaviour on those perturbed points. The applied perturbations alter the input features (from changing pixel values in an image to altering words in a text for NLP tasks) to assess their importance by measuring their impact on the model's prediction power. The features for which the model's behaviour stays the same when perturbed (e.g. the prediction doesn't change) are considered not important, and vice versa. For example, as shown in Figure 1, the input image is perturbed using random masks, in order to observe the class prediction made by the model under consideration. In this example, the final explanation is in the form of a saliency map which shows important pixels with respect to the model in red; meaning those pixels are essential for the model to predict the correct class (cat) of the original input image.

Our observation leads us to break down the process of PXAI methods into four steps, each one answering a specific question. The first step is the *Exploration*, which determines how to navigate the space of possible perturbations, where a perturbation is defined by a mask covering the elements to be perturbed of the original data. As this space is extremely large, an exhaustive exploration of all possible masks is generally not feasible. The objective at this stage is to identify the process by which these masks are generated in order to be diverse and useful, depending on the original data modality.

The second step is *Perturbation*, which defines the process of modifying the original data using the masks generated in the first step, where the features covered by the masks are perturbed and only them. In this step, the actual perturbed data are created. In the Figure 1 example, a simple pixel-wise multiplication between the mask and the input image is used to create the perturbed images. This is a common way to create perturbations, but in other tasks and for other data types, this step might require more sophisticated ways to perturb the data.

The third step is *Attribution*, which gives the perturbation a value based on an evaluation of the differences in the behaviour of the model when presented with either the perturbed or the original data. Depending on the modeling task and the data modality for the considered explanation method, different strategies could be used. The perturbation value represents the contribution of the applied perturbation in explaining the model's prediction. In the previous example, in Figure 1, the model's behaviour is based on its output ; more precisely, the perturbation's contribution is simply based on the model's logit output for the cat class $f_{cat}(x')$ when given as input the perturbed image. This captures how the model behaves under different perturbations.

The final step is *Aggregation*, which decides how to consolidate the contributions of each assessed perturbation into a coherent final explanation. As several perturbations are applied, we need to define a way to combine the contribution of every perturbation, calculated in the previous step, in the required form of the final explanation. In the example Figure 1, the final explanation is in the form of a saliency map consolidated from the weighted sum of all perturbation masks multiplied by the predicted class probability of each one. The final explanation's form depends on the data modality and the nature of the explanation method.

It is worth mentioning perturbation-based methods are very diverse, and can use different processes than the presented example in Figure 1. For instance, gradient-optimization-based methods search for a minimal perturbation mask that maximally affects the model's output by optimizing a well-defined objective function. In this case, the explanation process follows an iterative strategy to optimize the mask, which results in a different pattern than the one shown in 1. Although the approach is different, it still decomposes easily into our framework: an example of this case is the Meaningful Perturbation method [7] as decomposed in appendix A.

In what follows, we define precisely the mathematical definition of each step presented previously. Let us consider a machine learning model f, which takes an input $x \in \mathbb{R}^D$ and produces an output $y = f(x)$. Here, x represents the D multi-dimensional input vector. Additionally, we define masks $M \in [0,1]^D$ which define whether a feature of x should be perturbed or preserved. Using these notations, PXAI methods always conform to the use of the following steps:

- **Exploration ($\mathcal{E}$):** Exploring the space of possible perturbations. This iterative process generates the masks $M \in [0;1]^D$. This step can be represented as follows:

$$\mathcal{E} : k \rightarrow \mathcal{M}_k$$

  where $\mathcal{M}_k$ is the set of masks generated at step k.
- **Perturbation ($\mathcal{P}$):** Once the masks are defined, they are taken to indicate which parts of the input data x should

be modified by the perturbation function $\mathcal{P}$. This function is defined as:

$$x' = \mathcal{P}(x, M) \in \mathbb{R}^D$$

where x' is the perturbed version of x. This follows $x'_{\mathcal{I}} = x_{\mathcal{I}}$ where $M_{\mathcal{I}} = 0$ and $\mathcal{I}$ is a multidimensional index. In plainer terms, this means that elements whose corresponding value in the mask is 0 should not be perturbed, while a perturbation will be applied to those whose corresponding value is nonzero.

- **Attribution ($\mathcal{A}$):** This step gives a score to each mask via the perturbed inputs it entails. The score is based on the model's behavior towards the perturbed input, and can also include the impact of the masked perturbation on the data itself. This is represented as:

$$A_M = \mathcal{A}(f, x', x, M)$$

where A_M gives the contribution to the explanation from the mask M.

- **Aggregation ($\mathcal{G}$):** The final step involves aggregating the individual attributions obtained from every perturbation (A_M) to derive the final explanation. This step is expressed as:

$$\mathcal{S} = \mathcal{G}(f, \{(A_M, M) | \exists k \in \mathbb{N}, M \in \mathcal{M}_k\})$$

where $\mathcal{S}$ is the final explanation.

The systematic decomposition of PXAI methods into these four steps/components highlights the commonalities between them, such as when several PXAI methods share similar steps of the explanation process. This helps group similar methods, but also facilitates their code implementations by conducting the specific instantiation of their components for the considered methods.

B. Example of LIME's decomposition

Let us now detail the concrete example of the LIME-Image [15] algorithm and its decomposition in the Muppet framework.

- **Exploration:** LIME's exploration strategy is split into two main stages: first the computation of a superpixel segmentation of the image, then masks are built as the association of a fixed number of uniformly drawn superpixels from the first stage.
- **Perturbation:** This step is an occlusion operation defined as:

$$x' = \mathcal{P}_{Mask}(x, M) = x \odot (1 - M) + 0 \odot M$$

where every pixel in the drawn superpixels is set to black.

- **Attribution:** At this step, each perturbed input x' is passed through the model, and the output is collected. The LIME approach also adds a weight depending on the distance to the original example: $\mathcal{A}(f, x, x', M) = (f(x'), \mathcal{D}(x, x'))$ where D is the Radial Basis Function kernel distance.
- **Aggregator:** Conceptually, the previous steps allow the construction of a synthetic and locally-weighted dataset. Then LIME fits an inherently interpretable local model learned from the explored dataset: a LASSO model h is fit on the synthetic dataset to approximate locally f. The returned saliency $\mathcal{S}$ is the sum of all superpixels, each weighted by the learned coefficients of h.

Other decomposition examples for other explanation methods are summarized in Table I and detailed in Appendix A.

C. Implementation

Alongside the proposed theoretical decomposition of PXAI methods, we introduce Muppet (MUltiPle PErTurbations): an open-source Python framework that provides PXAI methods developed in a modular design. The implementation of every method in Muppet contains four modules, namely: *Explorer*, *Perturbator*, *Attributor*, and *Aggregator*, where every one corresponds to the decomposition steps. Muppet's strength lies in its ability to highlight the intersections among these modules in various PXAI methods.

By implementing every method in modular blocks, it fosters a collaborative environment where components can be shared and reused between different methods. As a result, Muppet facilitates the development and implementation of new methods and variations, as compositions of existing components, in few lines of code by the end user. In Table I we present some components' implementations within Muppet, showcasing their reusability by different methods. Special efforts have been made to test the reproducibility of existing methods implemented in Muppet. When possible, we compared the results given by our implementation of a given method with the reference implementation provided by the original authors, to ensure the results of both implementations match.

D. Example of a variant method creation in Muppet

As an illustrative example of the practicality and modularity of Muppet through its decomposition framework, let us consider the implementation of RISE [10] and RELAX [26] in Muppet. These are two distinct perturbation-based methods used for explaining image classification and image embeddings, respectively. We show how the latter can be easily implemented as a variant of the former.

When RISE [10] and RELAX [26] are decomposed following our proposed methodology, it becomes intuitive to find the shared commonalities between these two methods. They differ only for the modelling task and what to capture from the model behaviour when presented with perturbed data. Indeed, RISE and RELAX share three components, namely: their *Explorer*, *Perturbator*, and *Aggregator*, and only differ for the *Attributor* module when generating explanations: RELAX calculates a metric based on an output vector (the embedding), while RISE calculates its metric based on a single scalar (the class probability of the original example). Therefore, a RELAX implementation requires only the development of one

Method / Component	Explorer	Perturbator	Attributor	Aggregator
RISE [10]	Random Grid	Mask	Divergence (Class Score)	Weighted Sum
MP [7]	Optimizer	Blur	Loss	Optimized Saliency
FIT [23]	Indexer	Generator	Divergence (KL)	Sampling Mean
SESS [24]	Patch Grid	Crop + Resize	Partial Saliency	Saliencies Fusion
ScoreCam [25]	Feature Map	Blur	Divergence (Class Score)	Weighted Sum
OptiCam [18]	Optimizer	Blur	Loss	Optimized Saliency
RELAX [26]	Random Grid	Mask	Divergence (Cosine)	Weighted Sum
LIME image [15]	Segmentation	Mask	Preds + Weights (RBF)	Local model
KernelSHAP [27]	Segmentation	Mask	Preds + Weights (Shapley)	Local model

TABLE I: This table presents a concise summary of each of the considered PXAI methods and highlights their intersections thanks to the decomposition. The meaning of the various components is described in Appendix A

component, a new *Attributor*, and reuses all three others from RISE's implementation.

The presented example serves as a demonstration of how effortlessly new methods and variants can be developed in Muppet by assembling distinct but compatible modules tailored for each step in the explanation process. This consistency in components allows Muppet to accommodate various methods while maintaining a consistent modular structure. This flexibility not only streamlines the development process but also encourages innovation by allowing researchers to experiment with different combinations of modules to create new XAI methods. We refer the reader to detailed definitions of the methods' components provided in Appendix A.

E. Benchmarking capabilities

Being able to quickly develop new methods and variants is but one piece of the puzzle. Once new explainers are developed, it is important to be able to evaluate their performance quickly, especially compared to other methods and perhaps even the methods from which the new explainers were derived.

To that end, Muppet incorporates a benchmarking solution. Provided a method is implemented as a Muppet *Explainer* object, it enables the computation of a variety of metrics on several standard datasets and models for different modalities (text, image, time series). We rely on the Quantus [28] library to compute these evaluation metrics.

This benchmarking module was used in the experiments presented in section IV.

IV. Experiments

This section describes the experiments our benchmarking tool developed within the Muppet framework. The goal of these experiments is to illustrate the capabilities of the Muppet library, both in terms of benchmarking and ease of development of new variants.

A. Experimental Protocol

Experiments were conducted on three data modalities:

- Tabular data: using the dataset Dry beans. [29]. We trained a XGboost model on a train set and compute various metrics on the test set.
- Image data: using a subsample of the ImageNet dataset [30]. The evaluation is conducted using the ResNet pretrained model.
- Time series data: multi-channel synthetic spike time series from [23]. A GRU classifier has been trained and we computed explanations using FIT [23] and various configurations of RISE easily adapted to time series. The explanations are then evaluated using our benchmark tool.

For tabular data, we compare the KernelSHAP and LIME explainers, incorporating both Ridge and Lasso regularizations in the surrogate model. This experiment shows how to quickly gain insight by comparing variants.

For image data, we compare several methods which have some building blocks in common : RISE, MP, OptiCAM, and ScoreCAM. This experiment allows us to deduce the relative effects of such building blocks.

For time series data, we compare the FIT method [23] to variants of RISE adapted to time series. For those variants of RISE the upsampling is applied only on the time dimension.

For both data types, we evaluate the performance of the explainers in different aspects thanks to the benchmark capabilities proposed by Muppet. It relies on the explanation evaluation metrics implemented by the Quantus library [28] which categorize them into the following concepts.

Faithfulness This criterion evaluates how accurately the explanations reflect the model's predictions.

- We employ the *Faithfulness Correlation* metric [31], which shows to what extent the predicted logits of each perturbed sample and the average explanation attribution for a subset of features are (linearly) correlated.
- the *Faithfulness estimate* [32] computes the correlation between probability drops and attribution scores on various points
- the *Monotonicity* metric [20] tests if adding more positive evidence increases the probability of classification in the specified class.
- For images we also used the *IROF* metric [33] that computes the area over the curve per class for sorted mean importances of feature segments (superpixels) as they are iteratively removed (and prediction scores are collected)

Robustness This criterion assesses the stability of explanations under slight variations in input. No robustness metric has been added to the proposed experiment examples.

Randomisation This criterion evaluates how explanations degrade as the randomness increases in the data or the model.

We employ the *Random Logit* metric [34], which computes the distance between the original explanation and the explanation for a randomly chosen alternative class.

Complexity This criterion measures the conciseness of explanations. We use the *Complexity* metric [31], defined as the entropy of the fractional contribution of a feature to the total magnitude of the attribution, and the *Sparseness* [35] which relies on the Gini index to quantify whether only highly attributed features are truly predictive of the model output.

The list of metrics is not exhaustive and can be adapted to the needs of the user.

B. Results and analysis

1) Tabular: Results are presented in Figure 3. For each algorithm, namely SHAP and LIME, the results indicate that the performance of the Ridge and Lasso variants within the same algorithm is slightly different but broadly comparable for both.

LIME is surprisingly almost unaffected by the choice of regularization. For SHAP, the answer is more complicated. The original KernelSHAP paper [27], which is the implementation on which we base ours, opts for a Lasso regression, but also states that the regularization term should be zero. In our results, it is possible to quickly see that the choice of regularization can have an impact: switching Lasso to Ridge degrades the performance on all metrics. This suggests that the choice of Lasso in the original paper was motivated by its good sparsity properties that greatly improve the human readability of the explanation, with little to no impact on the faithfulness metrics.

2) Image data: Results are presented in Figure 2. Here, these results highlight potential trade-offs. The more recent OptiCAM variation has a similar Monotonicity and Faithfulness, but a better RandomLogit compared to its predecessor ScoreCAM, at the cost of worse sparsity. But the crucial point is that OptiCAM is an optimisation-based method that shares many building blocks with Meaningful Perturbation, whose own results are considerably worse. Yet, OptiCAM is able to achieve results that are closer to the best method in our benchmark, namely ScoreCAM, but with the same (lower) computing cost as Meaningful Perturbation (MP). Moreover, OptiCAM resolves the robustness problem of MP by constraining the optimization to a combination of the feature maps. Furthermore, we can also see that the RISE method still holds up surprisingly well, though this comes at a heavy computational cost.

3) Time series: Results are presented in Figure 4. Interestingly, it can be observed that the same approach obtains significantly different results depending on its configurations. This encourages limiting the number of parameters in an explainer, or emphasizes the need for efficient parameter-choosing heuristics when developing new explainers. The graph also indicates that the FIT explainer, as a method specifically designed to handle time series, displays significant improvements in terms of Faithfulness and Complexity compared to the ad-hoc RISE variants.

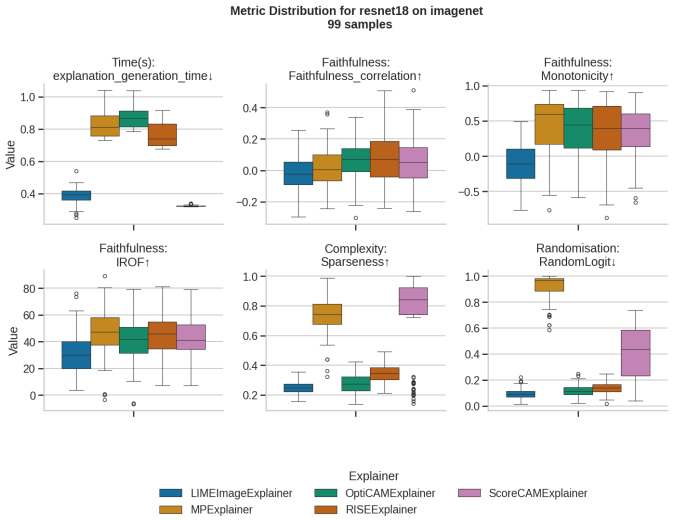

Fig. 2: Image benchmark results display the distribution for various metrics for five image classification model explainers: MP, RISE, OptiCAM, ScoreCAM, LIME Image.

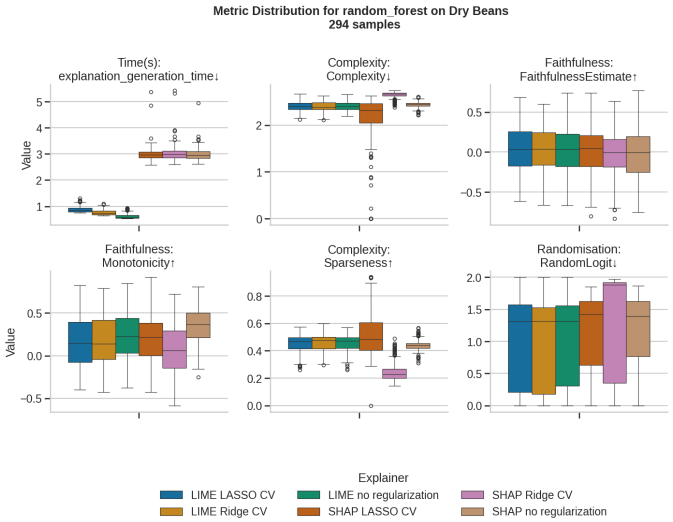

Fig. 3: Tabular benchmark on Random Forest model trained on the Dry Beans dataset. The Results display the distribution for various metrics for two set of explainers variants: LIME [15] and Kernel SHAP [27], with surrogate model without regression, with Ridge regularization or Lasso Regularization.

V. Conclusion and Future Work

In this paper, we proposed a novel methodology to decompose any state-of-the-art perturbation-based explainability (PXAI) approach into four components. This decomposition framework is based on the observation that PXAI methods share the same pattern in generating explanations. We decompose any method into the *Exploration*, *Perturbation*, *Attribution*, and *Aggregation* steps. Our framework allows for a concise analysis of similarities between methods, which leads to their unification and promotes the re-usability of the

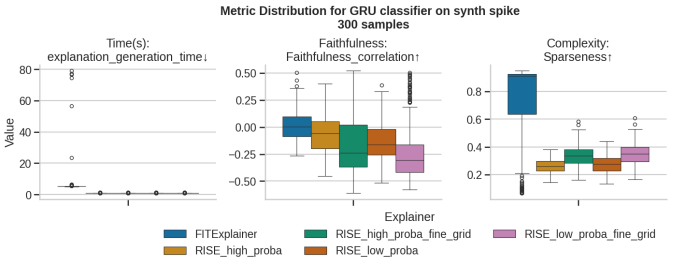

Fig. 4: Timeseries benchmark compares the FIT method [23] to 4 different configurations of RISE adapted to time series. Two main parameters have been modified: the grid size, and the perturbation probability, both parameters hold by the explorer component of RISE.

decomposed components among different methods. Furthermore, we introduce Muppet, an open-source Python library for explaining and debugging ML models. The library is developed in a modular design based on the decomposition framework. Muppet provides end-users with tools to interpret and debug machine learning models, and streamlines the extension of existing methods and the development of new ones: the modular design of Muppet allows the reuse of method components, which makes it easy for users to discover new XAI methods, expands upon existing ones, and speed up implementations while reducing code redundancy. Muppet's benchmarking capabilities, focusing on reproducibility, allow easy comparison between different XAI methods and their variants.

At the time of writing, our framework implements nine approaches from the literature of XAI using multiple shared components. Although significant groundwork has been laid to validate the current decomposition theory, further efforts will be dedicated to implementing more XAI techniques and examining alternative exploration strategies and ML tasks such as image segmentation or dense prediction tasks. Additionally, other responsible AI tools draw on data *perturbation*, such as counterfactual explanations, or analysis of model guarantees including robustness and stability. Following this observation, we plan to broaden our framework beyond simply feature-importance-based XAI methods.

References

[1] A. Holzinger, G. Langs, H. Denk, K. Zatloukal, and H. Müller, "Causability and explainability of artificial intelligence in medicine," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 9, no. 4, p. e1312, 2019.

[2] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information fusion*, vol. 58, pp. 82–115, 2020.

[3] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature machine intelligence*, vol. 1, no. 5, pp. 206–215, 2019.

[4] R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization. arxiv 2016," *arXiv preprint arXiv:1610.02391*, 2022.

[5] R. Fong, M. Patrick, and A. Vedaldi, "Understanding deep networks via extremal perturbations and smooth masks," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 2950–2958, 2019.

[6] I. Stepin, J. M. Alonso, A. Catala, and M. Pereira-Fariña, "A survey of contrastive and counterfactual explanation generation methods for explainable artificial intelligence," *IEEE Access*, vol. 9, pp. 11974–12001, 2021.

[7] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," in *Proceedings of the IEEE international conference on computer vision*, pp. 3429–3437, 2017.

[8] P. Dabkowski and Y. Gal, "Real time image saliency for black box classifiers," *Advances in neural information processing systems*, vol. 30, 2017.

[9] Q. Yang, X. Zhu, J.-K. Fwu, Y. Ye, G. You, and Y. Zhu, "Mfpp: Morphological fragmental perturbation pyramid for black-box model explanations," in *2020 25th International conference on pattern recognition (ICPR)*, pp. 1376–1383, IEEE, 2021.

[10] V. Petsiuk, A. Das, and K. Saenko, "Rise: Randomized input sampling for explanation of black-box models," *arXiv preprint arXiv:1806.07421*, 2018.

[11] W. Ding, M. Abdel-Basset, H. Hawash, and A. M. Ali, "Explainability of artificial intelligence methods, applications and challenges: A comprehensive survey," *Information Sciences*, 2022.

[12] X. Bai, X. Wang, X. Liu, Q. Liu, J. Song, N. Sebe, and B. Kim, "Explainable deep learning for efficient and robust pattern recognition: A survey of recent developments," *Pattern Recognition*, vol. 120, p. 108102, 2021.

[13] M. Ancona, E. Ceolini, C. Öztireli, and M. Gross, "Gradient-based attribution methods," *Explainable AI: Interpreting, explaining and visualizing deep learning*, pp. 169–191, 2019.

[14] M. Ivanovs, R. Kadikis, and K. Ozols, "Perturbation-based methods for explaining deep neural networks: A survey," *Pattern Recognition Letters*, vol. 150, pp. 228–234, 2021.

[15] M. T. Ribeiro, S. Singh, and C. Guestrin, "" 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*, pp. 1135–1144, 2016.

[16] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[17] I. C. Covert, S. Lundberg, and S.-I. Lee, "Explaining by removing: A unified framework for model explanation," *The Journal of Machine Learning Research*, vol. 22, no. 1, pp. 9477–9566, 2021.

[18] H. Zhang, F. Torres, R. Sicre, Y. Avrithis, and S. Ayache, "Opti-cam: Optimizing saliency maps for interpretability," *arXiv preprint arXiv:2301.07002*, 2023.

[19] N. Kokhlikyan, V. Miglani, M. Martin, E. Wang, B. Alsallakh, J. Reynolds, A. Melnikov, N. Kliushkina, C. Araya, S. Yan, *et al.*, "Captum: A unified and generic model interpretability library for pytorch," *arXiv preprint arXiv:2009.07896*, 2020.

[20] V. Arya, R. K. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, S. Houde, Q. V. Liao, R. Luss, A. Mojsilović, *et al.*, "One explanation does not fit all: A toolkit and taxonomy of ai explainability techniques," *arXiv preprint arXiv:1909.03012*, 2019.

[21] V. Arya, R. K. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, S. Houde, Q. V. Liao, R. Luss, A. Mojsilović, *et al.*, "Ai explainability 360: Impact and design," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, pp. 12651–12657, 2022.

[22] W. Yang, H. Le, S. Savarese, and S. Hoi, "Omnixai: A library for explainable ai.(2022)," 2022.

[23] S. Tonekaboni, S. Joshi, K. Campbell, D. K. Duvenaud, and A. Goldenberg, "What went wrong and when? instance-wise feature importance for time-series black-box models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 799–809, 2020.

[24] O. Tursun, S. Denman, S. Sridharan, and C. Fookes, "Sess: Saliency enhancing with scaling and sliding," in *European Conference on Computer Vision*, pp. 318–333, Springer, 2022.

[25] H. Wang, Z. Wang, M. Du, F. Yang, Z. Zhang, S. Ding, P. Mardziel, and X. Hu, "Score-cam: Score-weighted visual explanations for convolutional neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 24–25, 2020.

[26] K. K. Wickstrøm, D. J. Trosten, S. Løkse, A. Boubekki, K. ø. Mikalsen, M. C. Kampffmeyer, and R. Jenssen, "Relax: Representation learning

explainability," *International Journal of Computer Vision*, vol. 131, no. 6, pp. 1584–1610, 2023.
[27] S. M. Lundberg and S. Lee, "A unified approach to interpreting model predictions," *arXiv preprint arXiv:1705.07874*, 2017. Presented at the 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.
[28] A. Hedström, L. Weber, D. Krakowczyk, D. Bareeva, F. Motzkus, W. Samek, S. Lapuschkin, and M. M.-C. Höhne, "Quantus: An explainable ai toolkit for responsible evaluation of neural network explanations and beyond," *Journal of Machine Learning Research*, vol. 24, no. 34, pp. 1–11, 2023.
[29] UCI Machine Learning Repository, "Dry Bean." Online, 2020.
[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet-Sample(1000)." Online, 2009. Accessed via FiftyOne Zoo.
[31] U. Bhatt, A. Weller, and J. M. Moura, "Evaluating and aggregating feature-based model explanations," *arXiv preprint arXiv:2005.00631*, 2020.
[32] D. Alvarez-Melis and T. S. Jaakkola, "Towards robust interpretability with self-explaining neural networks," 2018.
[33] L. Rieger and L. K. Hansen, "Irof: a low resource evaluation metric for explanation methods," *arXiv preprint arXiv:2003.08747*, 2020.
[34] L. Sixt, M. Granz, and T. Landgraf, "When explanations lie: Why many modified bp attributions fail," in *International conference on machine learning*, pp. 9046–9057, PMLR, 2020.
[35] P. Chalasani, J. Chen, A. R. Chowdhury, X. Wu, and S. Jha, "Concise explanations of neural networks using adversarial training," in *International conference on machine learning*, pp. 1383–1391, PMLR, 2020.

APPENDIX

In addition to the decompositions of some explanation methods summarize in Table I, we detailed here the definition of the components on which the decomposition relies.

- **Explorer**: Exploring the space of possible perturbation.
  - *Random Grid*: The random grid explorer draws a set of small multidimensional vectors filled with Bernoulli variables, which are then up-sampled with interpolation: $\forall M \in \mathcal{M}_{Random}, M = \mathcal{U}(G)$ where $G_{\mathcal{I}} \sim Bernoulli(p)$
  - *Optimizer*: This explorer produces masks that are the iterates of the optimization of a given objective function, relying usually on the model output regularized on characteristics of the mask iterates M^t. For example, MP [7] uses the total variation to regularize the objective function.
  - *Indexer*: $\mathcal{M}_{idx} = \{\mathbf{1}_{\mathcal{I}_1}, \mathbf{1}_{\mathcal{I}_2}, ..., \mathbf{1}_{\mathcal{I}_k}\}$, where every $\mathbf{1}_{\mathcal{I}_k}$ is a vector full of 0 except at multidimensional index $\mathcal{I}_1$.
  - *Patch Grid*: $\mathcal{M}_{Patch} = \{P_1, ..., P_N\}$ which are scaled sliding windows on the input.
  - *Feature Map*: This explorer uses upsampled versions of the feature maps from the last convolutional layer (l) of f: $\mathcal{M}_{FeatureMap} = \{\mathcal{U}(f_i^l(x))|\forall i \in [1..C_l]\}$
  - *Segmentation*: Some Explorers may rely on other algorithms to segment the input data, such as superpixel segmentation for images.
- **Perturbator**: Perturbing using pure functions that take as input the original model input and a mask
  - *Mask*: $\mathcal{P}_{Mask}(x, M) = x \odot (1 - M) + \bar{x} \odot M$, where $\odot$ is the element-wise multiplication, and $\bar{x}$ can be any constant reference vector of the size of x.
  - *Blur*: $\mathcal{P}_{Blur}(x, M) = x \odot (1 - M) + x * K \odot M$ where $*$ is the convolution operation between x and the Gaussian kernel K.
  - *Crop and Resize*: Crop the input according to the mask and resize it to the original input size.
  - *Generator*: $\mathcal{P}_{Generator}(x, M) = x \odot (1 - M) + G_M \odot M$ where the values of G_M are sampled from a generative model G.
- **Attributor**: Extract information from the model's behavior on the perturbed input.
  - *Divergence*: A similarity measure capturing the deviation of the model's output when presented to the perturbed input. For example, $\mathcal{A}(f, x, x', M) = f(x) - f(x')$ the predicted class score on the perturbed inputs in RISE [10], or the KL divergence in FIT [23], or Cosine Similarity based in RELAX [26].
  - *Loss*: $\mathcal{A}(f, x, x', M) = Loss(J(M_t, f, x', x))$ captures the loss of an objective function J which can be used by optimization based explorers.
  - *Partial Saliency*: It calculates $\mathcal{A}(f, x', x, M) = \mathcal{S}_{XAI}(f, x')$ a partial saliency map using any chosen XAI method $\mathcal{S}_{XAI}$, as well as the class score.
  - *Weight and predict*: Some approaches attribute to each candidate perturbation a weight based on its similarity to the base input, as well as the model's output for the perturbed input. For example, LIME [15] uses Radial Basis Function (RBF) similarity and KernelSHAP [16] a weighting based on estimated Shapley values.
- **Aggregator**: Consolidate the contributions from each attributed perturbation into a final explanation.
  - *Weighted Sum*: This aggregator computes the weighted mean of the explored masks $\mathcal{S} = \sum_{M \in \mathcal{M}} A_M \times M$, using the attribution as weights.
  - *Sampling Mean*: $\mathcal{S} = \sum_{M \in \mathcal{M}} (\frac{1}{N} \sum_{n=1}^{N} A_M^n) \times M$ where N is the number of sampled perturbed inputs generated for a mask $M \in \mathcal{M}$ in a Monte Carlo approach.
  - *Optimized Saliency*: For the optimization-based procedure, the final saliency is often the last iterate of the optimization algorithm: $\mathcal{S} = M^t$.
  - *Saliencies Fusion*: It calculates the weighted and localized average of the saliency attributions: $\mathcal{S} = \sum_{M \in \mathcal{M}} f(x'_M) \times \mathcal{S}_{XAI}(x'_M)$ where x'_M is the perturbed images for mask M.
  - *Local Model*: Methods such as LIME [15] or SHAP [16] learn a local and explainable model based on the synthetic weighted dataset composed of the perturbations and their corresponding attributed weights and model outputs. The explanation of this surrogate local model serves as an explanation for the input prediction.