

ForestMonkey: Toolkit for Reasoning with AI-based Defect Detection and Classification Models

Jiajun Zhang, Georgina Cosma*

Dept of Computer Science, School of Science

Loughborough University, UK

Email: j.zhang8@lboro.ac.uk; g.cosma@lboro.ac.uk*

* Corresponding author

Sarah Bugby

Dept of Physics, School of Science

Loughborough University, UK

Email: s.bugby@lboro.ac.uk

Jason Watkins

Railston & Co. Ltd.

Nottingham, UK

Email: jason@railstons.com

Abstract—Artificial intelligence (AI) reasoning and explainable AI (XAI) tasks have gained popularity recently, enabling users to explain the predictions or decision processes of AI models. This paper introduces *Forest Monkey (FM)*, a toolkit designed to reason the outputs of any AI-based defect detection and/or classification model with data explainability. Implemented as a Python package, FM takes input in the form of dataset folder paths (including original images, ground truth labels, and predicted labels) and provides a set of charts and a text file to illustrate the reasoning results and suggest possible improvements. The FM toolkit consists of processes such as feature extraction from predictions to reasoning targets, feature extraction from images to *defect characteristics*, and a decision tree-based AI-Reasoner. Additionally, this paper investigates the time performance of the FM toolkit when applied to four AI models with different datasets. Lastly, a tutorial is provided to guide users in performing reasoning tasks using the FM toolkit.

Index Terms—Morphological analysis, AI-Reasoner, defect characteristics, explainable AI.

I. INTRODUCTION

Object detection and classification tasks using *artificial intelligence (AI)* are popular in industry applications and beyond. The outputs of AI models need to be explained to foster trust in the results during decision making. *Explainable AI (XAI)* techniques offer solutions for explaining and reasoning the outcomes of AI models, leading to the development of various XAI toolkits. One such toolkit is XAI360 [1], published by Arya *et al.* of the *International Business Machines Corporation (IBM)* in 2020, which integrates 14 diverse XAI methods.

In 2023, Ali *et al.* [2] conducted a survey on a set of XAI techniques and proposed four XAI taxonomies: scoop-based, model-based, complexity-based, and methodology-based, along with their corresponding use cases. Scoop-based XAI techniques, also known as data explainability methods [3]–[5], analyse the features extracted from the data to establish the relationship between the input and output of the AI model. Model-based XAI techniques, such as model explainability methods [6]–[8], break down the AI model into steps and provide explanations for each step. Complexity-based XAI techniques, such as intrinsic interpretable models (e.g. decision trees, Bayesian models, linear models, and k-nearest neighbours), offer explanations with varying levels

This research is funded through joint funding by the School of Science at Loughborough University with industrial support from Railston & Co Ltd.

of detail based on the model’s complexity. Methodology-based XAI techniques, referred to as post-hoc explainability methods [9]–[11], interpret the AI models by analysing the model’s backpropagation routes or perturbation signals. Zhang *et al.* [12] developed an AI reasoning framework that enables reasoning capabilities for the outputs of an AI-based detection and/or classification model. The framework incorporates a proposed feature extraction method known as *defect characteristics (DefChars)* and an ensemble *decision tree (DT)*.

In this paper, a toolkit named *Forest Monkey (FM)* is introduced, which implements Zhang *et al.*’s [12] framework using the Python programming language. The FM toolkit provides reasoning for predictions made by an AI-based detection and/or classification model. The FM toolkit takes as input images, ground truth labels, and predicted labels from the AI model. Subsequently, the toolkit converts the predicted labels into reasoning targets and transforms the images into a DefChars matrix. Then, the toolkit generates the AI reasoning results, which include a set of charts and text-based improvement suggestions. By examining the reasoning results, users can gain a deeper understanding of their dataset and make improvements to their AI model.

In this paper, Section II provides an overview of the structure of the FM toolkit. Section III explains the implementation details of the toolkit. Section IV applies the toolkit to four different AI-based models using diverse datasets and discusses the performance in terms of execution time. Finally, Section V presents a tutorial on using the toolkit and provides an explanation of the output generated by the toolkit. The code for the FM toolkit and tutorial is provided in <https://github.com/edgetrier/AI-Reasoner>.

II. TOOLKIT OVERVIEW

The FM toolkit is a Python-based package library. Figure 1 illustrates the architecture overview of the FM toolkit. Zhang *et al.*’s [12] AI-Reasoner serves as the foundation for the FM toolkit, which is a post-hoc and model-agnostic framework with data explainability. As a result, the toolkit only requires input images, ground truth labels and predicted labels from the AI model. The outputs of the toolkit are a set of charts and text-based improvement suggestions, which are stored as *png*

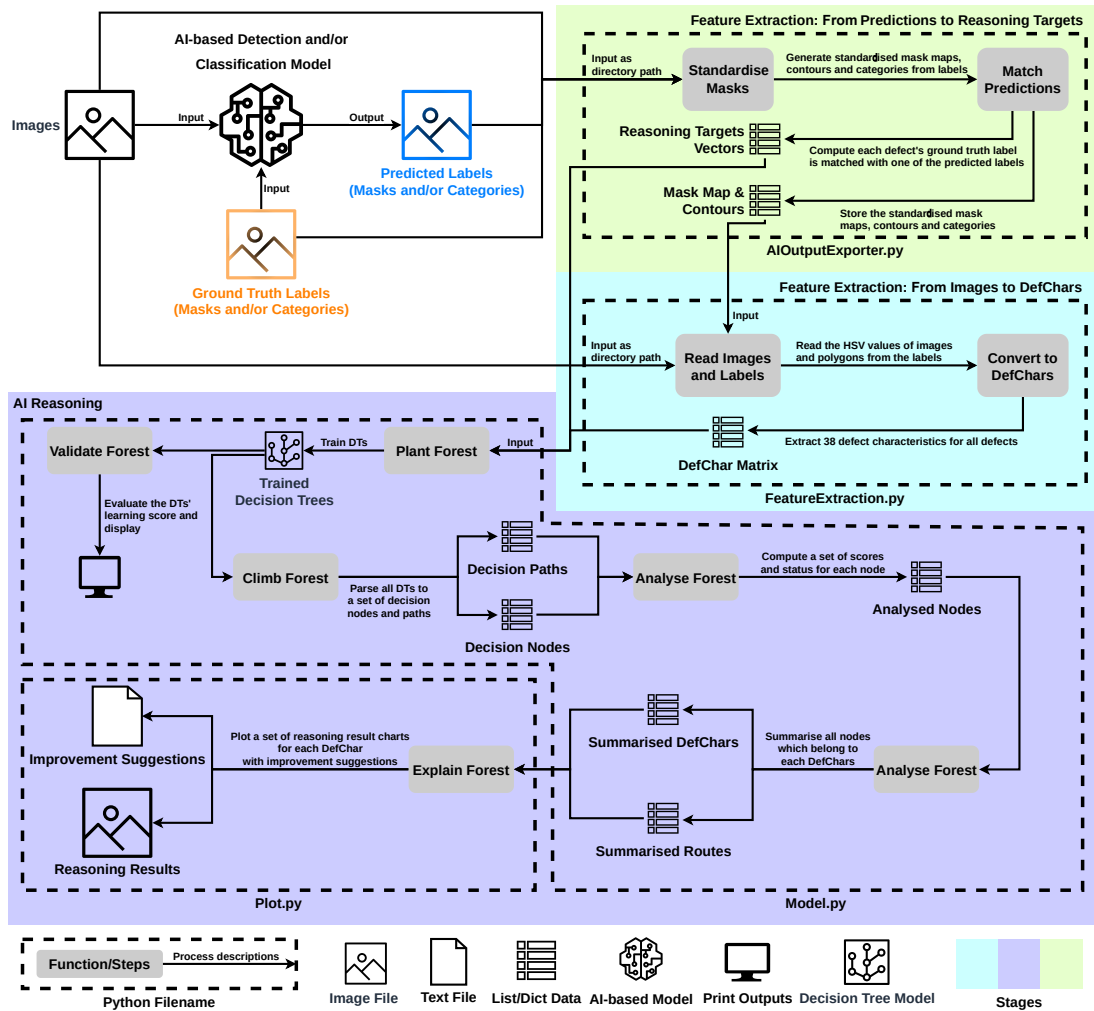


Fig. 1. FM toolkit structure overview.

and *txt* formatted files, respectively. The FM toolkit comprises three stages to perform an AI reasoning task:

- Feature extraction from predictions to reasoning targets: This stage converts all predictions into a set of reasoning target vectors based on the AI model’s prediction tasks, such as detection, classification, or joint detection and classification. The reasoning targets include labels such as “detected”, “undetected”, “correctly classified”, and “misclassified”. Additionally, this stage standardises the mask-based labels to maintain consistent mask maps and contours of the defect regions.
- Feature extraction from images to DefChars: In this stage, all defect images are processed to generate a DefChar matrix. This is achieved by reading and processing the hue, saturation, and brightness (HSV) values and mask maps of the images.
- AI reasoning stage: This stage takes the DefChar matrix and reasoning target vectors as input to analyse the importance of each DefChar in causing correct or incorrect predictions by the AI model. It involves six steps:

plant forest, *validate forest*, *climb forest*, *analyse forest*, *summarise forest*, and *explain forest*. Finally, this stage generates improvement recommendations in the form of text and visualisations to explain the results and provide reasoning to users.

III. IMPLEMENTATION

The FM toolkit is implemented using various packages, including *scikit-learn*, *OpenCV*, *NumPy*, *tqdm*, *matplotlib*, *shapely*, *pillow*, and *polygenerator*.

- *NumPy* for processing array-based and matrix-based data.
- *OpenCV* and *pillow*: image data processing, including computing the mask map and contours.
- *Shapely* and *polygenerator*: converting polygon contours into shape-based DefChars.
- *scikit-learn*: DT model implementation that is utilised for the AI reasoning functionalities.
- *matplotlib*: generating the reasoning result charts.

The hardware requirements for running the FM toolkit include at least 8 gigabytes of RAM and any Intel/AMD CPU; a GPU is not required.

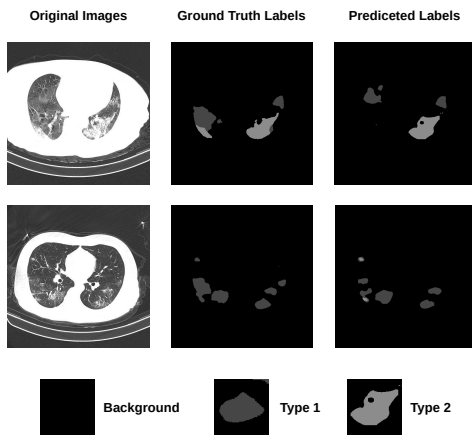


Fig. 2. Example of input data for FM toolkit, the real values in the highlighted area are the categories of defects $C \in \{1, 2\}$.

A. Data Preparation

A classic dataset usually consists of pre-processed images and their corresponding mask-based ground truth labels (stored in *png* format) when applied in an AI model. Once the detection and/or classification tasks are performed by the AI model, its prediction results need to be converted into mask-based labels, mirroring the format of the ground truth labels. Figure 2 provides an example of the input data structure. The images, ground truth labels and predicted labels should be stored in three separate folders.

B. Feature Extraction: From Predictions to Reasoning Targets

```
def process_dir(imgs_path, gt_masks_path,
               predicted_masks_path, contain_type, only_type):
    ...
    return label_data={"mask map": [...],
                      "contours": [...],
                      "reasoning_targets": {...}}
```

This stage involves the extraction of reasoning target vectors from AI predictions, as well as the mask map and contours from the ground truth labels. This can be accomplished by calling the *process_dir()* function; the required inputs for this function are:

- The path to the images folder.
- The path to the ground truth labels folder.
- The path to the predicted labels folder.

Additionally, two Boolean values, *contained_type* and *only_type*, are required to specify the extraction of reasoning target vectors for different tasks, such as a detection task (*contained_type = False*, *only_type = False*), a classification task (*contained_type = True*, *only_type = True*), or a joint detection and classification task (*contained_type = True*, *only_type = True*). The output of the function is a dictionary that contains the mask maps, contours, and reasoning target vectors for all ground truth defects.

C. Feature Extraction: From Images to DefChars

```
def checkImage(imgs_path):
    ...
def readLabel(label_dict):
    ...
def loadData():
    ...
def featureExtract():
    ...
    return defchar={"defect_1": {...},
                   "defect_2": {...}, ...}
```

This stage involves a sequence of four functions: *checkImage()*, *readLabel()*, *loadData()*, and *featureExtract()*. The *checkImage()* function is responsible for loading the images, while the *readLabel()* function loads the labels, which include mask maps and contours. The *loadData()* function reads the images and labels and creates a dictionary with basic information about the images and labels. The *featureExtract()* function is utilised to extract the DefChars and stores them in the dictionary. These functions are called in order to extract the DefChars from the images.

D. Pre-processing before AI Reasoning

```
def convert2List(defchar, feature_list,
                 label_dict):
    ...
    return id_list=[defect_id, ...],
           feature_list,
           data=[[defchar 1], [defchar 2], ...],
           targets={"detected": [...],
                   "undetected": [...], ...}
def load_feature_data(data):
    ...
def load_target_data(targets, target_name):
    ...
```

In this section, there are three utility functions (i.e. *convert2List()*, *load_feature_data()*, and *load_target_data()*) implemented in the *Model.py* file. The *convert2List()* function returns several array-based lists derived from the feature extraction stage, including an id list, a feature list, a DefChars matrix and reasoning target vectors. The *load_feature_data()*, and *load_target_data()* functions are utilised to correspondingly load and prepare the feature and reasoning target data for the AI-Reasoner model. These functions are responsible for converting the dictionary-based data into array-based data and loading it into the AI-Reasoner model.

E. Plant Forest

```
def plant_forest(n_tree=200):
    ...
    return model=[DT_1, DT_2, ...]
```

The *plant_forest()* function is responsible for building and training multiple DT models using the loaded data. The function allows for an optional input to specify the desired number of DT models, which determines the number of decision rules

generated. By invoking the `plant_forest()` function, the DT models are constructed and trained using the loaded data.

F. Validate Forest

```
def val_forest(model, feature_list):
    ...
    return good_learned=bool(),
           eval_[learning_score, TPR, TNR],
           error_feature=[...]
```

The `val_forest()` function is responsible for evaluating the learning capability of the AI-Reasoner. It computes the overall learning scores, including true positive rate (TPR) and true negative rate (TNR), by averaging the learning scores of each trained DT model. Additionally, the function provides a set of DefChars where these DefChar made mistakes during the reasoning task.

G. Climb Forest

```
def climb_forest(model, feature_list):
    ...
    return path=[...], node=[...], route=[...]
```

The `climb_forest()` function is responsible for parsing all decision nodes and paths from the trained DT models. It stores the parsed information in three lists: decision paths, routes, and decision nodes. Additionally, the list of DefChars names is required for the parsing process; the list can be accessed by calling `FeatureExtraction.feature_list`.

H. Analyse Forest

```
def analyse_forest(path, node, error):
    ...
    return analysed_node=[...]
```

The `analyse_forest()` function takes the parsed decision nodes and paths as input. It then computes a set of values for each node, which helps analyse the importance of the node in reasoning the AI predictions. The function outputs a list that extends the input node list with the computed values; this extended list provides additional information and insights about each node, allowing for a more in-depth analysis of the reasoning process.

I. Summarise Forest

```
def summary_forest(analysed_node, feature_list,
                  route, feature_range):
    ...
    return summary={"defchar 1":{...}, ...},
           route_to_1=[...],
           route_to_0=[...]
```

The `summary_forest()` function computes the importance for each DefChar to reflect the importance of each DefChar in influencing the AI predictions. It takes several inputs, including the analysed nodes list, parsed route, DefChars

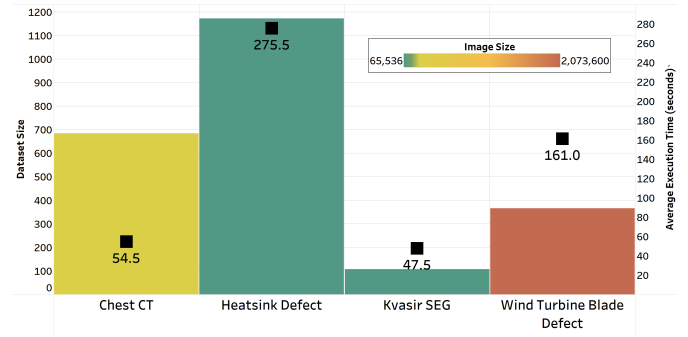


Fig. 3. Average execution time of the FM toolkit for different datasets. The black dots represent the average execution time, indicated on the right axis. The bars represent the dataset size, indicated on the left axis, and the colours of the bars correspond to the dataset's image size, as shown in the legends.

list, and value ranges. The function outputs a dictionary that contains the computed scores for each DefChar to provide a comprehensive overview of the importance of each DefChar.

J. Explain Forest

```
def explain_forest(report, feature_list,
                  save_path, route_plot=None):
    ...
```

The `explain_forest()` function is responsible for generating a set of charts and providing improvement suggestions based on the summarised overviews from the previous stage. The input includes the generated dictionary from `summary_forest()` function, DefChars list and folder path where the reasoning results will be saved. Furthermore, a set of important decision routes can be optionally plotted by setting `route_plot` as true. The reasoning result charts are saved in the `png` format, while the improvement suggestions are stored in a `txt` format file.

IV. PERFORMANCE

In this section, the FM toolkit was applied to four different AI-based defect detection and classification models using different datasets. The models include a *COVID-CT-mask-net* trained on a chest CT image dataset [13], a *lightweight fully convolutional network* trained on a heatsink defect image dataset [14], a *deep residual U-Net++* trained on the Kvasir-SEG dataset [15], and an *image-enhanced mask R-CNN* trained on a wind turbine blade defect image dataset [16]. Table I presents the defect distributions of these four datasets, providing an overview of the reasoning targets and quantities of defects present in each dataset. Additionally, the running time in each step of the FM toolkit was recorded to analyse its performance. This allows for an assessment of the toolkit's efficiency and provides insights into potential areas for improvement. A computer with an AMD 16-core CPU and 32 gigabytes RAM was used to evaluate performance in execution time.

Figure 3 illustrates the execution times when applying the FM toolkit to different datasets. It shows that the execution time is positively correlated with the dataset size and image

TABLE I
INFORMATION OF FOUR DATASETS; WHERE N/A REPRESENTS NO SUCH TASKS.

| Dataset | Image Size | Number of Defects for Each Reasoning Target (percentage) | | | | Total |
|---------------------------|------------|--|------------|----------------------|---------------|-------|
| | | detected | undetected | correctly classified | misclassified | |
| Chest CT | 512×512 | 244 | 147 | 215 | 29 | 391 |
| Heatsink Defect | 256×256 | 804 | 368 | 774 | 30 | 1172 |
| Kvasir-SEG | 256×256 | 104 | 3 | N/A | N/A | 107 |
| Wind Turbine Blade Defect | 1920×1080 | 311 | 55 | 296 | 15 | 366 |

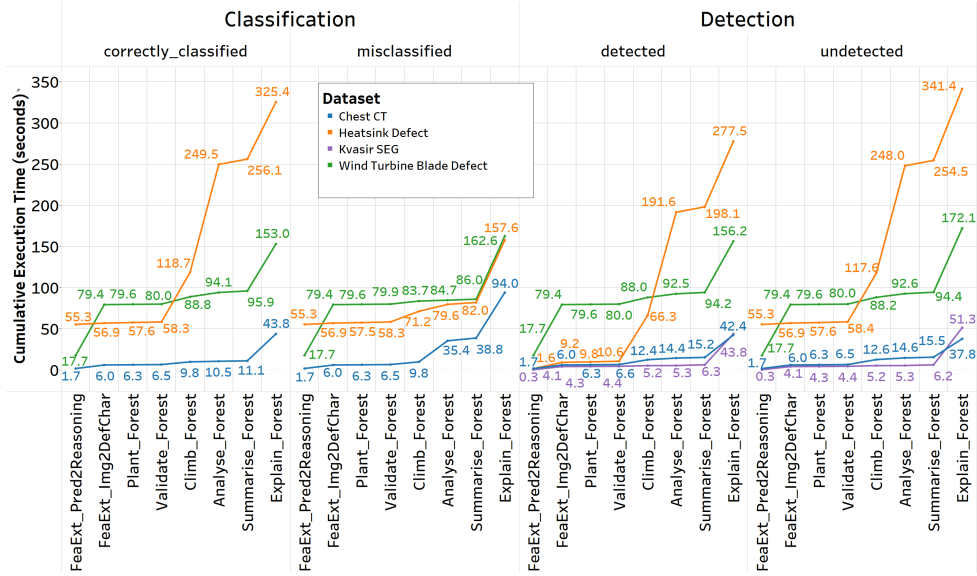


Fig. 4. Cumulative execution time of each stage in the FM toolkit for reasoning targets of different datasets.

size, indicating that larger datasets or images require longer execution times. Figure 4 illustrates the cumulative execution times of each stage for different reasoning targets. Among the various stages of the FM toolkit, *feature extraction from images to DefChars*, *climb forest*, *analyse forest*, and *explain forest* consumed relatively more time compared to other stages. In summary, the FM toolkit can complete a reasoning task in at least 40 seconds. However, the execution time may increase when dealing with datasets containing large amounts or large-size images.

V. TUTORIAL

This section provides a tutorial for adopting the FM toolkit to reason with the outputs of an AI model.

Step 1: Import the FM toolkit.

```
from ForestMonkey import AIOutputExporter
from ForestMonkey import FeatureExtraction
from ForestMonkey import Model
from ForestMonkey import Plot
```

Step 2: Set the directory paths of original images, ground truth labels and predicted labels from the AI model; the related data preparation is described in Section III-A.

```
images="/path/.../"
gt_labels="/path/.../"
predicted_labels="/path/.../"
```

Step 3: Feature extraction: From predictions to reasoning targets

```
contain_type=True
type_only=False
label_data=AIOutputExporter.process_bydir(
    images,gt_labels,predicted_labels)
```

Step 4: Feature extraction: From images to DefChars

```
FeatureExtraction.checkImages(images)
FeatureExtraction.readLabel(label_data)
FeatureExtraction.loadData()
defchar=FeatureExtraction.featureExtract()
```

Step 5: Pre-processing before AI reasoning

```
feature_list = FeatureExtraction.feature_list
id_list,feature,data,target=Model.convert2List(
    (defchar,feature_list,label_data)
Model.load_feature_data(data)
```

Step 6: Execute AI reasoning task for all reasoning targets

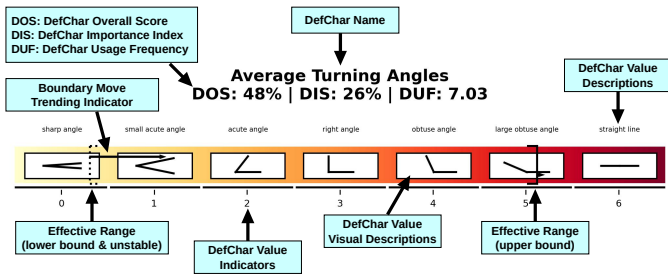


Fig. 5. An example chart illustrating one of the DefChar's reasoning results generated by the FM toolkit; the blue textual boxes represent descriptions of each element in the chart.

```

for t in target.keys():
    Model.load_target_data(target[t],t)
    model = Model.plant_forest()
    tree_validated,scores,errors=Model.
    val_forest(model,feature)
    path,node,route=Model.climb_forest(model,
    feature_name=feature)
    analysed_node=Model.analyse_forest(path,
    node,error)
    report,route_t,route_nt=Model.
    summary_forest(analysed_node,route,feature,
    FeatureExtraction.get_FeatureRange())
    Plot.explain_forest(report,feature,
    "/directory/path/to/save/"+t)

```

After completing these six steps, the reasoning results will be saved in the specified directory. Figure 5 illustrates an example chart generated by the FM toolkit. The chart allows users to understand the significance of each DefChar and its value range in influencing the AI model's correct or incorrect predictions in detection and/or classification tasks. Additionally, users can refer to the *improvement_recommendations.txt* file for improvement suggestions on how to enhance their dataset and model based on the reasoning results.

VI. CONCLUSION

This paper presents the integration of Zhang et al.'s [12] AI-Reasoner framework into a toolkit called FM, implemented in Python. The FM toolkit can be easily used by importing it as a Python package, and a detailed tutorial is provided to guide users in utilising the toolkit effectively. Furthermore, the FM toolkit is evaluated by applying it to four different AI-based models with diverse datasets to assess its execution performance. In terms of future work, several enhancements are suggested for the FM toolkit. Firstly, the implementation of GPU-enabled parallel computations could be explored to accelerate the execution speed. Additionally, the development of interactive interfaces would enhance user experience and make the toolkit more user-friendly. Furthermore, visualisations of the DefChars could be incorporated to provide users with a better understanding of the reasoning process.

REFERENCES

[1] V. Arya, R. K. E. Bellamy, P.-Y. Chen, A. Dhurandhar, M. Hind, S. C. Hoffman, S. Houde, Q. V. Liao, R. Luss, A. Mojsilović, S. Mourad,

P. Pedemonte, R. Raghavendra, J. Richards, P. Sattigeri, K. Shanmugam, M. Singh, K. R. Varshney, D. Wei, and Y. Zhang, "One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques," *arXiv e-prints*, p. arXiv:1909.03012, Sep. 2019.

[2] S. Ali, T. Abuhmed, S. El-Sappagh, K. Muhammad, J. M. Alonso-Moral, R. Confalonieri, R. Guidotti, J. Del Ser, N. Díaz-Rodríguez, and F. Herrera, "Explainable Artificial Intelligence (XAI): What we know and what is left to attain Trustworthy Artificial Intelligence," *Information Fusion*, p. 101805, 2023. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S1566253523001148>

[3] S. Tilouche, V. Partovi Nia, and S. Bassetto, "Parallel coordinate order for high-dimensional data," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 14, no. 5, pp. 501–515, 2021. [Online]. Available: <https://onlinelibrary.wiley.com/doi/abs/10.1002/sam.11543>

[4] W. J. Murdoch, C. Singh, K. Kumbier, R. Abbasi-Asl, and B. Yu, "Definitions, methods, and applications in interpretable machine learning," *Proceedings of the National Academy of Sciences*, vol. 116, no. 44, pp. 22071–22080, 2019. [Online]. Available: <https://www.pnas.org/doi/abs/10.1073/pnas.1900654116>

[5] T. Campbell and T. Broderick, "Automated Scalable Bayesian Inference via Hilbert Coresets," *Journal of Machine Learning Research*, vol. 20, no. 15, pp. 1–38, 2019. [Online]. Available: <http://jmlr.org/papers/v20/17-613.html>

[6] M. Hind, D. Wei, M. Campbell, N. C. F. Codella, A. Dhurandhar, A. Mojsilović, K. Natesan Ramamurthy, and K. R. Varshney, "TED: Teaching AI to Explain Its Decisions," in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 123–129. [Online]. Available: <https://doi.org/10.1145/3306618.3314273>

[7] M. Al-Shedivat, A. Dubey, and E. Xing, "Contextual Explanation Networks," *J. Mach. Learn. Res.*, vol. 21, no. 1, jan 2020.

[8] C. Chen, O. Li, D. Tao, A. Barnett, C. Rudin, and J. K. Su, "This Looks Like That: Deep Learning for Interpretable Image Recognition," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019.

[9] Lundberg, Scott M and Lee, Su-In, "A Unified Approach to Interpreting Model Predictions," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[10] 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*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1135–1144. [Online]. Available: <https://doi.org/10.1145/2939672.2939778>

[11] A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin, "Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation," *Journal of Computational and Graphical Statistics*, vol. 24, no. 1, pp. 44–65, 2015. [Online]. Available: <https://doi.org/10.1080/10618600.2014.907095>

[12] J. Zhang, G. Cosma, S. Bugby, A. Finke, and J. Watkins, "Morphological Image Analysis and Feature Extraction for Reasoning with AI-based Defect Detection and Classification Models," *arXiv e-prints*, p. arXiv:2307.11643, Jul. 2023.

[13] A. Ter-Sarkisov, "COVID-CT-Mask-Net: Prediction of COVID-19 from CT Scans Using Regional Features," *Applied Intelligence*, vol. 52, p. 9664–9675, 2022.

[14] K. Yang, Y. Liu, S. Zhang, and J. Cao, "Surface Defect Detection of Heat Sink Based on Lightweight Fully Convolutional Network," *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1–12, 2022.

[15] D. Jha, P. H. Smedsrud, M. A. Riegler, D. Johansen, T. Lange, P. Halvorsen, and H. D. Johansen, "ResUNet++: An Advanced Architecture for Medical Image Segmentation," in *2019 IEEE International Symposium on Multimedia (ISM)*. Los Alamitos, CA, USA: IEEE Computer Society, dec 2019, pp. 225–2255. [Online]. Available: <https://doi.ieeeecomputersociety.org/10.1109/ISM46123.2019.00049>

[16] J. Zhang, G. Cosma, and J. Watkins, "Image Enhanced Mask R-CNN: A Deep Learning Pipeline with New Evaluation Measures for Wind Turbine Blade Defect Detection and Classification," *Journal of Imaging*, vol. 7, no. 3, 2021. [Online]. Available: <https://www.mdpi.com/2313-433X/7/3/46>