A semi-autonomic framework for developing Machine Learning-based applications using Fairness metrics

ABSTRACT

The increasing use of Machine Learning (ML) in digital solutions has created a need for new algorithms and fairness metrics to ensure fairer decissions. However, this has made data scientist analysis more complex, due to the need to balance fairness metrics with traditional evaluation metrics and a greater variety of algorithms. This work presents a semi-autonomous framework to train machine learning models that identifies more optimized configurations across different contexts. The framework uses a module using the *Pipes-and-Filters* architecture with several pre-implemented algorithms and a module using the MAPE-K architecture to evaluate the ideal balance. Several case studies were conducted to determine the feasibility and extensibility of the proposed framework. The results showed that the framework can help data scientists better understand the training process and enable software engineering studies to help train trustworthy machine learning models.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **Computer systems organization** → **Self-organizing autonomic computing**; *Pipeline computing*; Maintainability and maintenance; • **Software and its engineering** → *Software product lines*; Software evolution.

KEYWORDS

Machine Learning, Artificial Intelligence Ethics, Social Responsibility in Artificial Intelligence, Autonomic Computing, Fairness Metrics

ACM Reference Format:

. 2024. A semi-autonomic framework for developing Machine Learning-based applications using Fairness metrics. In *Proceedings of ACM SAC Conference (SAC'24).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/xx.xxx/xxx_x

1 INTRODUCTION

Artificial intelligence (AI) and machine learning (ML) techniques have been used for many years in computer science, particularly in areas such as robotics and games, where they can automate tasks and generate insights from large volumes of data. However, these applications can also introduce biases that reflect existing prejudices in today's society. Biased data entries result in an algorithm that performs discriminations in its classification [8], and evaluation metrics typically evaluate the entire dataset rather than specific groups, so it can be difficult to detect bias in the results.

Due to this problem, it is possible to establish different metrics to determine how well the model is prepared for sensitive data [6], a term that is known as Fairness. The main goal of Fairness is to ensure that privileged and non-privileged groups, regardless of their protected attributes, are treated equally, as evidenced by similar metrics. Advances in academic research on fairness have led to new methods for reducing biases present in datasets, such as *Reweighing* [13], *Adversarial Debiasing* [20] and *Reject Option Classification* [14]. While these methods have improved fairness metrics, they may disfavor other metrics that are widely used to assess the effectiveness of machine learning models.

For this context, a framework for developing Machine Learning applications was developed [1], which is handled by Data Scientists in a semi-autonomous way, focused on two main objectives:

- Facilitate the creation of fair and reliable models by automating the following process: Data preparation, training and evaluation.
- Establish a balance between Evaluation metrics and Fairness metrics to efficiently obtain fairer models.

This structure is divided into 4 modules: Data Engineering, ML Module, Autonomic Manager, and User Interface. The Data Engineering module prepares the data for the ML Module by modifying the German Credit Dataset [2] and Lendingclub Dataset [1] datasets. The ML Module is developed using the *Pipes-and-Filters* architecture. For the Autonomic Manager, the MAPE-K [3] architecture analyzes a knowledge repository to identify the best architectural configuration for the ML Module. The User Interface is a web application template.

2 THE PROPOSED FRAMEWORK

2.1 Architecture and Implementation

The framework architecture is divided into 4 modules:

- **Data Engineering:** Module with the objective of executing data transformation and cleaning processes.
- **ML Module:** Module that executes a Pipeline of an automated ML application, with stages of data preparation (Preprocessing), training (Processing) and result evaluation (Postprocessing) of the final model generation.
- **Autonomic Manager:** Module containing a MAPE-K control *loop* that controls the ML Module as a Managed Element to automate all its steps.
- **User Interface):** Module which goal is to provide a simpler and more intuitive user experience for the Data Scientist in order to configure and start the Autonomic Manager.

The integration between these modules is illustrated in Figure 1. The Data Engineering module is used by the Data Engineer to

SAC'24, April 8-12 2024, Avila, Spain

ACM ISBN 979-8-4007-0243-3/24/04…$15.00
https://doi.org/xx.xxx/xxx_x

[1] Repository containing the code implementation: https://anonymous.4open.science/r/FAIR-2EFD

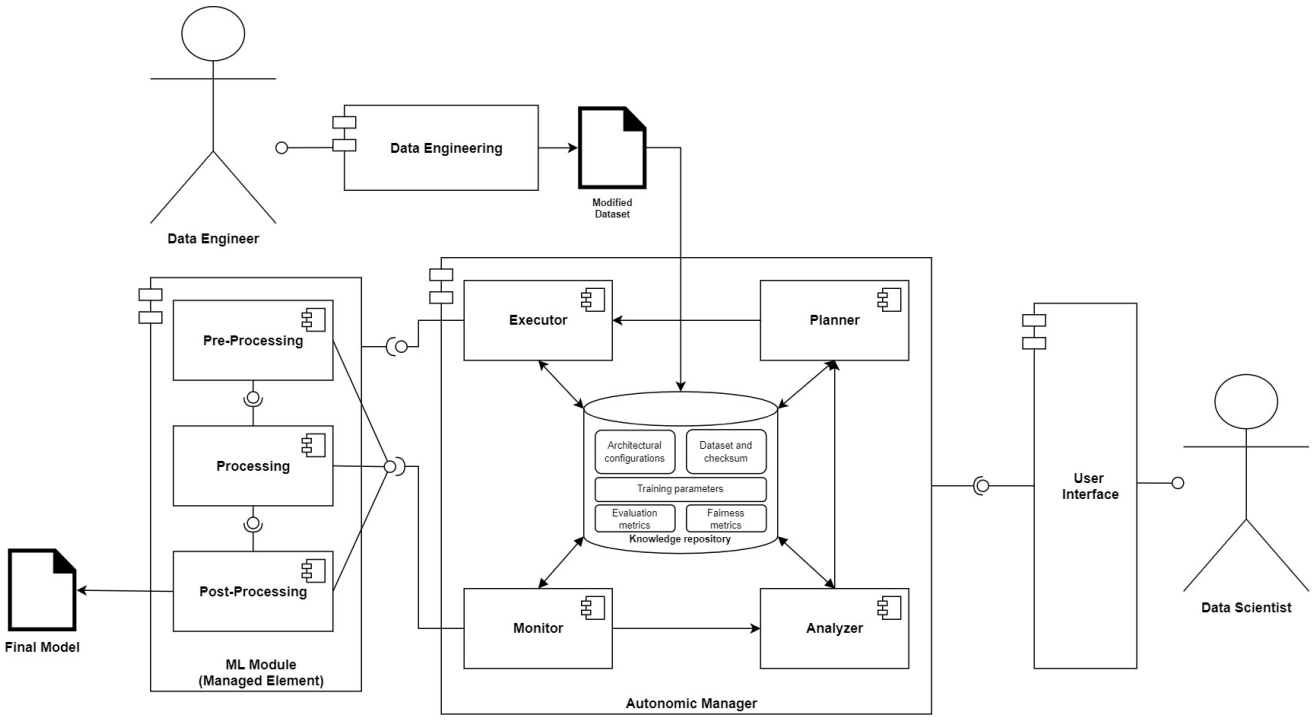

Figure 1: Architecture components of the proposed framework

prepare the dataset for execution in the ML Module. The User Interface module is used by the Data Scientist to specify the architectural configuration, which depends on the context of the problem. The ML Module can be executed manually or with the help of the Autonomic Manager.

The process of manual execution can be visualized in the Sequence Diagram in Figure 3. The Data Engineer first prepares the dataset using the Data Engineering module and stores them in the knowledge repository. The Data Scientist then starts the ML Module, passing in the necessary parameters. During execution, the ML Module starts the Monitor component from Autonomic Manager to collect data during execution, retrieves the dataset from the knowledge repository and executes the preprocessing, processing, and post-processing steps defined by the passed parameters to generate the final model. At the end of the execution, the Monitor saves all collected data in the knowledge repository for future use.

The process of execution with the help of the Autonomic Manager can be visualized in the Sequence Diagram in Figure 2. The Data Scientist, through the User Interface module, selects the dataset and requests an execution. The Autonomic Manager then searches for all previous executions of the ML Module that have already used the modified dataset and are stored in the knowledge repository. It then performs an analysis by performing calculations presented in section 2.4. Next, the Autonomic Manager then loops through the configuration of existing execution plans and executes them. It then returns the best execution options to the Data Scientist, who can choose the most appropriate option and request a new execution for the ML Module. This execution is equivalent to the one already explained in Figure 3. The Autonomic Manager then searches for the data from this execution to display it to the Data Scientist in the Interface.

2.2 The ML Module

The ML module is parameterized to allow the Data Scientist to control the architectural configuration used in its execution. This includes the dataset, protected attribute, and algorithms used. It is implemented using the *Pipes-and-Filters* architecture, which allows for interchangeable Pipes and Filters depending on the parameterization used, making it easy to program execution plans within the Autonomic Manager. These Pipes and Filters are divided into three types of components, illustrated in Figure 1, categorized by the newly developed algorithms that were created to improve the learning process's ability to handle biased data:

- **Pre-processing:** These algorithms try to remove biased data by transforming it. The implemented algorithms in these components are *Disparate Impact Remover* [11], *Learning Fair Representations* [19], *Reweighing* [13] and *Optimized Preprocessing* [9].
- **Processing:** These algorithms try to mitigate discrimination during the training process. The implemented algorithms in these components are *Adversarial Debiasing* [20], *Exponentiated Gradient Reduction* [4], *Grid Search Reduction* [5], *Meta Fair Classifier* [10], *Prejudice Remover* [15] and *Rich Subgroup Fairness* [16].
- **Post-processing:** These algorithms modify the predictions of an already trained model to produce fairer outcomes. The implemented algorithms in these components are *Equalized*

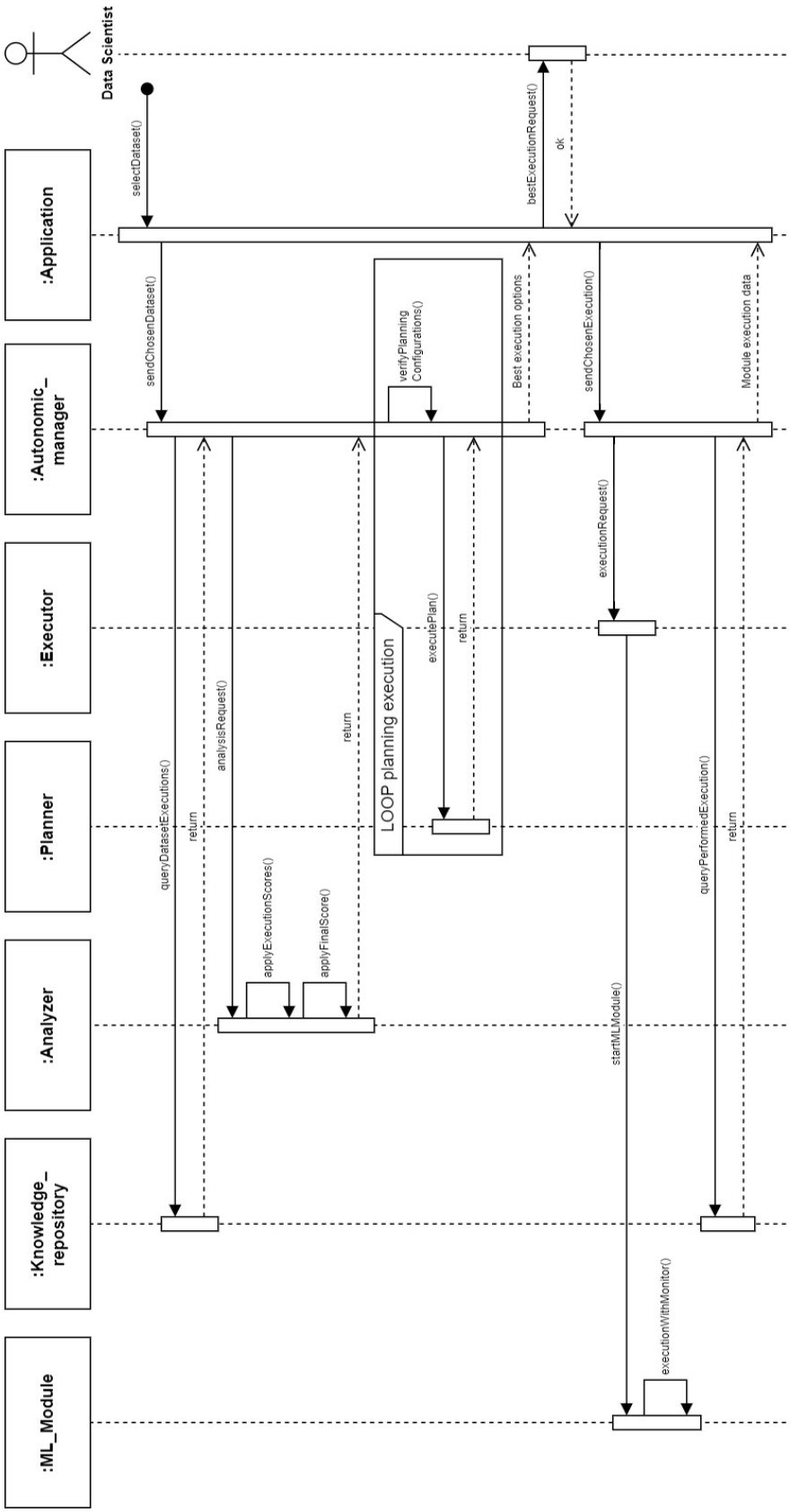

Figure 2: Sequence diagram with help of the Autonomic Manager

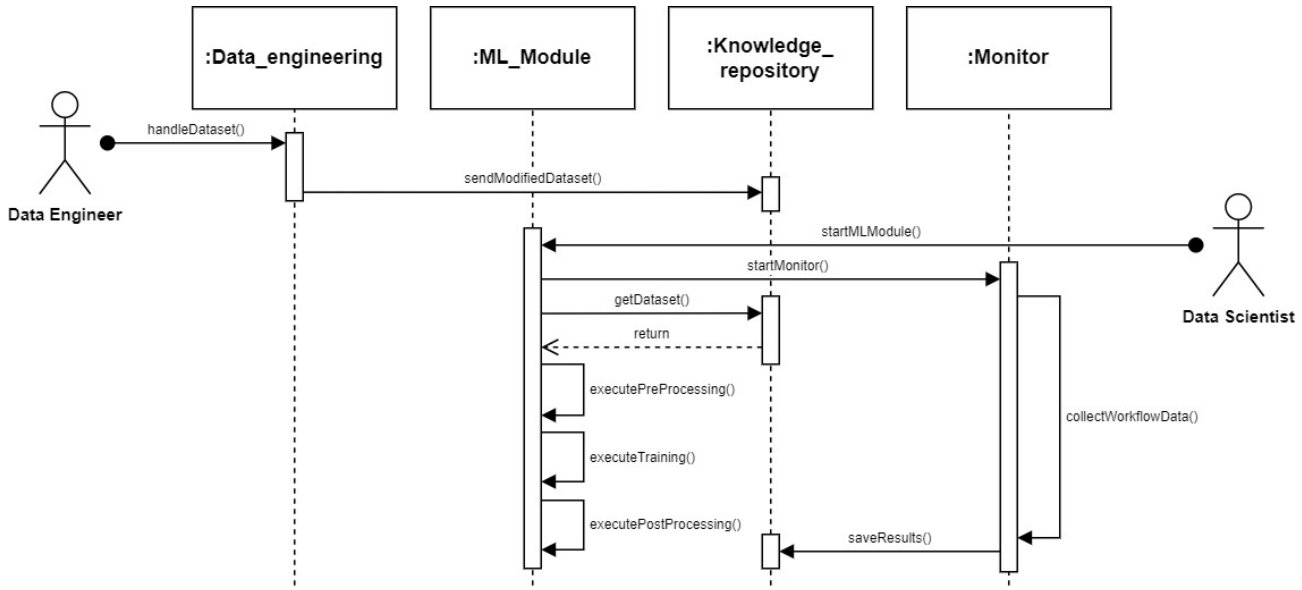

Figure 3: Sequence diagram from a manual ML Module execution

Odds [12], *Calibrated Equalized Odds* [17] and *Reject Option Classification* [14].

Some steps were additionally implemented in the Pre-processing components: The acquisition of the dataset; acquisition of the protected attribute, privileged group, and unprivileged group; division of the dataset into train, validation and test sets; the data preparation based in the protected attribute, privileged group, and unprivileged group. Because Pre-processing and Post-processing algorithms are not part of model training, Logistic Regression, Gradient Boosting, Random Forest and Support Vector Machines were additionally implemented in the Processing components. The calculation of Evaluation and Fairness metrics was additionally implemented in the Post-processing components.

2.3 Monitor Component

The Monitor is initiated along with the execution of the ML Module, as illustrated in Figure 3. The following metadata are stored:

- **Pre-processing**: Parameters related to the architectural configuration executed by the module (dataset, protected attribute, and pre-processing algorithm) and the checksum of the dataset used.
- **Processing**: Parameters related to the architectural configuration executed (training algorithm) and parameters used for training.
- **Post-processing**: Parameters related to the architectural configuration executed (post-processing algorithm) and metrics of the resulting model (evaluation metrics and fairness metrics).

The data collected per execution are organized into two different sets: One containing the metadata and information related to the architectural configuration executed by the ML Module, and one containing the metrics obtained by the resulting model from that execution. Both sets have an execution identifier to establish consistency between the data in the two sets. This organization facilitates the development of the Analyzer component.

2.4 Analyzer Component

The Analyzer extracts and analyzes ML Module executions from the knowledge repository based on their metrics, assigning each execution a score that consolidates its evaluation and fairness metrics. The analyses are then grouped by architectural configuration, and each configuration receives a final score. The scores for each architectural configuration are grouped together and placed in a dataset, with additional metadata such as execution date. This dataset is then passed to the Planner to help develop better strategies.

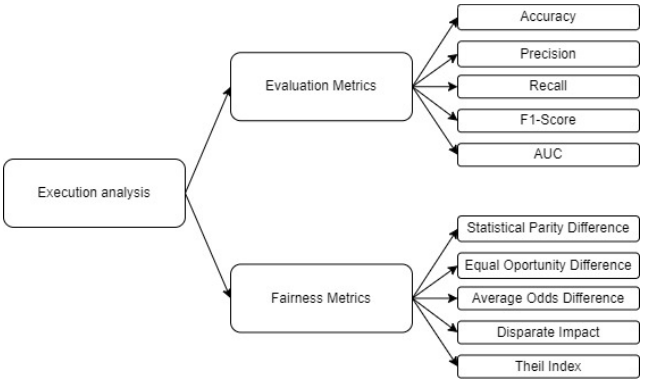

Figure 4: Metrics tree used in execution analysis

To analyze each execution made by the ML Module, the metrics are weighted according to their importance in the context of the problem, as determined by the Data Scientist. This is done

to consolidate the evaluation and fairness metrics into a single score, which simplifies planning strategies. This calculation uses a metrics tree, illustrated in Figure 4, to divide the metrics into two groups: Evaluation Metrics and Fairness Metrics. The Evaluation Metrics group includes the metrics Accuracy, Precision, Recall, F1-Score, and Area Under the ROC Curve (AUC). The Fairness Metrics group includes the metrics Statistical Parity Difference, or discrimination [19], Equal Opportunity Difference [7], Average Odds Difference [7], Disparate Impact [7] and Theil Index [18]. The weights w_E for the Evaluation Metrics group and w_F for the Fairness Metrics group are assigned and the way used to consolidate all metrics was through a score S defined by a weighted mean, displayed in equation 1, between the S_E and S_F scores.

$$S = \frac{w_F \times S_F + w_E \times S_E}{w_F + w_E} \tag{1}$$

To calculate the S_E and S_F for the Evaluation Metrics and Fairness Metrics groups, respectively, a weighted mean is calculated for each group. The weights for each metric are denoted by $w_{m_{Ei}}$ for the Evaluation Metrics group and $w_{m_{Fi}}$ for the Fairness Metrics group, but there are some differences to consider given the nature of the metrics in each group In the case of S_E, all calculated metrics are in the interval $[0, 1]$ and their results are directly proportional, meaning that the best results have the highest values. This implies that a weighted mean can be used without any modifications.

$$S_E = \frac{\sum_{i=1}^{n_{m_E}} w_{m_{E_i}} \times m_{E_i}}{\sum_{i=1}^{n_{m_E}} w_{m_{E_i}}} \tag{2}$$

In the equation 2, n_{m_E} is the number of evaluation metrics used (for the tree defined in Figure 4, $n_{m_E} = 5$), $w_{m_{E_i}}$ is the weight given to a given evaluation metric and m_{E_i} is the value of a given evaluation metric in the analyzed execution.

However, for the case of S_F, there are three different intervals and situations for the Fairness metrics present in the group:

- **Statistical Parity Difference**, **Equal Opportunity Difference** and **Average Odds Difference** involve the difference between two or more indicators in the interval $[0, 1]$, being in the interval $[-1, 1]$ and 0 is considered the best result.
- **Theil Index** is in the interval $[0, 1]$, but it is inversely proportional: 0 is considered the best result and 1 the worst.
- **Disparate Impact** is the only that involves the ratio of two indicators, being in the interval $[0, +\infty[$ and 1 is considered the best result.

To normalize the fairness metrics m_{F_i} to a metric m'_{F_i} in the interval $[0, 1]$, each case was treated separately. For the first two cases, $m'_{F_i} = 1 - |m_{F_i}|$ is used because produces the same results. For the third case, it needed another calculation. The following equation consolidates all cases:

$$m'_{F_i} = \begin{cases} 1 - |\frac{1}{m_{F_i}} - 1| & \text{if } m_{F_i} \text{ is Disparate Impact e } m_{F_i} > 1 \\ 1 - |m_{F_i} - 1| & \text{if } m_{F_i} \text{ is Disparate Impact and } 0 \leqslant m_{F_i} \leqslant 1 \\ 1 - |m_{F_i}| & \text{otherwise} \end{cases} \tag{3}$$

After obtaining m'_{F_i}, it is possible to obtain S_F using another weighted mean:

$$S_F = \frac{\sum_{i=1}^{n_{m_F}} w_{m_F} \times m'_{F_i}}{\sum_{i=1}^{n_{m_F}} w_{m_{F_i}}} \tag{4}$$

In the equation 4, n_{m_F} is the number of fairness metrics used (for the tree defined in Figure 4, $n_{m_F} = 5$), $w_{m_{F_i}}$ is the weight given to a given fairness metric, and e m'_{F_i} is the value of a given fairness metric in the analyzed execution that was normalized to the interval $[0, 1]$.

To consolidate the final scores S'_E, S'_F e S' for each architectural configuration, each execution is grouped according to the architectural configuration used, and a simple arithmetic mean is calculated according to the n' executions found for that configuration. To facilitate the visualization of the scores in the Interface module by the Data Scientist, the scores S'_E, S'_F e S' are multiplied by an arbitrary factor $X = 1000$ and then rounded, as shown in equation 5.

$$S'_F = \left\lfloor X \times \frac{\sum_{i=1}^{n'} S_{F_i}}{n'} \right\rfloor \quad S'_E = \left\lfloor X \times \frac{\sum_{i=1}^{n'} S_{E_i}}{n'} \right\rfloor \quad S' = \left\lfloor X \times \frac{\sum_{i=1}^{n'} S_i}{n'} \right\rfloor \tag{5}$$

2.5 Planner Component

To determine the architectural configuration that will be executed, some strategies were implemented in the Planner component:

- **Algorithm filtering**: Algorithms that may not perform well on ML Module or whose models have metrics that require further analysis by the Data Scientist are filtered out before selecting the ideal models.
- **Result range**: A minimum and maximum score range is used to determine scores that can be considered reliable for evaluation, thus reducing distortions caused by executions with unreliable metric results.

After evaluating these strategies, the Planner component will return the best 5 scores for the Data Scientist to choose, and will ask the Executor to run the ML Module again after choosing.

3 FRAMEWORK EVALUATION

Three case studies were conducted to assess the feasibility of the framework, future maintenance, and the ability of the Autonomic Manager to select the best options in different contexts. In all cases, the goal was to predict a credit rating (suitable or not suitable) using a fixed set of *features* in a chosen dataset. In the case of system evolutions, the lines of code performed in each change were also counted, to define whether such changes are simple to make.

The best architectural configurations were evaluated with 3 different weightings in the overall score:

- 50% for Evaluation metrics and 50% for Fairness metrics, for a balanced setup.
- 75% for Evaluation metrics and 25% for Fairness metrics, for a configuration that prioritizes quality over fairness.
- 25% for Evaluation metrics and 75% for Fairness metrics, for a configuration that prioritizes fairness over quality.

All executions were performed using a knowledge repository containing approximately 650 previous executions with various architectural configurations of datasets, protected attributes, and algorithms. The repository includes all the evaluation and fairness

metrics described on section 2.4, and all of which are weighted equally within their respective groups.

Since no studies were found where bias reduction is used in more than one step of the machine learning process, only one bias reduction algorithm is performed per execution. Due to unreliable data, executions with the algorithms *Optimized Preprocessing* and *Reject Option Classification* were filtered, corresponding to approximately 5% of total base size.

3.1 Case Study 1: Autonomic Manager feasibility and usefulness

This case study evaluated the behavior of the Autonomic Manager using the German Credit Dataset [2]. Age and nationality were used as protected attributes, and a result range of 500 to 950. The results are presented in Tables 1, 2, and 3.

Two surprising observations were made in these executions. First, post-processing algorithms were predominantly used, especially in settings that prioritized quality. This is contrary to the expectation that fairness-enhancing algorithms would improve fairness at the expense of quality. Second, pre-processing algorithms combined with Support Vector Machines as the training algorithm were predominant in configurations that prioritized fairness.

In all executions, it was found that it is difficult to identify the most balanced choice between the two groups of metrics due to the large number of metrics and algorithms used. Additionally, the difference between the metrics is very small, making the choice even more difficult. Consolidating the metrics into groups simplifies the visualization of which architectural configurations are more balanced. The use of weights for each metric and each group could be calibrated to achieve the best desired balance for a given situation.

In a development context, the Autonomic Manager simplifies the Data Scientist's decision-making and significantly reduces the time to obtain and deploy an optimized model. This is because it eliminates the need to execute multiple algorithms due to the availability of a prior knowledge repository. Additionally, there could be processing and cost savings in the event of future problems with the same dataset, as the executions saved by the teams that use this process can be used by other teams to verify better choices.

Table 1: Best configurations chosen by Autonomic Manager 50% Evaluation/50% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Age	Nothing	Logistic Regression	Equalized Odds	968	860	**914**
Nationality	Nothing	Random Forest	Calibrated Equalized Odds	902	922	**912**
Nationality	Nothing	Gradient Boosting	Calibrated Equalized Odds	870	925	**898**
Age	Nothing	Gradient Boosting	Equalized Odds	927	862	**894**
Age	Reweighing	Gradient Boosting	Nothing	804	931	**868**

Table 2: Best configurations chosen by Autonomic Manager 75% Evaluation/25% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Age	Nothing	Logistic Regression	Equalized Odds	968	860	**941**
Age	Nothing	Gradient Boosting	Equalized Odds	927	862	**910**
Nationality	Nothing	Random Forest	Calibrated Equalized Odds	902	922	**907**
Nationality	Nothing	Gradient Boosting	Calibrated Equalized Odds	870	925	**883**
Age	Nothing	Random Forest	Equalized Odds	898	799	**874**

Table 3: Best configurations chosen by Autonomic Manager 25% Evaluation/75% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Age	Disparate Impact Remover	Support Vector Machines	Nothing	747	989	**928**
Nationality	Disparate Impact Remover	Support Vector Machines	Nothing	747	989	**928**
Age	Nothing	Adversarial Debiasing	Nothing	742	979	**920**
Nationality	Reweighing	Support Vector Machines	Nothing	755	972	**918**
Nationality	Learning Fair Representations	Support Vector Machines	Nothing	755	972	**918**

3.2 Case Study 2: System evolution by adding a new dataset

This case study evaluated the Autonomic Manager's versatility in different contexts and the simplicity of system maintenance. The Lendingclub Dataset [1] was added in ML Module and used as the dataset, with Income as the protected attribute and a result range of 500 to 980. The results are presented in Tables 4, 5, and 6.

Table 4: Best configurations chosen by Autonomic Manager 50% Evaluation/50% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Income	Learning Fair Representations	Random Forest	Nothing	991	968	**979**
Income	Nothing	Gradient Boosting	Equalized Odds	988	969	**978**
Income	Reweighing	Random Forest	Nothing	991	963	**977**
Income	Learning Fair Representations	Logistic Regression	Nothing	981	973	**977**
Income	Reweighing	Gradient Boosting	Nothing	987	964	**976**

Table 5: Best configurations chosen by Autonomic Manager 75% Evaluation/25% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Income	Nothing	Logistic Regression	Equalized Odds	985	965	**980**
Income	Learning Fair Representations	Gradient Boosting	Nothing	987	960	**980**
Income	Learning Fair Representations	Logistic Regression	Nothing	981	973	**979**
Income	Nothing	Grid Search Reduction	Nothing	989	950	**979**
Income	Reweighing	Logistic Regression	Nothing	981	965	**977**

Table 6: Best configurations chosen by Autonomic Manager 25% Evaluation/75% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Income	Learning Fair Representations	Logistic Regression	Nothing	981	973	**975**
Income	Nothing	Gradient Boosting	Equalized Odds	988	969	**974**
Income	Learning Fair Representations	Random Forest	Nothing	991	968	**973**
Income	Nothing	Exponentiated Gradient Reduction	Nothing	986	966	**971**
Income	Reweighing	Gradient Boosting	Nothing	987	964	**970**

In this case study, the best architectural configurations were completely different for this problem context and data, with a predominance of *Learning Fair Representations* for bias reduction and Logistic Regression for training. Algorithms with post-processing bias reduction and training algorithms such as *Support Vector Machines* were not as efficient as in the previous case study. This demonstrates the Autonomic Manager's ability to help make decisions in a more efficient and agile way in different problem contexts and data. However, the data and metadata obtained did not provide enough information to understand why these changes occurred.

To process the Lendingclub Dataset, modifications were necessary to do the system evolution and add this dataset as an option in

ML Module. These were counted according to their *commits* made in the repository and displayed in the Table 7.

Table 7: Number of modifications performed when adding a new dataset to ML Module

Module	Changed lines	Total lines	Changed files	Total files	% changed lines	% changed files
Data Engineering	122	277	2	3	**44,04%**	**66,67%**
ML Module	76	1982	5	38	**3,84%**	**13,16%**
Autonomic Manager	4	889	1	17	**0,45%**	**5,88%**
User Interface	13	2905	2	14	**0,45%**	**14,29%**
TOTAL	**215**	**6053**	**10**	**72**	**3,55%**	**13,89%**

The first conclusion that can be discussed is related to the modification in the Autonomic Manager, which was based on additional parameters for the integration between with the User Interface, without affecting the MAPE-K architecture-based components. This fact and the large difference between the results of Case Study 1 and 2 reinforce the autonomy proposed in MAPE-K control *loop*, enabling different architectural configurations based on the metadata captured in the ML Module.

The User Interface required minimal modifications to accommodate the new dataset option for Data Scientists. The ML Module and Data Engineering module underwent the most significant changes. In the ML Module, most modifications (34 lines, or 44.74%) were made to data preprocessing to ensure the correct application of training algorithms. Although structuring the ML Module using the *Pipes-and-Filters* architecture requires additional classes (2 for pipes and 1 for filters), Data Engineers and Data Scientists will spend most of their time and make more modifications during data processing and transformation steps.

Although using the *Pipes-and-Filters* architecture requires writing a few more lines of code, it allows for the encapsulation of algorithms and separation of concerns in a simple way, which leads to good code design. This makes system maintenance and evolution relatively simple, as long as the developer knows which files to modify. Therefore, creating documentation is extremely important for new developers to understand the system as a whole and avoid adding unnecessary code.

3.3 Case Study 3: System evolution with developer with no prior knowledge adding a new training algorithm

This case study describes another system evolution, which involved adding a new classification algorithm. System maintenance is again discussed, this time focusing on the changes made by other developers. To determine whether the chosen architectures are versatile and simple enough for new developers to understand the system context and easily make changes, the Lendingclub Dataset [1] was used, with Income as the protected attribute and a result range of 500 to 980.

In a 1-2 hour development session with another developer, a new classification algorithm was added to the ML Module structure using the documentation created in the previous case study. During the session, some documentation errors and bugs were identified and corrected through the developer's observation and feedback. To avoid wasting time and allow the developer to focus on evolving the system, some troubleshooting was performed on a small portion of the session. After the session, the developer completed a questionnaire indicating positive impressions, a relatively straightforward implementation, and a profile with some experience in data and software engineering careers. Although the initial objective was for the developer to be guided only by the documentation, the developer themselves found it easy to adapt to the experiment, even without additional assistance. This suggests that the initial design decisions were correct and facilitated the development of evolutions.

The implemented classification algorithm was Naive Bayes, as it is a widely used classifier. After development, the results are present below in Tables 8, 9 and 10.

Table 8: Best configurations chosen by Autonomic Manager 50% Evaluation/50% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Income	Learning Fair Representations	Random Forest	Nothing	991	968	**979**
Income	Nothing	Gradient Boosting	Equalized Odds	988	969	**978**
Income	Reweighing	Random Forest	Nothing	991	963	**977**
Income	Learning Fair Representations	Logistic Regression	Nothing	981	973	**977**
Income	Reweighing	Gradient Boosting	Nothing	987	964	**976**

Table 9: Best configurations chosen by Autonomic Manager 75% Evaluation/25% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Income	Nothing	Logistic Regression	Equalized Odds	985	965	**980**
Income	Learning Fair Representations	Gradient Boosting	Nothing	987	960	**980**
Income	Learning Fair Representations	Logistic Regression	Nothing	981	973	**979**
Income	Nothing	Grid Search Reduction	Nothing	989	950	**979**
Income	Reweighing	Logistic Regression	Nothing	981	965	**977**

Table 10: Best configurations chosen by Autonomic Manager 25% Evaluation/75% Fairness

Architectural Configuration				Score		
Protected attribute	Pre-processing	Processing	Post-processing	Evaluation	Fairness	**Overall**
Income	Nothing	Naive Bayes	Calibrated Equalized Odds	991	976	**980**
Income	Learning Fair Representations	Logistic Regression	Nothing	981	973	**975**
Income	Nothing	Gradient Boosting	Equalized Odds	988	969	**974**
Income	Learning Fair Representations	Random Forest	Nothing	991	968	**973**
Income	Nothing	Exponentiated Gradient Reduction	Nothing	986	966	**971**

In the LendingClub dataset context, adding Naive Bayes to the ML Module proved beneficial, presenting a higher score than expected for configurations that prioritize fairness. However, it was not highlighted because of the established maximum threshold value for the Analyzer. As seen in previous case studies, this may not work in other contexts, and the established data and metadata do not explain why Naive Bayes had a positive behavior for this dataset.

The evolutions made in the ML Module to add Naive Bayes were counted according to their *commits* made in the repository and displayed in the Table 7.

Table 11: Number of modifications performed when adding a new algorithm to ML Module

Module	Changed lines	Total lines	Changed files	Total files	% changed lines	% changed files
Data Engineering	0	277	0	3	**0,00%**	**0,00%**
ML Module	60	2042	5	39	**2,94%**	**12,82%**
Autonomic Manager	2	891	2	17	**0,23%**	**11,77%**
User Interface	71	2948	3	14	**2,41%**	**21,43%**
TOTAL	**132**	**6157**	**9**	**72**	**2,14%**	**12,50%**

In this case study, the Data Engineering module was the only module that did not require modifications, as it had already been implemented in a previous case study. Additionally, the only modification required to the Autonomic Manager was the addition of Naive Bayes in your parameterization, which could be transferred to an external configuration file, eliminating the need for future code modifications. Overall, the proposed evolution required fewer changes than the one in the previous case study. The only module that required more modifications was the User Interface, primarily due to a single component. Aside from this exception, adding an algorithm is a relatively straightforward evolution to make, and with the documentation created, a relatively experienced developer can perform this task without major difficulties.

4 CONCLUSIONS AND FUTURE WORK

The reviewed literature on Fairness checks only covers binary classification problems. For evolutions and new methods, refactorings in the ML Module are likely to occur. The *Pipes-and-Filters* architecture allows for the encapsulation of all the procedures present in a workflow of a Machine Learning application into cohesive steps, which can be changed if necessary to test with another algorithm, protected attribute, or dataset, and makes it easier for other developers to understand.

The MAPE-K architecture enables the creation of a flow to analyze the data obtained in the ML Module to facilitate decision-making. Although autonomy is viable, human supervision is necessary due to the challenges still faced in the fairness domain, where the social context of the problem is extremely important when evaluating whether a model is considered good. The use of weights for the metrics and different strategies in the analysis and planning phases of the Autonomic Manager ensures a balance of evaluation/fairness metrics and helps to define the context for an evaluation. However, it still depends on a Data Scientist with domain knowledge to understand the needs of the analyzed problem and whether the results are acceptable for publishing an optimized and fair model. The Data Scientist can use a variety of factors to determine the weights, such as the specific business goals, the regulatory environment, and the ethical considerations.

Overall, these architectures are well-suited to the implementation of the main objectives. For future work, the following possibilities can be explored: In the Autonomic Manager, the Analyzer can perform a deeper analysis by increasing the number of indicators, considering groups in addition to fairness metrics and evaluation metrics and adaptations to a Logic Scoring of Preference (LSP) based approach. In the ML Module, techniques such as Data Augmentation and K-Fold Cross-Validation can be introduced to improve results. It is also possible to change the focus to solving MLOps problems, using the system to determine a better deployment in case of worsening in the metrics of a model already used by customers.

REFERENCES

[1] [n. d.]. Lendingclub Dataset. https://www.kaggle.com/datasets/wordsforthewise/lending-club

[2] [n. d.]. Statlog (German Credit Data) Data Set. https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)

[3] 2005. *An Architectural Blueprint for Autonomic Computing.* Technical Report. IBM. https://www-03.ibm.com/autonomic/pdfs/AC%20Blueprint%20White%20Paper%20V7.pdf

[4] Alekh Agarwal, Alina Beygelzimer, Miroslav Dudík, John Langford, and Hanna Wallach. 2018. A reductions approach to fair classification. In *International Conference on Machine Learning*. 60–69.

[5] Alekh Agarwal, Miroslav Dudík, and Zhiwei Steven Wu. 2019. Fair regression: Quantitative definitions and reduction-based algorithms. In *International Conference on Machine Learning*. 120–129.

[6] Tom Begley, Tobias Schwedes, Christopher Frye, and Ilya Feige. 2021. Explainability for fair machine learning.

[7] Sumon Biswas and Hridesh Rajan. 2020. Do the Machine Learning Models on a Crowd Sourced Platform Exhibit Bias? An Empirical Study on Model Fairness. (2020), 642–653.

[8] Joy Buolamwini and Timnit Gebru. 2018. Gender Shades: Intersectional Accuracy Disparities in Commercial Gender Classification. In *Conference on Fairness, Accountability and Transparency (Proceedings of Machine Learning Research, Vol. 81)*. 77–91.

[9] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. 2017. Optimized Pre-Processing for Discrimination Prevention. In *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper/2017/file/9a49a25d845a483fae4be7e341368e36-Paper.pdf

[10] L Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K Vishnoi. 2019. Classification with fairness constraints: A meta-algorithm with provable guarantees. In *Proceedings of the conference on fairness, accountability, and transparency*. 319–328.

[11] Michael Feldman, Sorelle A. Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and Removing Disparate Impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 259–268. https://doi.org/10.1145/2783258.2783311

[12] Moritz Hardt, Eric Price, Eric Price, and Nati Srebro. 2016. Equality of Opportunity in Supervised Learning. In *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper/2016/file/9d2682367c3935defcb1f9e247a97c0d-Paper.pdf

[13] Faisal Kamiran and Toon Calders. 2011. Data Pre-Processing Techniques for Classification without Discrimination. *Knowledge and Information Systems* (2011). https://www.researchgate.net/publication/228975972_Data_Pre-Processing_Techniques_for_Classification_without_Discrimination

[14] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. 2012. Decision Theory for Discrimination-Aware Classification. In *2012 IEEE 12th International Conference on Data Mining*. 924–929.

[15] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. 2012. Fairness-Aware Classifier with Prejudice Remover Regularizer. 35–50. https://www.researchgate.net/publication/262176212_Fairness-Aware_Classifier_with_Prejudice_Remover_Regularizer

[16] Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. 2018. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *International Conference on Machine Learning*. 2564–2572.

[17] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. 2017. On Fairness and Calibration. In *Advances in Neural Information Processing Systems*. https://proceedings.neurips.cc/paper/2017/file/b8b9c74ac526fffbeb2d39ab038d1cd7-Paper.pdf

[18] Till Speicher, Hoda Heidari, Nina Grgic-Hlaca, Krishna P. Gummadi, Adish Singla, Adrian Weller, and Muhammad Bilal Zafar. 2018. A Unified Approach to Quantifying Algorithmic Unfairness. (2018). http://dx.doi.org/10.1145/3219819.3220046

[19] Rich Zemel, Yu Wu, Kevin Swersky, Toni Pitassi, and Cynthia Dwork. 2013. Learning Fair Representations. In *Proceedings of the 30th International Conference on Machine Learning*. 325–333. https://proceedings.mlr.press/v28/zemel13.html

[20] Brian Zhang, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating Unwanted Biases with Adversarial Learning. 335–340. https://www.researchgate.net/publication/330299272_Mitigating_Unwanted_Biases_with_Adversarial_Learning