Software and Systems Modeling (2022) 21:1135–1157
https://doi.org/10.1007/s10270-022-00983-5

THEME SECTION PAPER

# AI-powered model repair: an experience report—lessons learned, challenges, and opportunities

Angela Barriga[1] · Adrian Rutle[1] · Rogardt Heldal[1]

Received: 31 March 2021 / Revised: 22 December 2021 / Accepted: 26 January 2022 / Published online: 22 February 2022

**Abstract**
Artificial intelligence has already proven to be a powerful tool to automate and improve how we deal with software development processes. The application of artificial intelligence to model-driven engineering projects is becoming more and more popular; however, within the model repair field, the use of this technique remains mostly an open challenge. In this paper, we explore some existing approaches in the field of AI-powered model repair. From the existing approaches in this field, we identify a series of challenges which the community needs to overcome. In addition, we present a number of research opportunities by taking inspiration from other fields which have successfully used artificial intelligence, such as code repair. Moreover, we discuss the connection between the existing approaches and the opportunities with the identified challenges. Finally, we present the outcomes of our experience of applying artificial intelligence to model repair.

**Keywords** Artificial intelligence · Model repair · Challenges · Opportunities

## 1 Introduction

In model-driven engineering (MDE), the increasing complexity of software systems is handled by utilizing abstract software models. When producing these models, software developers may introduce various issues which corrupt the models or reduce model quality, such as syntactic errors, smells [1], antipatterns [2], and inter-model inconsistencies [3]. In addition, the chances of corrupting a model increase along with the size of development teams and the number of software requirements, lack of coordination, misunderstanding, and mishandled collaborative projects [4]. These issues may lead to severe challenges in the later phases of the development process since the reliability and accuracy of these models are important to correctly produce the software systems which they represent.

Communicated by L. Burgueño, J. Cabot, M. Wimmer and S. Zschaler.

✉ Angela Barriga
abar@hvl.no

Adrian Rutle
rohe@hvl.no

Rogardt Heldal
aru@hvl.no

1 Western Norway University of Applied Sciences, Bergen, Norway

Ensuring that a model does not contain issues is a time-consuming task. As a consequence, a variety of approaches to automatic model repair have been proposed over the last decades to tackle the repair of corrupted models from different perspectives and applied to different models [5–7]. Some of these approaches provide automatic repair; however, there might be multiple repair solutions that do not satisfy all modelers' preferences. Likewise, there are multiple types of issues that could appear in the same model. Cognifying [8] model repair could be the solution for giving automatic approaches enough power to deal with this diversity of problems.

Over the last years, artificial intelligence (AI) has risen as a disruptive trend that is bringing a new era for how we design and maintain technology. AI brings the potential to automate complex tasks and replicate human behavior in multiple domains, including software engineering. The ability of AI to recognize patterns and learn how to deal with them allows automating repetitive tasks such as bug fixing and code repair [9], testing [10], quality assurance [11,12], or verification [13]. Despite the potential of AI to provide automatic model repair, there is not much research done in this direction.

Recently, there has been an increasing interest in the modeling community to combine AI and MDE. Conferences such as MODELS [14] are focusing on AI-related special themes (e.g., 2021 edition: Modeling for Human-AI Collaborative Society, 2020 edition: Modeling in the era of data) and new workshops such as the Workshop on Artificial Intelligence and Model-driven Engineering (MDE Intelligence) [15] are appearing and becoming more popular. Likewise, several authors in the field have already identified the application of AI in MDE as a challenge to overcome in order to increase the adoption of MDE [8,9,16]. As stated in [16], to automate and make more powerful maintenance solutions, we could extend MDE techniques by exploiting AI techniques. AI is slowly but steadily being adopted in different MDE processes and AI-powered model repair is not an exception.

In this paper, we explore the existing research on AI-powered model repair. Then, we identify open research challenges that the community needs to overcome to improve the adoption of AI-powered model repair. Furthermore, we present a number of unexploited research opportunities based on the successful usage of AI in comparable fields. Finally, we present lessons learned while developing our tool for AI-powered model repair.

**Structure of the paper** This paper is organized as follows: Sect. 2 presents background related to model repair and AI. Section 3 introduces an exploration of approaches of the AI-powered model repair field. Then, Sects. 4 and 5 present, respectively, research challenges, and opportunities of the AI-powered model repair field. Section 6 presents lessons we learned while developing our own AI-powered model repair tool. In Sect. 7, we discuss the outcomes of the identified approaches, challenges, and opportunities sections. Section 8 presents the threats to validity of our research. Then, we conclude the paper in Sect. 9.

## 2 Background

AI-powered model repair is the intersection of two fields: AI and model repair. Hence, it is necessary to understand the basics of these two fields before diving into further detail about their combination. In this section, we briefly present what model repair and AI are, in order to provide a better understanding of the topics later presented in this paper.

### 2.1 Model repair

In this paper, we refer to anything that might be considered as wrong or improvable within a model as an *issue*. Issues might be of different types depending on their nature and the context where they appear. For example, an issue could be a syntactic error triggered when an instance model does not comply with its corresponding metamodel (e.g., Ecore class diagrams and the Ecore metamodel) or with constraints defined by the user (e.g., OCL constraints), but it could also be an issue among models representing different aspects of the same software system (e.g., class and sequence diagrams with mismatching messages and operations [17]).

Other types of issues out of the scope of this paper would include over- and under-constraining issues (e.g., too many constraints imposed over a model and too few variables to solve the issues caused by violating the constraints, and vice versa, respectively). We also could consider smells as issues indicating which parts of the models are more prone to get corrupted (e.g., duplicated features in a hierarchy that might lead to increasing the model's complexity).

Various methods could be used to detect issues in models, e.g., model-checking [18,19] which determine whether a model satisfies a set of logical formulae. If it does, the model is considered correct with respect to those formulae. However, if not, the model contains issues that need to be managed to restore correctness with respect to those formulae [20,21]. Managing these issues is the goal of model repair. Citing Macedo et al. [22]: *"One of the main challenges in model repair is that, for any given set of inconsistencies* (issues in our terms), *there (possibly) exists an overwhelming number of updates that resolve them."* Hence, finding the appropriate repair actions is not a trivial process.

The existing model repair approaches tackle this problem from different perspectives. For example, some approaches try to find the repaired version of the model closer to the original one in terms of structure [4], to get the repair actions from the ones previously applied in the model [23], or by directly asking the user which repair actions they prefer, either by interacting with the user during the repair process [6,24] or by asking for her preferences before the repair starts [25]. The goal of AI-powered model repair is to take advantage of the potential of AI to find the most optimal repair actions to manage the issues in a model.

### 2.2 Artificial intelligence

AI can be defined as the attempt to build automated imitation of intelligent behavior, embedding intelligence into machines [26]. Most AI advances in the last years, especially within software engineering, are related to an AI branch called machine learning (ML). ML is a subset of AI that allows machines to learn from data without being programmed explicitly, achieving automated detection of meaningful patterns in data. ML focuses on the use of the strengths and special abilities of computers to complement human intelligence, often performing tasks that fall way beyond human capabilities (e.g., processing a great amount of information in a tiny time-span).

ML is a field with multiple algorithms, which can be classified depending on how they learn from data. There exist multiple classifications of ML approaches but, to avoid unnecessary complexity, in this section we follow a simplified ML classification that aims to introduce concepts which are relevant to the existing approaches in the AI-powered model repair field. Our classification distinguishes between three groups of algorithms: supervised learning, unsupervised learning, and reinforcement learning (RL) [26]. In the following, we will briefly explain these three groups. Additionally, we will introduce some ML architectures[1] such as neural networks (NNs) and deep learning (DL), given their current importance in the ML field. We will also explore logic-based approaches, as they will be relevant later in the paper.

*Supervised learning* Supervised learning algorithms learn a function that maps an input to an output based on example input–output pairs. They infer a function from labeled training data consisting of a set of training examples [27]. This sort of algorithms is suitable for tasks like classification and regression.

In classification, the algorithm takes an input and returns either a specific label or a number specifying the confidence score for a particular label [28]. An example of a classification task would be distinguishing pictures of cats and dogs. Even though all cats and dogs are unique, we are still usually able to tell them apart. For this to be supervised learning, we need some training data describing color, shape, distinguishing features, etc., and a label specifying the correct prediction (e.g., whether the picture contains a cat or a dog). A successful algorithm should be able to recognize previously unseen cats and dogs after training on this data.

In regression (also called prediction), the algorithm takes an input and returns a number as output [29]. An example of a regression task would be to predict the price of housing in ten years. The algorithm could train on data containing attributes such as the size of the houses, initial value, evolution during the years, and quality of the materials. Then, the algorithm would fit a function to this data to predict the output price.

*Unsupervised learning* Unsupervised learning is a type of algorithms that learns patterns from unlabeled data. The algorithms look to find some recurrent patterns in the input data [26]. Then, as output, these algorithms group the introduced data, displaying which data points in the input data are more related to each other. The most common application of these algorithms is clustering.

Clustering is a task for finding clusters/groups within the input data [29]. An example application is image compression, where the objective is to reduce the file size of the image. The input is the pixels that make up the image, and each pixel is represented by an RGB-value. The algorithm will find all the pixels related to a specific color (e.g., red), calculate the average color, and set it for all the pixels of that color. This will slightly reduce the details of the image, although in an undetectable way for the human eye, and save storage space.

*Reinforcement learning* RL consists of algorithms able to learn by themselves how to interact in an environment without existing pre-labeled data, only needing a set of available actions and rewards for each of these actions [30]. By using rewards, these algorithms can learn which are the best actions to interact with an environment. The learning process of RL comes from the interaction between an agent (the intelligence in the algorithm) and an environment (the problem to solve). The agent performs actions in the environment that change its state. For example, if the environment is a maze, the agent is a robot, the action is walking one step to the right and the state the current position of the robot in the maze, then, the new state would be the robot's new location: one position to the right. If the action is positive for the agent (moving toward the maze exit), it receives a reward, contrarily (stepping on a wall) it will be penalized. The agent will continue performing actions trying to get the highest reward until it reaches its ultimate goal, e.g., reaching the exit of the maze. Algorithms able to beat humans in chess or Go are RL algorithms.

Now that we have introduced the three main categories of learning algorithms in ML, we continue with some well-known ML architectures and approaches that can solve tasks related to supervised, unsupervised, and reinforcement learning.

*Neural networks and deep learning* Unlike the previous paragraphs, NNs are not a set of algorithms distinguished by the way they learn, but by their structure. NNs are an ML architecture, and there exist multiples different types of NNs able to solve problems within supervised, unsupervised, and reinforcement learning.

NNs are inspired by the brain's structure and simulate a net of neurons able to identify patterns and correlations in data. NNs contain multiple layers of a data structure called neurons, which are connected with each other. These connections have changing weights that simulate the connection of neurons in the human brain. Connections with stronger weights will be favored and will lead to learning the solution for a given problem. NNs allow to perform multiple tasks given a dataset with enough examples from which the network can learn.

DL is a branch of NNs, which has become the most popular trend in ML in the last years. DL follows a similar structure to traditional NNs, but they usually have more layers of neurons, and neurons are inter-connected in a more complex way. DL NNs are computationally demanding, which is why their use has not been feasible until recently.

It is usual to combine NNs and DL with other algorithms. For example, a NN could be used together with an RL algo-

[1] We use the term ML architecture instead of ML models to avoid confusion with software models.

rithm in problems where the consequences of applying an action in an environment are not known beforehand, or a DL NN could solve complex RL tasks where the reward function is unknown, as in [31], where authors make use of a DL NN to learn goals defined by high-level user preferences.

*Logic-based learning* Logic-based learning is a branch of AI suited for search problems, that bridges ML and knowledge representation—providing information about the world in a form that computer systems understand. It involves creating logic-based programs automatically from examples and domain knowledge. Given a set of examples of what is known to be true or false in a system, logic-based approaches create a hypothesis space (search space) for computing possible solutions to the problem faced.

Logic-based approaches can learn from small labeled structured data using declarative prior knowledge expressed in some logic formalism. These approaches support transfer and continuous learning. Additionally, the solutions produced are interpretable and guaranteed to meet the semantic properties of the system they work with.

Similar to NNs, logic-based approaches are able to solve problems within supervised, unsupervised, and reinforcement learning. There exist multiple languages and specific logic-based approaches. In this paper, we will focus on inductive logic programming (ILP), as it is applied to model repair in several works in the literature. ILP makes use of logic programming to represent examples, background knowledge, and hypotheses. Given background knowledge and examples encoded in logic programming, the ILP system can explore hypotheses to derive as a solution a logic program that represents all the positive and none of the negative examples.

## 3 Exploration of the AI-powered model repair field

A lot of work has been done in the area of automatic model repair [22] but, although AI can be considered an automated approach, not all automated approaches are AI. Hence, before detailing each approach, it is important to state the difference between automated and AI approaches. An approach falls within the AI scope when it was not tailor-programmed for a specific task. An AI approach can learn how to solve different tasks without needing further code change. With this distinction, for example, some rule-based [6,32], graph-transformation [33,34], and brute force [35] approaches are discarded. Some rule-based approaches could be considered as rule-based ML when they are able to identify, generate, or modify rules on their own; however, approaches in the literature [6,36] usually have as a pre-requisite some kind of definition of the rules.

In the following, we present an exploration of the existing work and approaches of AI-powered model repair. We include different directions where researchers have worked on the intersection between AI and model repair. Therefore, this section can be used as a guide for future research in this area. The purpose is to present the current status on the field and to promote the start of new and innovative research. Additionally, we include also model refactoring techniques since model repair can be seen as an activity that aims at resolving issues in models by refactoring them. Wherever there are similar approaches, we group them by the kind of learning algorithms they use. Table 1 summarizes the approaches identified throughout this section.

To provide a better understanding of each group of approaches presented, we will explain how they could repair the model presented in Fig. 1. For each example, we will use one of the concrete works presented in each group, since these works are different implementations of the same approach. The sample model in Fig. 1 represents a part of a smart system in which a device has several statuses, categories, owners, and an address. The model contains several issues: classes with duplicated attributes and references (issues 1 and 2), an operation with two return parameters instead of a single one (issue 3), a containment reference with an upper bound greater than 1 (issue 4), an attribute which is actually an association (issue 5), and an attribute without a type (issue 6). Each of these issues can be addressed by applying different actions. To repair the model in Fig. 1, one has to handle all the issues in the model. For example, issue1 could be handled by deleting or updating one of the duplicated classes Status or Public_Status or by creating a hierarchy within these classes. This hierarchy could consist of promoting one of the initial classes to superclass or making both of them children of a new superclass. References could remain in the children or belong to the superclass.

### 3.1 Tree learning

The most extended approach is to use decision trees to support the repair. Decision trees are nonparametric supervised learning methods used for both classification and regression tasks. This approach goes from observations about an item to conclusions about the item's target value.

Using tree learning, Kretschmer et al. introduce in [36] an approach for discovering and validating values for repairing issues automatically; they group alike these repair values if they have the same effect, which impacts positively the scalability of the approach. Prior to the validation, an input model with a set of consistency rules (such as OCL) is required. For each issue, a validation tree is constructed. The tree identifies all model elements involved (leaves) and shows how their values cause the issues. Then, using different techniques, the values on the leaves are changed, which may or

**Table 1** Summary of the identified approaches in the AI-powered model repair field

| Technique | Approach | Description | Technologies |
|---|---|---|---|
| Tree learning | Kretschmer et al. [63] | Automatic discovery and validation of repairing values with Boolean logic. | OCL |
| | Khelladi et al. [62] | Repairs models and ranks repairs depending on the positive or negative side-effect they produce. | UML, OCL |
| | Model/ Analyzer [92] | Given constraints, determines which specific parts of a model must be checked and repaired. | Independent of modeling languages, OCL |
| Automated planning | Badger [90,91] | Chooses the minimum repair actions to reach a user-defined goal. Users can prioritize different repairs. | UML |
| Markov decision process | PARMOREL [17,19-21,57] | Extensible framework for model repair that allows personalized and automatic repair using reinforcement learning. | Independent of modeling languages |
| Neural networks | Burgueño et al. [29] | Provides model transformation without specifying code for any specific transformations. | Independent of modeling languages, JSON |
| | Sidhu et al. [100] | Refactors UML diagrams with symptoms of design flaws. | UML |
| Genetic algorithms | Ghannem et al. [50] | Interactive repair based on the similarity with the original model. | UML |
| Inductive logic programming | Alrajeh et al. [11] | Framework that combines model-checking with ILP to assure correctness in models. | - |
| | Fumagalli et al. [48] | Framework that combines model-finding techniques and ILP to automatize the repair of conceptual models. | OntoUML |
| | Fumagalli et al. [49] | Assistant to identify domain constraints missing from models. | OntoUML |

may not solve the issues. Finally, by using Boolean logic, the tree is analyzed to obtain which modifications solved the issues. The modifications found are concrete and can be executed automatically to repair the inconsistent model. Alike repairs are grouped and presented to the user as repair options.

Also tree-powered, Khelladi et al. [37] present a model repair approach that ranks repairs depending on the positive or negative side-effect they produce by using a validation tree. They also identify alternative repair paths and cycles of repairs. The approach works with UML models and OCL constraints. As output, a ranking of repairs is produced. Rather than an automatic approach, this approach can be considered as a guide and support system to assist modelers in solving model issues.

Lastly, Model/Analyzer [38] is a tool that, by using the syntactic structure of constraints, determines which specific parts of a model must be checked and repaired. A model with a series of constraints is required as input. The approach is independent of modeling and constraint languages, but the authors test Model/Analyzer on UML models with OCL constraints. This tool deals with a large number of repairs by focusing on what caused an issue and presenting repair actions as a linearly growing repair tree. The tree takes into account the side-effects of solving each issue, avoiding repairs that lead to new issues. At the end of the execution, issues are visualized, showing information about what parts of the model contributed to causing the issues and how to fix them. Knowing their origin may help users to solve the issues and prevent them in the future.

When model repair is performed applying tree learning with Model/Analyzer, issues in the models and repair actions are represented following Boolean logic (e.g., *issue1* could be represented as *c1.attrs==c2.attrs*, and a repair action as

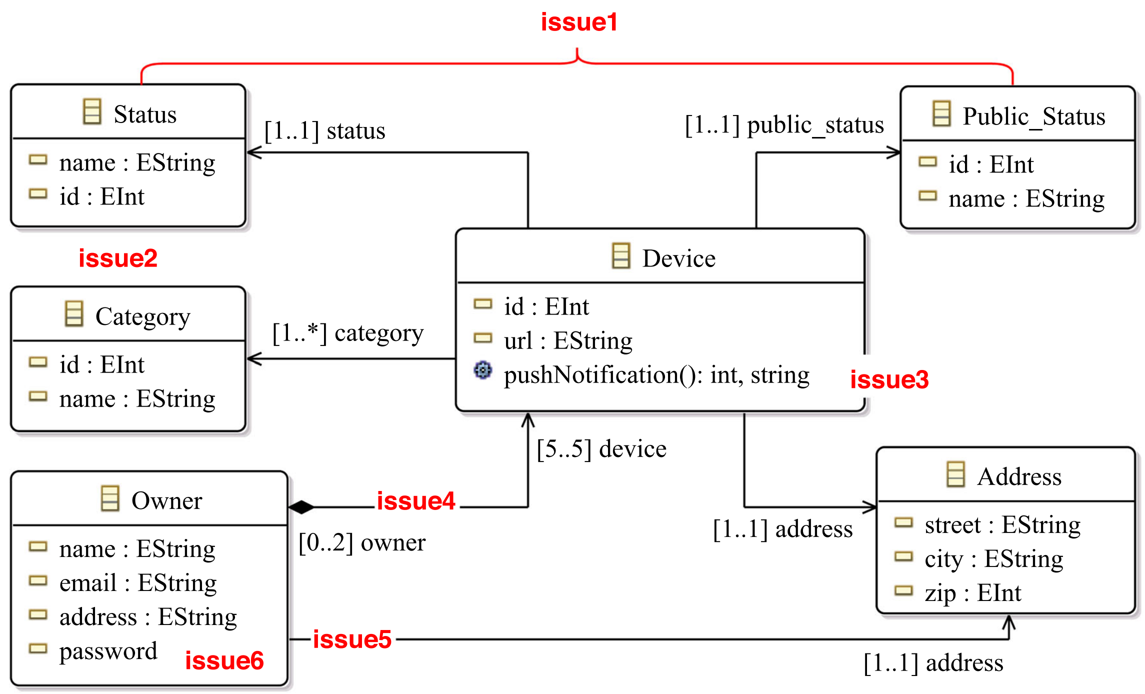


**Fig. 1** Sample model containing a variety of issues

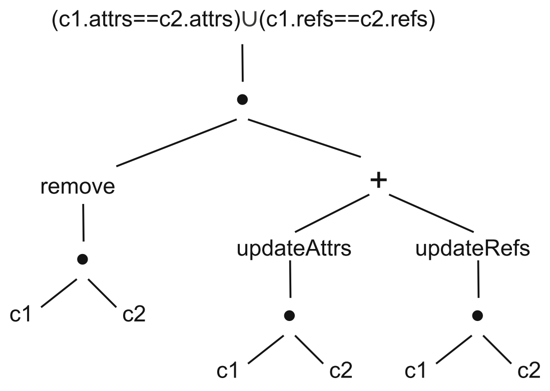


**Fig. 2** Example of a repair tree for the model from Fig. 1

*c1.remove*). Model/Analyzer generates repair trees where the root node represents the logical expression which encodes the issue to be solved, • represent alternatives, and + sequences, respectively, of repair actions. Focusing on the model in Fig. 1, *issues 1* and *2* and a representative sample of their repair actions would be represented as displayed in Fig. 2. In this simplified repair tree, we can see several repair options, the first alternative being to remove one of the classes (*c1* and *c2*) or to update them. If the update is chosen, then a sequence of actions will be necessary to repair both issues: to update the attribute (*attrs*) of one of the classes, and to do the same with one of the references (*refs*). Based on the tree, the users would be able to select the repair solution they prefer.

## 3.2 Automated planning

Automated planning is an AI technique that focuses on the optimization of sequences of actions. Unlike classical classification and control problems, planning solutions are complex and have to be discovered and optimized in a multi-dimensional space. Planning can be classified as a branch of RL [39]. By using automated planning, it is possible to generate plans that lead from an initial state to a defined goal.

Puissant et al. present Badger in [40,41], a tool based on automated planning. Each input model is defined as a series of elementary operations which are needed to create the model (e.g., *create, addProperty, addReference*, etc., see Fig. 3). By default, Badger chooses the minimum repair actions to reach the defined goal; however, users can modify this to prioritize specific types of actions or parts of the model to repair. Their approach is applied to different types of structural issues in UML models. Badger can be adapted to work in different domains. To show this, the authors create a metamodel to represent and repair Java code smells. The planner does not require the user to specify resolution rules manually or to specify information about the causes of the issues, hence the tool is fully automated.

When repairing models using Badger, the planner decomposes the model subject to repair into logical conditions. The planner needs as input: (i) the initial state consisting of the model's logical conditions (see Fig. 3 for an example of how class *Status* from Fig. 1 is represented), (ii) a desired goal consisting of the set of consistency rules that

```
create(c1, class).
addProperty(c1, name, 'Status').
create(att1, property).
addProperty(att1, name,'id').
addReference(att1, type, EInt).
create(att2, property).
addProperty(att2, name,'name').
addReference(att2, type, EString).
```

**Fig. 3** Initial state in logical conditions for class *Status* from Fig. 1

need to be satisfied in the model (in Fig. 1, for example *issue1* would be represented as *forall(class.has(attrs), other_class.has(other_attrs))*, and *issue6* would be represented as *attribute. has(type)*), and (iii) a set of primitive actions that can be performed (e.g., *setType (attribute, type)*). Actions contain a precondition and an effect. The effect of an action is executed only if the precondition is satisfied in the model, for example, the action *setType (attribute, type)* is only applied when the precondition *attribute does not have a type* is met. The model is repaired with a plan consisting of a sequence of actions that removes all issues from the model (e.g., focusing on *issues 1*, *2*, and *6*: to remove one of the faulty classes, attributes, or references, to rename the faulty attributes, and to set a type in the attribute with a missing type). The plan is generated using a recursive best-first search (RBFS) algorithm. The algorithm explores the search space (consisting of the issues existing in the model and the combinations of repair actions to solve them) to find a repair sequence that do not produce any new issues in the model. The sequence of actions chosen is the most optimal in terms of exploration of the search space or with respect to the preferences chosen by the user.

## 3.3 Markov decision process

Over the last years, the authors of this paper have developed PARMOREL [25,42–45], a customizable and extensible model repair framework that enables users to deal with different issues in different types of models.

In PARMOREL, the model repair problem is formulated as a Markov Decision Process (MDP) [30]. MDPs are defined in terms of a finite set of states and a finite set of actions. During our research, we could not find any work within the model repair field using MDPs, hence, in this section we only present PARMOREL. In PARMOREL, the states are sets of issues in the model, and the set of actions is defined by the editing actions available. MDPs are mathematical models used to solve sequential decision-making problems. At specified points in time, a decision agent observes the state of a system and chooses an action. The action choice and the state make the system transition to a new state at a subsequent discrete point in time. The agent receives a reward signal at each transition. The goal of the MDP is to find a policy (i.e., a mapping from states to actions) that maximizes the rewards accumulated over time.

The framework uses RL algorithms to implement the MDP (such as Q-learning or Q($\lambda$)). By using these algorithms, PARMOREL is able to automatically find the best sequence of actions for repairing a broken model according to high-level user preferences. Actions aligned with these preferences will be rewarded.

PARMOREL asks users for their preferences before the repair process starts and then, the repair is performed automatically, giving as output a repaired version of the input model. If users desire further interaction, PARMOREL integrates a mechanism to allow them to manually select a solution from the multiple repair sequences found by the algorithm.

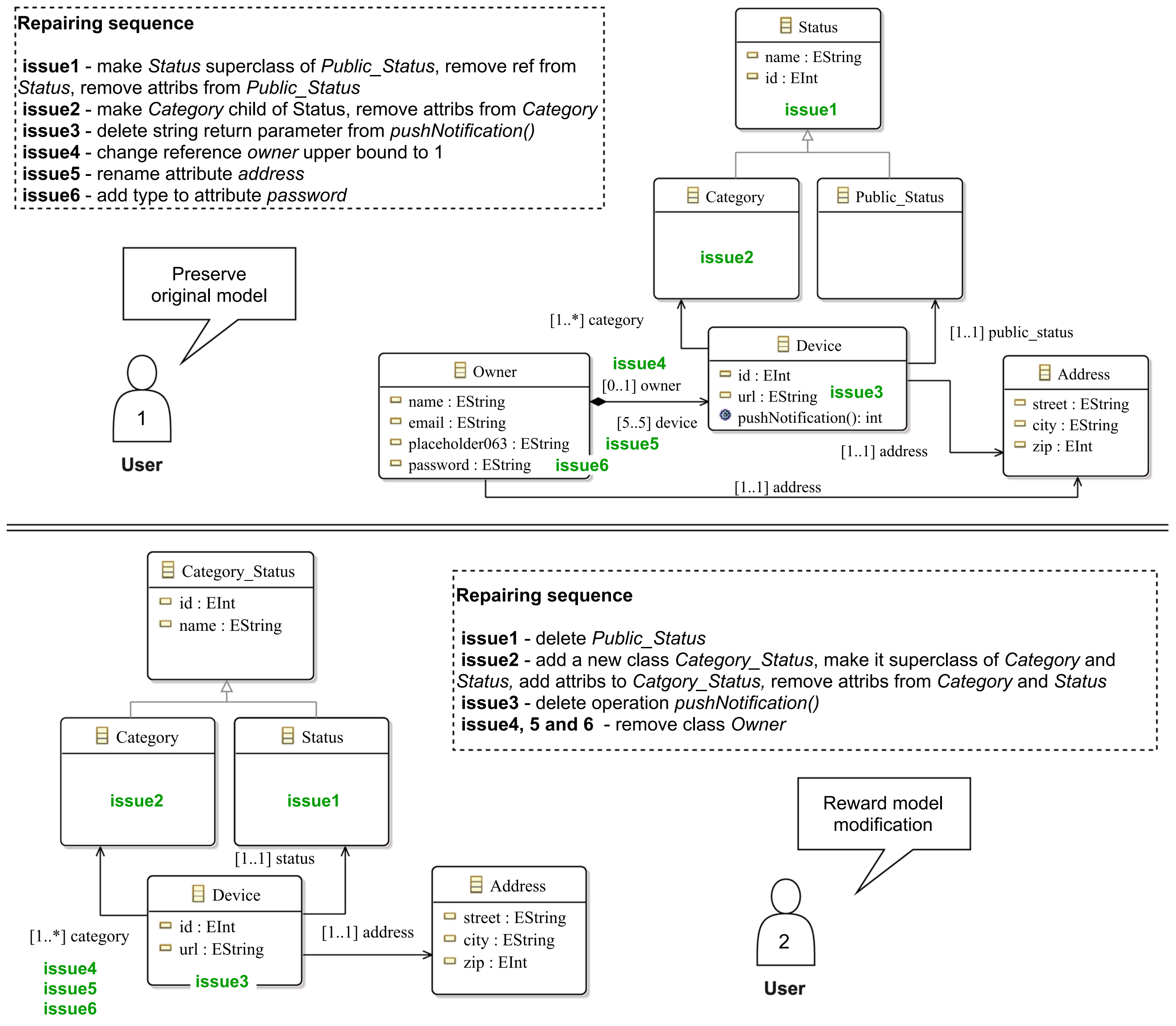
For example, to repair the model in Fig. 1, first, PARMOREL analyzes if there exist any issues in the model. Then, for every issue found, PARMOREL applies different repair actions, obtaining a reward according to how much the selected actions match the user preferences. Finally, PARMOREL selects the sequence of repair actions that leads to the repaired model most aligned with the user preferences. In Fig. 4, we display the repair results produced by PARMOREL when repairing the model from Fig. 1 when a user prefers to preserve the original structure of the model (*User1*), or to modify it as much as possible (*User2*).

PARMOREL has been applied to repair syntactic errors and smells in Ecore class diagrams and to resolve inter-model inconsistencies between UML class and sequence diagrams. To repair these issues, users can select to boost different quality characteristics (maintainability, understandability, complexity, and reusability), to select the repair solution that reduces the model distance between the repaired model and the original one, or to reduce coupling in the repaired model.

The framework consists of a series of modules that can be implemented by users. Modules allow personalizing the type of models to repair, the issues to be addressed, the repair actions, the RL algorithm to use, and the repair preferences [25]. Instead of having a specific tool for each kind of issue and each type of model, PARMOREL works as a unified extensible framework, which can be extended to support the new issues that modelers need to solve in their models.

## 3.4 Neural networks

Some approaches make use of neural network (NN) architectures. NNs are an ML architecture, and there exist different types of NNs able to solve problems within supervised, unsupervised, and reinforcement learning. NNs are inspired by the brain's structure and simulate a net of neurons able to identify patterns and correlations in data. NNs contain multiple layers of a data structure called neurons, which are connected



**Repairing sequence**

**issue1** - make *Status* superclass of *Public_Status*, remove ref from *Status*, remove attribs from *Public_Status*
**issue2** - make *Category* child of Status, remove attribs from *Category*
**issue3** - delete string return parameter from *pushNotification()*
**issue4** - change reference *owner* upper bound to 1
**issue5** - rename attribute *address*
**issue6** - add type to attribute *password*

**Repairing sequence**

**issue1** - delete *Public_Status*
**issue2** - add a new class *Category_Status*, make it superclass of *Category* and *Status,* add attribs to *Catgory_Status,* remove attribs from *Category* and *Status*
**issue3** - delete operation *pushNotification()*
**issue4, 5 and 6** - remove class *Owner*

**Fig. 4** Different repair solutions for the model from Fig. 1 using PARMOREL

with each other. These connections have changing weights that simulate the connection of neurons in the human brain. Connections with stronger weights will be favored and will lead to learning the solution for a given problem. These algorithms allow to perform multiple tasks given a dataset with enough examples from which the network can learn.

One of these tasks can be to repair models. When using a NN for model repair, it is necessary to provide the NN with a dataset representing the issues subject to repair in the model and their corresponding solutions (in the example of Fig. 1, the dataset would contain models including *issues 1-6* and different repaired versions of those models). The goal is that the NN will learn from the dataset how to repair new models not contained in the dataset. Hence, the dataset should contain enough and diverse examples so that the NN can extrapolate how to repair the issues in different models. For example, if in the majority of the training dataset removing the duplicated attributes from *issue1* has led to solving it, then, next time *issue1* appears, the algorithm will decide to repair the model by removing the attributes. The dataset might not include anything related to class *Status*, *Category*, etc. It would work on the level of *class, attribute, reference*, etc. Hence, the dataset will need to include models with a recurring structure. So that the NN can learn, it is necessary to extract and represent the delta between the wrong and the right versions of the models in the dataset. This delta is usually a minimal sequence of operations to get from the broken version of the model to the repaired one. Once the NN has trained enough on the dataset, when a model with issues is given as an input to the NN, the NN will produce as output either a set of actions to repair the model or a repaired version of the model.

In [46], the authors present a NN for model transformation without specifying code for any specific transformations. Although not specifically model repair, we consider this approach close enough to be included in this section. They make use of a dataset of UML models generated by a Java program, but the approach is open for other modeling languages. Models are stored in a tree structure using JSON formatting, the root contains the keyword MODEL, and its children are the model elements. Then, the network transforms the input models into their corresponding output models extracting the model transformation needed.

Tackling model refactoring, in [47] the authors make use of a deep NN architecture to refactor UML diagrams with symptoms of design flaws. In this approach, the deep NN learns to recognize the presence of functional decomposition in UML models of object-oriented software, producing as output a refactored model without flaws. They use a dataset comprising feature vectors of distinct UML class diagrams. They obtained the dataset from the UML-Ninja repository [48] and extracted metrics from them by using SDMetrics [49]. In both approaches, the authors claim their results are promising but there are still a series of open challenges needed to be addressed, such as the size of the training dataset, diversity of data, and generality.

## 3.5 Genetic algorithms

Genetic algorithms are used to generate solutions to optimization and search problems by using biologically inspired operators such as mutation, crossover, and selection. These algorithms fall under an AI branch called evolutionary algorithms; however, since they aim to solve similar problems as RL algorithms (searching for solutions that maximize or minimize a reward or cost function) and behave in a similar way (finding a solution by interacting with an environment with no training data), genetic algorithms can be considered a branch of RL [50].

When solving the model repair problem with genetic algorithms, the goal of the reward or cost function will be to remove the issues existing in a model (such as *issues 1-6* in Fig. 1). In this sense, genetic algorithms are similar to MDPs in terms of goals; however, they select the actions to remove issues differently. To remove the issues, the genetic algorithm will get as input a set of repair actions (e.g., for *issue1* the available actions would be to remove one of the classes, update or remove the duplicated attributes, create a hierarchy with one of the classes, etc.). A subset of the actions available to repair, called "population" will be selected in each iteration of the algorithm. Some actions will be forwarded to the next iteration or removed from the population, depending on how much they maximize or minimize the reward function of the algorithm. They will be chosen also by a random factor, which will mix actions that otherwise could have never been selected. For example, the first population could be to remove *issues 1, 2*, and *6* by removing one of the classes with repeated features and removing the untyped attribute. Then, this last action could be forwarded to the second population, which would also include to remove *issues 1* and *2* by updating the duplicated attributes and reference. When the algorithm finishes mixing populations, the remaining population will be a set of actions able to remove the issues existing in the model.

In [51], the authors present an approach based on an interactive genetic algorithm. Unlike classical genetic algorithms, in this work, the authors include interaction with the user to take their feedback into account. These algorithms use a fitness function to determine the goal to solve. In this paper, the authors make use of a fitness function that combines the similarity between the analyzed UML design model and models from a base of examples, and the modelers' feedback. Modelers introduce their feedback after some iterations of the algorithm, indicating which solutions from the ones found they prefer.

Users can specify different parameters, such as the percentage of solutions shown in each interaction. The tool takes as input a base of examples of refactored models and an initial model to refactor, then, it generates as output a sequence of refactorings to be applied on the input model. This tool is developed as an Eclipse plugin. As dataset, they use Ref-Finder [52] to extract refactorings performed in different Java projects.

## 3.6 Inductive logic programming

As stated in Sect. 2.2, ILP uses logic programming as a computational mechanism for representing and learning a hypothesis from incomplete or incorrect background knowledge and a set of examples [53]. Logic programs are based on a set of rules. The task for the learning algorithm is to compute a hypothesis that extends the background knowledge to comply with the set of positive examples without covering the negative ones.

When repairing models with ILP, the set of positive examples will include models without issues and the negative ones will include models with issues (e.g., different variations of the issues, for example in different parts of the models). As happened with NNs, diversity in the set of positive and negative examples will increase the ability of the algorithm to find solutions to new models. The background knowledge will be the model subject to repair and its issues (e.g., the model represented in Fig. 1), and the hypothesis the set of actions to remove the issues (e.g., all actions available to repair *issues* 1–6).

In [53], Alrajeh et al. propose a framework that combines model-checking with ILP to assure correctness in models.

Model-checking is a method for verifying whether a finite-state model of a system meets a given property, usually by generating counterexamples to test the property [54]. As a first step, model-checking is used to check if a model satisfies all its desired properties, and if it does not, the system looks for counterexamples that satisfy the desired properties in the model. Next, the logic-based component of the framework takes these counterexamples and traces of the violation of the property and finds solutions from the hypotheses space. Finally, the selection of a solution is domain-dependent and requires input from a human domain expert. There might exist multiple violations to multiple properties in a model that require several iterations of this process. The authors have evaluated the approach by addressing a variety of requirements-engineering problems and validating it against several benchmark studies.

Fumagalli et al. [55] propose another framework to automatize the repair of conceptual models. They use a combination of model-finding techniques and ILP. In model-finding, given a first-order logic formula and a domain of interpretation, a finder assesses whether the formula is satisfiable in that model [56]. Besides the difference between model-finding and checking, this approach follows a similar process as the framework from Alrajeh et al. First, by using model-finding, they generate positive (intended) and negative (unintended) model configurations. Then, to address the unintended configurations, this set of positive and negative examples is used for feeding the ILP learning process. A set of possible solutions is generated and finally, one is chosen by the "knowledge engineer." The work presents a proof-of-concept of the framework through the solution of a sample model. Their objective is to work with OntoUML models and learn from the configurations structures that are recurrent in them, such as patterns.

Also Fumagalli et al. [57] use ILP to assist modelers in identifying domain constraints that are missing from their models. Identifying the necessary constraints in a model will allow to keep that model free of issues in the future and to create more precise models regardless of the expertise of the modeler in the models' domain. The ILP process here is three-folded and can be grouped into three main phases: generation, assertion, and induction. In the generation phase, a model-finding process is executed, creating different versions of the original model. These versions are combined with the original model in the assertion phase, creating new instances of it which can be used by the modeler to elicit negative and positive examples. Lastly, the induction phase gets as input the negative and positive examples and the original model. The output is a set of logical constraints that can be used by the modeler to complete the input domain model, i.e., to make the model satisfy the positive and avoid the negative examples.

## 4 Challenges

The adoption of AI within MDE has already been identified as a challenge for the community by authors in [8,58,59]. This adoption is challenging due to specific aspects of MDE and the model repair field, but there also exist some challenges that are transversal to the application of AI in any field. In order to address these challenges, the community needs to be aware of them and start working on their resolution, otherwise, the AI-powered model repair field will eventually fall into stagnation.

Hence, in this section, we present challenges the modeling community has to overcome so that the research of AI-powered model repair can grow and become a more mature field. Figure 5 summarizes the challenges identified throughout this section.

### 4.1 Data

The main challenge for AI adoption in MDE in general, not only in model repair, is about data. Many well-known ML algorithms (like those focused on classification and regression) depend on large amounts of data to learn how to repair a problem [27]. This challenge has already been identified by other researchers working on AI approaches in MDE [46,51].

As stated in [8,9,16], the available modeling repositories (such as REMODD [60], ATL Zoo [61], MDEForge [62], GenMyModel [63], Img2UML [64] (now called UML-Ninja) or the ones presented in [65] and [66]) contain an amount and diversity of modeling data which is still limited. Future repositories should contain a curated catalog containing a diverse range of modeling data [8], including good and bad examples, model instances, metamodels, transformations, etc.

Not only more repositories and with greater variety are needed, but the data offered should be sufficiently labeled and following a proper structure [9]. Without labeling, many ML algorithms would not be able to make use of the data. Hence, an interesting research challenge for the community would be to make a collective effort in labeling new and existent datasets. Additionally, the community should create guidelines about the minimum quality and curation a modeling dataset would require. For example, as stated in [67] a common problem among code datasets is the number of duplicate files (up to 40%) they present, a problem expected to happen in modeling repositories as well when they grow in size and number.

As presented in Sect. 2.2, there exist ML approaches that can work without much data, such as RL and logic-based approaches. So the model repair problem can still be solved by ML despite this data challenge. However, being limited in terms of data reduces considerably the scope of researching how ML can improve model repair, and makes it impossible

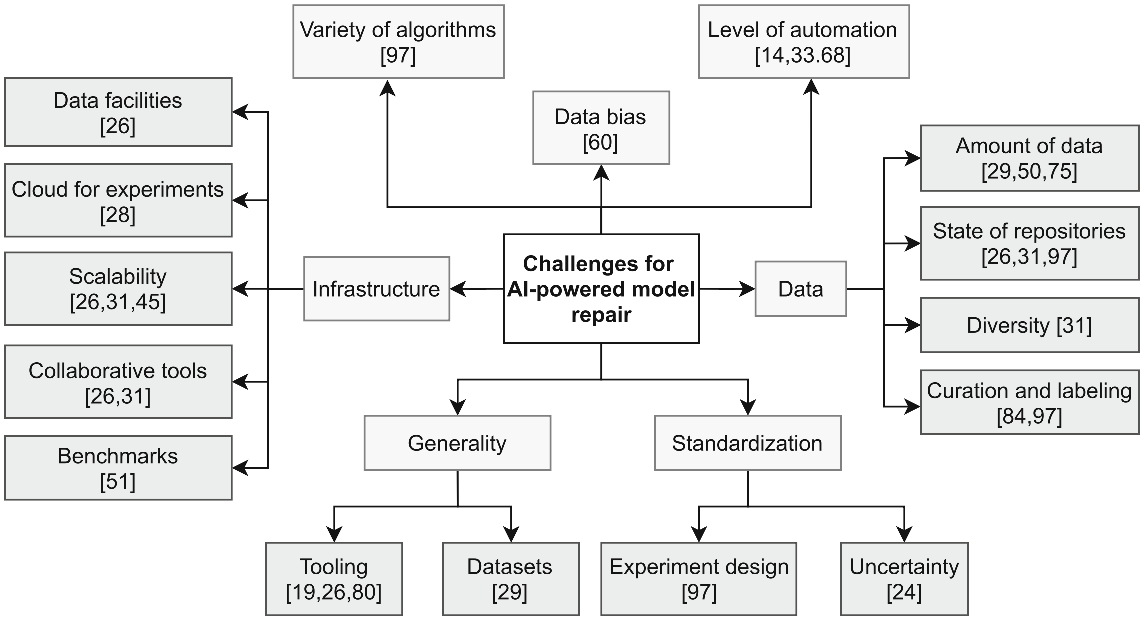

Challenges for AI-powered model repair
Variety of algorithms [97]
Level of automation [14,33.68]
Data bias [60]
Infrastructure
Data facilities [26]
Cloud for experiments [28]
Scalability [26,31,45]
Collaborative tools [26,31]
Benchmarks [51]
Data
Amount of data [29,50,75]
State of repositories [26,31,97]
Diversity [31]
Curation and labeling [84,97]
Generality
Tooling [19,26,80]
Datasets [29]
Standardization
Experiment design [97]
Uncertainty [24]

**Fig. 5** Research challenges identified for AI-powered model repair

to adapt and apply some of the state-of-the-art algorithms that are providing promising results in other fields (such as new DL approaches).

Furthermore, it would be interesting to start a discussion about the structure future repositories should have, in order to reach a consensus on new datasets produced. By following a similar standardized structure, less effort would be needed to apply AI techniques in datasets produced by other researchers. The modeling community can learn from the dataset problems that have already appeared when applying AI in SE problems. In this direction, as pointed out in [9] imbalanced size of datasets was identified as a major obstacle in evaluating the AI techniques empirically. At the same time, a lack of generality and overfitting problems appeared in cross-project datasets.

## 4.2 Infrastructure

To extend AI adoption, we stress the need for not only more modeling data to support research, but also facilities to make it easier for modelers to store and acquire such data [16]. Hence, it would be important to create not only static repositories just containing data but interactive platforms where other modelers could store their datasets. In this direction, works like [68] could offer a common scalable infrastructure to reproduce and replicate experiments.

Scalability itself is a challenge identified in [8,16,69]. It is expected that the demand for working with very large modeling artifacts will continue growing. Hence, the community will need to focus on improving the performance of modeling infrastructure, fragmenting and splitting techniques for dealing with models, and performance optimization. Additionally, in [8,16] it is stressed how, when powered by AI, tools should be able to support collaboration between users. Hence, it could be interesting to discuss and create protocols for future collaborative modeling tools.

Additionally, it would be interesting to create benchmarks for researchers to compare their own datasets and tools. In this direction, Gogolla et al. [70] propose a benchmark to assess the validation and verification of techniques on UML and OCL models. Creating a common benchmark for model repair would benefit the field by providing a better understanding of how different tools, approaches, and datasets relate to each other.

## 4.3 Standardization

Standardization will be required with respect to the experimentation process and results of repairing with AI. As stressed in [9], due to the stochastic nature of AI, there might be big differences in results when executing the same experiment multiple times, even over the same input data. This reduces the validity of experiments as well as the trust level of practitioners toward AI techniques.

Additionally, it would be desirable that the community reached a consensus on how to measure and fight against uncertainty in AI-powered model repair. Works like the one presented by Bertoa et al. in [71] could serve as inspiration for dealing with uncertainty.

### 4.4 Generality

Mussbacher et al. [72] discuss that most AI applications in MDE are tailor-made for solving specific problems. However, regarding model repair, the diversity of type of models, issues to solve, and how to solve them make time-consuming to build individual solutions to address each of these nuances [25]. This diversity is a challenge for scalability and producing tailor-made solutions slows the development of the AI model repair field. Hence, in a desirable future, the community would avoid from-scratch solutions and adopt more general solutions based on standardized protocols, like modular frameworks or plug-and-play settings [16].

This challenge is far from trivial. Some algorithms such as NNs have difficulties in predicting output solutions for input models differing from the training distribution they have learned [46]. More research will be needed in this direction, focusing on how to treat datasets to achieve more general results.

### 4.5 Data bias

Bias has been pointed out in the literature as one of the biggest challenges for AI adoption in general [73]. An AI algorithm's performance and results will be as good as the data it is fed with. If the data used as input is biased, this is, inclined toward certain results or omitting some representative samples, the results will also be biased and hence, faulty. Bias regarding model repair could happen, for example, if we only collected models designed by students. This could lead to algorithms choosing the same repair options that would have been chosen by students, to assume that all issues in models are those that appear at a student level or that every model looks like and has the size of those designed in a classroom.

Future model repositories should be populated with data coming from diverse modelers. Data should not only be diverse in terms of the kind of models it contains, but also in terms of the modelers creating this data. These models should be produced by people with different experiences in modeling, age, gender, occupation, domain expertise, etc. Otherwise, the results obtained will be partial, hard to trust, and difficult to generalize.

### 4.6 Level of automation

Citing Macedo et al. [22]: "*Since the selection of the most suitable repair update is ultimately a choice of the developer, approaches to model repair must balance the level of automation of the process with the need for user guidance in the generation of the alternative solutions.*" Fully automated methods lead to overgeneralized solutions that are not always adequate, and strong interaction comes with a high computational effort, therefore, in a desirable future, model repair tools should seek an equilibrium between automation and interaction [24].

Although automation frees the user from repetitive and manual tasks, thus saving time for more critical tasks, it is important to adapt its level to the preference of the user. In some domains it might be enough with using data collected specifically by the user, to use their preferences as input or to keep some degree of interaction during or after the repair process.

In this direction, some authors in the literature [74] are advocating for keeping the user in-the-loop, offering them interaction during the learning process, and sometimes incorporating their knowledge and feedback into the algorithms' training.

### 4.7 Variety of algorithms

In the current state of the AI-powered model repair, as presented in Sect. 3, most approaches fall within the RL scope , mainly due to the lack of data available to apply supervised or unsupervised algorithms. The approaches that fall under the supervised learning scope do not use a lot of data or are at an initial research stage. However, as more data becomes available, it might happen that supervised learning almost monopolizes new research.

This is happening nowadays when applying ML to software engineering [9]. In the papers analyzed by Shafiq et al., more than 70% of the approaches applied some kind of supervised algorithms. As the AI-powered model repair is still an emerging field, it would be desirable that a variety of AI branches are researched, hence obtaining more diverse results. Only by doing so, we will be able in some years to know which AI techniques are more appropriate for model repair, or some domain-specific scenarios.

## 5 Opportunities

More and more approaches within the software development field are rapidly adopting AI to create intelligent, self-learning systems [9] and to cognify [8] previously human-performed tasks. In [9], the authors analyze the number of articles published about applying ML in software engineering, taking into account those published between 1991 and 2019. They found out that almost two-thirds of the articles were published between 2016 and 2019. This explosion of research represents an opportunity for developing the AI-powered model repair field. This momentum is expected to continue in the coming years and AI-related research is likely to be encouraged and rewarded. The model repair field can take an advantage of this momentum by getting inspiration from what is being done in other fields and performing experimental research that otherwise would remain unnoticed.

In this section, we explore how AI is being applied in model-related fields (such as software engineering and code repair) and in other modeling problems beyond model repair. Then, we analyze how that research could be applied into the model repair field. From the explored works, we identify research opportunities to apply AI in model repair. Figure 6 summarizes the opportunities identified throughout this section.

## 5.1 Prediction

In [9], the authors present a state-of-the-art study about the adoption of ML in software engineering. The categories they identify can serve as an indicator of how ML could provide further applications within model repair. Indeed, they found that most of the works focus on "Quality assurance and analytics," mostly in bug prediction and verification, which are important aspects to model repair too. By further developing the early identification of issues in models, the repair could go from being reactive to proactive, repairing models as soon as issues appear, requiring minimal intervention from the modeler.

In this direction, as an example, in [75], the authors perform a comparative study with different ML techniques for identifying a set of four smells. They achieve high accuracy without needing much data for each smell. Furthermore, by achieving issues prediction, modelers could be aware of which models are more prone to get corrupted, and with enough advance, the repair process could be even prevented. Regarding code refactoring, different ML techniques [76,77] have been applied to predict and identify which parts of the code are prone to be refactored. By doing so, the time spent on refactoring is reduced.

## 5.2 Inspiration from code repair

The fields of code and model repair are similar enough so that the approaches built in code repair can serve as an inspiration to solve problems within the model repair field. Hence, there is an opportunity to study and adapt what is being done in code repair to create new model repair approaches.

Focusing on automatic code repair, several approaches have been proposed, mostly using the newest types of NNs and DL. In [78] authors feed a deep NN with code errors and their following changes to learn how to provide repair automatically. Also using DL, White et al. [79] produce repair patches taking into account code similarities to produce faster solutions. With the appropriate amount of modeling data, these techniques could be applied to model repair, taking approaches that use model history [4,23] one step forward.

Other approaches which do not require labeled data to work could be especially interesting for the model repair field. For example, authors in [80] make use of a set of generative adversarial NNs to repair software vulnerabilities. Another example can be found in a search-based approach presented by Moghadam et al. in [81]. In this work, the authors present Code-Imp, a tool for refactoring Java pro-

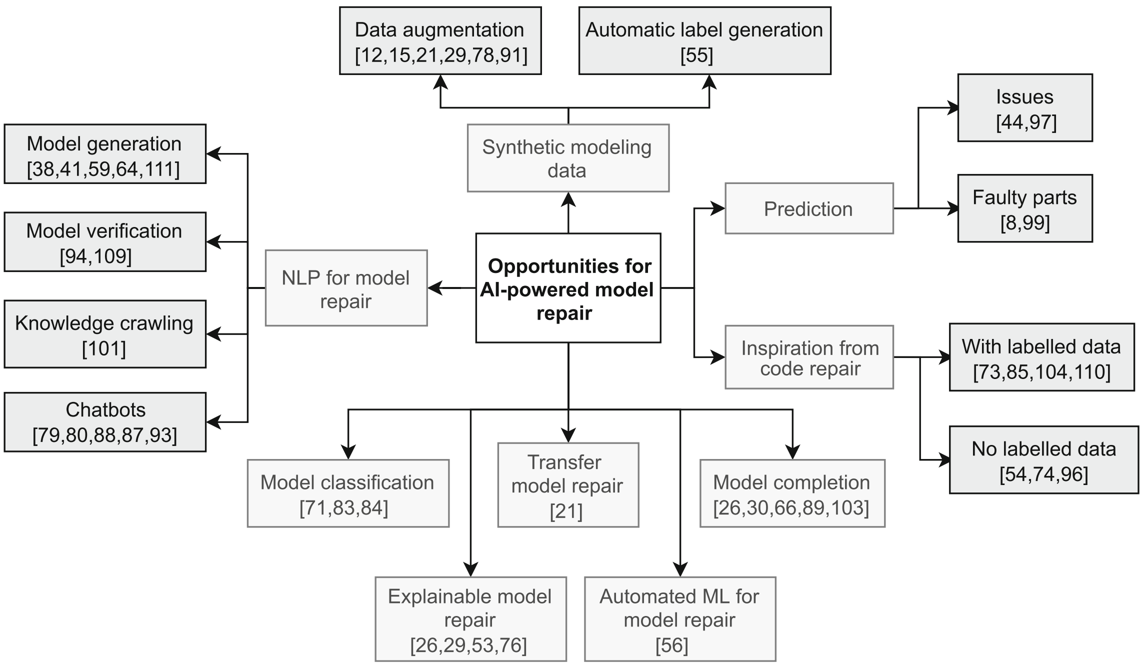

Data augmentation [12,15,21,29,78,91]
Automatic label generation [55]
Model generation [38,41,59,64,111]
Synthetic modeling data
Issues [44,97]
Model verification [94,109]
Prediction
Faulty parts [8,99]
NLP for model repair
Opportunities for AI-powered model repair
Knowledge crawling [101]
Inspiration from code repair
With labelled data [73,85,104,110]
Chatbots [79,80,88,87,93]
No labelled data [54,74,96]
Model classification [71,83,84]
Transfer model repair [21]
Model completion [26,30,66,89,103]
Explainable model repair [26,29,53,76]
Automated ML for model repair [56]

**Fig. 6** Research opportunities identified for AI-powered model repair

grams based on quality metrics that achieves promising results at code-level by using hill-climbing algorithms [82].

### 5.3 Model completion

Another popular application of ML within programming is code completion. In these approaches, an ML system is fed with code examples and uses them to predict how new code will be completed. For example, authors in [83] make use of DL to complete Python code, in [84] of mixed NN to complete Python and JavaScript code, and in [85] of Bayesian NNs to complete Java code.

The modeling community has already identified model completion as an opportunity to improve the autonomy of MDE tools [16]. With the appropriate modeling data, model completion based on what has been done on similar models would be possible. Within repair, this could be applied to see how similar models have been constructed, using them as suggestions for how to repair. Model completion could provide faster repair, especially in common issues that are repeated in different models. Additionally, it could provide new tools for teaching MDE, ease the repair process to novice modelers or modelers without enough expertise in the repair field.

In this direction, in [86] Burgueño et al. present a natural language processing (NLP) approach that provides autocomplete suggestions of partial domain models. This approach is also related to the NLP opportunity presented a couple of paragraphs below. They use the textual information available for the modeling project and its related domain, as well as the user's feedback to create the completion suggestions.

### 5.4 Explainable model repair

AI systems are often designed as black boxes. They output a result without providing any information or reasoning about how that result was obtained, making sometimes solutions non-intuitive and difficult to understand [87]. Authors in [46] identify as a challenge the social acceptance of AI algorithms within MDE. They claim that users may be reluctant to trust a piece of software that they are not able to understand due to the black-box nature of these algorithms.

Explainable artificial intelligence (XAI) aims to break this black-box nature by providing AI results together with explanations about their origin. This could increase AI trustiness. We can take as an example the work of Monperrus et al. [88]. Here, the authors envision how XAI could be applied in automatic bug fixing.

The community has already identified the potential of XAI for reasoning on when model changes are done [16]. XAI would improve AI-powered model repair by providing explanations of why a repair was preferred over others or why certain changes were made in a model. This could take recommender and semi-automatic approaches a step beyond, since users would have more information available before deciding what changes to perform in the model.

### 5.5 Natural language processing for model repair

A lot of work has been done in the last years in applying NLP to solve modeling problems. NLP improves human-computer interaction as it allows computers to understand natural language, so users can communicate with them in a natural and simple way. In this direction, several works have been proposed to automatically generate models from user requirements [89,90], transformation actions [91], and user stories [92]. Other authors [93] make use of NLP to verify the correctness of UML models and to recommend relevant domain concepts to rename metamodel elements [94].

Not within MDE but also using NLP, authors in [95] propose a recommender system for programming tasks. The system is fed with knowledge obtained from programming forums such as StackOverflow. Making use of NLP, given a query, it provides programmers with the most appropriate solution and comprehensive code examples, thus saving search time. By crawling available MDE forums, this same approach could be applied for model repair.

There exist open-source approaches, as Huggingface [96], which provide a collection of pretrained ML systems and offer an open API extensible to fine-tune it to the specific needs of each project. Using pretrained and extensible APIs such as Huggingface could ease the development of NLP systems for model repair, especially within domain-specific modeling scenarios.

An interesting branch of NLP is the use of chatbots. Users interact with an AI-powered bot by exchanging chat messages. Regarding modeling, chatbots have been applied in collaborative modeling environments [97] and to ease modeling tasks such as making a model instance conforming to its metamodel [98] or abstracting model querying from specific languages [99].

By applying NLP and chatbots to model repair, users could define in an intuitive and natural way their repair preferences, criteria to consider a model as broken, consistency rules, repair actions, etc. The application of this technology could make the field more accessible, especially to students or new modelers, a challenge already identified by authors in [59]. For example, it would not be necessary anymore to know OCL in order to define consistency rules. Adapting the modeling language to the modelers' expertise and at the right skill level would also lead to an improvement of the quality of the modeling process and the resulting modeling artifact [72]. Furthermore, a chatbot could act as an assistant that notified users when models require repair and allowed interactive repair. Likewise, chatbots could be used in online or collaborative model repair environments.

### 5.6 Model classification

Other approaches focus on using classification techniques in different modeling tasks. In [100] authors use a nearest neighbor search mechanism to identify whether student modeling assignments were plagiarized. Nguyen et al. [67,101] use different types of NNs to group models contained in datasets by similarity.

These techniques could be applied in model repair to detect similarities in a set of broken models, both in the models themselves and the issues they might contain. Then, similar models could be repaired together, either automatically, reusing a similar kind of repair, or by allowing some degree of interaction from the user. By presenting users with batches of models broken in a similar way, fewer steps might be necessary to repair them all. Classification has the potential to reduce the repair time and to ease the repair process, especially when dealing with large sets of models. By ordering and presenting similar models together the repair process would become more intuitive. Additionally, classification could be applied when creating or working with model repositories as a way to organize or even labeling them.

### 5.7 Transfer model repair

Transfer learning (TL) is a research line in ML that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem to solve it faster [102]. TL permits sharing and reusing the experience gained in different users' repairs. Hence, repair time can be reduced by avoiding repeated calculations for errors to which a solution is already learned.

In PARMOREL [44], we have applied TL to reuse learning when repairing the same type of models and issues for users selecting different preferences. In the future, as more AI-powered model repair approaches appear and a bigger variety of algorithms are applied, it will be interesting to study how knowledge gained can be transferred not only within an approach but between different approaches. Likewise, in other fields it is quite common to work with pretrained NNs and to re-train just their last layers, hence learning new knowledge but reducing the training time by taken advantage of the already known from solving other problems.

### 5.8 Automated ML for model repair

Automated ML (AutoML) [103] is a field of ML that focuses on using ML methods themselves to find out which is the best ML algorithm to solve a given problem and which hyperparameters values work better for the chosen algorithm (tuning of configuration values which modifies how an ML algorithm behaves).

In order to design ML architectures, it is needed some expertise in the field, which most modelers may lack. This expertise comes from holding knowledge about coding, statistics, mathematics, and ML itself, so it is not an easy-to-acquire skill. The application of AutoML could ease how modelers design their ML systems, serving as an entry-point to modelers with valuable MDE knowledge but illiterate in ML. Following this approach, modelers that nowadays are discouraged by the difficulty of the field might feel motivated to pursue their own AI-powered model repair research. Hence, the amount of research performed in the field could potentially grow.

### 5.9 Synthetic modeling data

As stated in the data challenge in Sect. 4.1, the lack of data is the biggest challenge to overcome in order to produce more diverse AI-powered model repair research. Until more data become available, there are some techniques that could help with experimenting with few data.

One of them is data augmentation, a technique that consists of increasing the amount of data by adding slightly modified copies of already existing data. It is also able of creating newly synthetic data from existing data. Some of the approaches presented in this paper use different data augmentations techniques. For example, authors in [40] use the approach proposed in [104] to generate big-sized UML models, in [44] we used the tool presented in [105] to mutate already existing EMF class diagrams, and authors in [46] use a Java program to generate a model transformation dataset. Data augmentation could be the way not only to improve experimentation in AI-powered model repair but also to create new datasets and repositories.

In other fields, different types of NNs are being applied to perform more and more complex data augmentation, with the goal of making synthetic data as close as possible to the real one. In this direction, authors in [106] make use of a generative adversarial NN to create a dataset in such a sensitive field as Parkinson's research. Experimenting with different AI techniques to augment data, especially with newer NNs, could improve the quality of newly created modeling datasets.

Another technique is automatic label generation, where different algorithms, such as DL [107] can be applied to automatically label a dataset, only needing a sample of similar data labeled. As stated in Sect. 4.1, not only the lack of data is a problem, but also the fact that it is mostly unlabeled. Labeling can be a time-consuming task if performed manually, hence, automatic label generation could accelerate the creation of labeled datasets.

## 6 Lessons learned while developing PARMOREL

As stated in Sect. 3, the authors of this paper have developed over the last years PARMOREL, a framework for automatic and personalizable model repair based on RL [25]. We consider that presenting our experience might be useful for other researchers, hence, in this section, we discuss the lessons we learned while developing PARMOREL, connecting them to the challenges presented in Sect. 4 and the consequences these challenges had in the development. The key points from this section are summarized as lessons composed by challenges, consequences, and solutions in Fig. 7.

When we decided to tackle the model repair problem using AI, we started by analyzing AI techniques and what was being done in the code repair field. As stated in Sect. 5, most code repair approaches use ML algorithms that depend on data that is still not available in the modeling field, which is also connected to the data challenge in Sect. 4.1. Due to this situation, we opted to use RL, since these algorithms can solve problems without needing large or labeled data. The limited availability of data reduced the scope of our research.

We have experimented with different kinds of models: Ecore and UML models, which contained different kinds of issues to repair: syntactic errors, smells, and inter-model inconsistencies. Hence, although we did not need data to train our algorithm, we needed a diversity of subject models for our experiments, (diversity is identified as a challenge within data in Sect. 4.1). Despite just needing raw data, without labeling or previous curation, obtaining these models was a difficult task. Due to the lack of data, sometimes we had to mutate and introduce issues in the models ourselves, to create models inspired by what was available in some repositories, and even to reduce the scope of our experiments. For example, when working with inter-model consistency, we focused on restoring the consistency between UML class and sequence diagrams. We needed to find pairs of these models with inconsistencies, and we were able to find isolated class and sequence diagrams, but not representing the same system, let alone with inconsistencies. We can say that the lack of data, as stated in the data challenge in Sect. 4.1, has been the biggest challenge for our research.

PARMOREL follows a modular structure that users can extend to implement different functionalities. For example, by extending the preferences module, users can include new repair preferences in PARMOREL. This way, users can select preferences that customize the repair results, for example, improving different quality characteristics. For implementing these modules, we rely on external tools. For instance, when working with inter-model consistency we used as preference the reduction of coupling in the sequence diagrams. To measure the coupling in the diagrams we used SDMetrics [49]. During our experimentation, we found that SDMetrics was able to measure the coupling when working with models created with UML Designer 9.0, but not when the models were retrieved from GenMyModel [63]. Even when working with the same kind of models, a slight change in their format or structure made them unsupported by the tools or, the tool provided different results. This is aligned with the generality challenge presented in Sect. 4.4; not only tools are tailor-made for a single problem, but they are too rigid regarding the data they use. This increases the difficulty of replicating

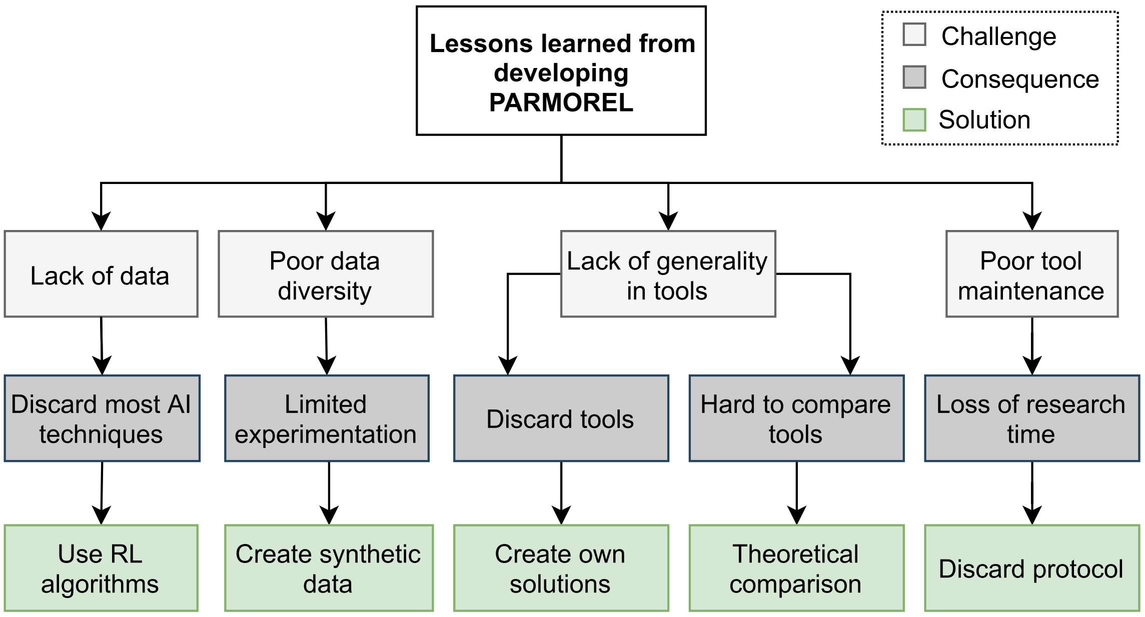


**Fig. 7** Summary of challenges faced during PARMOREL's development together with their consequences and solutions

experiments as well as combining and working with different tools. Since, sometimes, we could not reuse some of these tools, we overcame this challenge by implementing our own solutions.

Some tools also present problems regarding their availability and maintenance (e.g., broken links, lack of documentation, outdated requisites). Installations often require old IDE distributions, hard-to-find libraries versions, and the installation requisites are sometimes not properly documented. Dealing with this is a task that takes time from actual research and experimentation. Eventually, we designed a discard protocol and by following it, tools that did not meet some requisites were discarded.

Regarding the infrastructure (specifically benchmarking) and again the generality and data challenges from Sects. 4.1, 4.2, and 4.4 at the moment, it is quite hard to compare the existing AI-powered model repair tools. The same challenge exists for general model repair tools; it is not straightforward to compare various tools due to the lack of benchmarks. Most tools focus on repairing a specific type of issues on a specific type of models, which differs from what other tools do. Hence, the results from each tool are obtained under different settings. This threatens the validity of the experimentation, making it harder to objectively compare different tools. Hence, in most cases, only a theoretical comparison is feasible by analyzing the techniques each tool use.

## 7 Discussion

In this section, we briefly discuss the outcomes of our analysis of existing AI-powered model repair approaches, challenges, and opportunities for AI-powered model repair. Furthermore, we analyze how the identified challenges are reflected in the identified approaches and the opportunities (see Fig. 8), providing a more general picture of the state of the field. Additionally, we present how challenges (resp. opportunities) may affect each other (see Fig. 8). This discussion is based on our experience within the AI-powered model repair community and the lessons we learned while developing our own approach (see Sect. 6).

*Exploration of approaches* Section 3 presented an exploration of approaches within the AI-powered model repair field, showing the research done when combining AI and model repair. We have presented existing approaches in six subsections: tree learning, automated planning, Markov decision process, neural networks, genetic algorithms, and

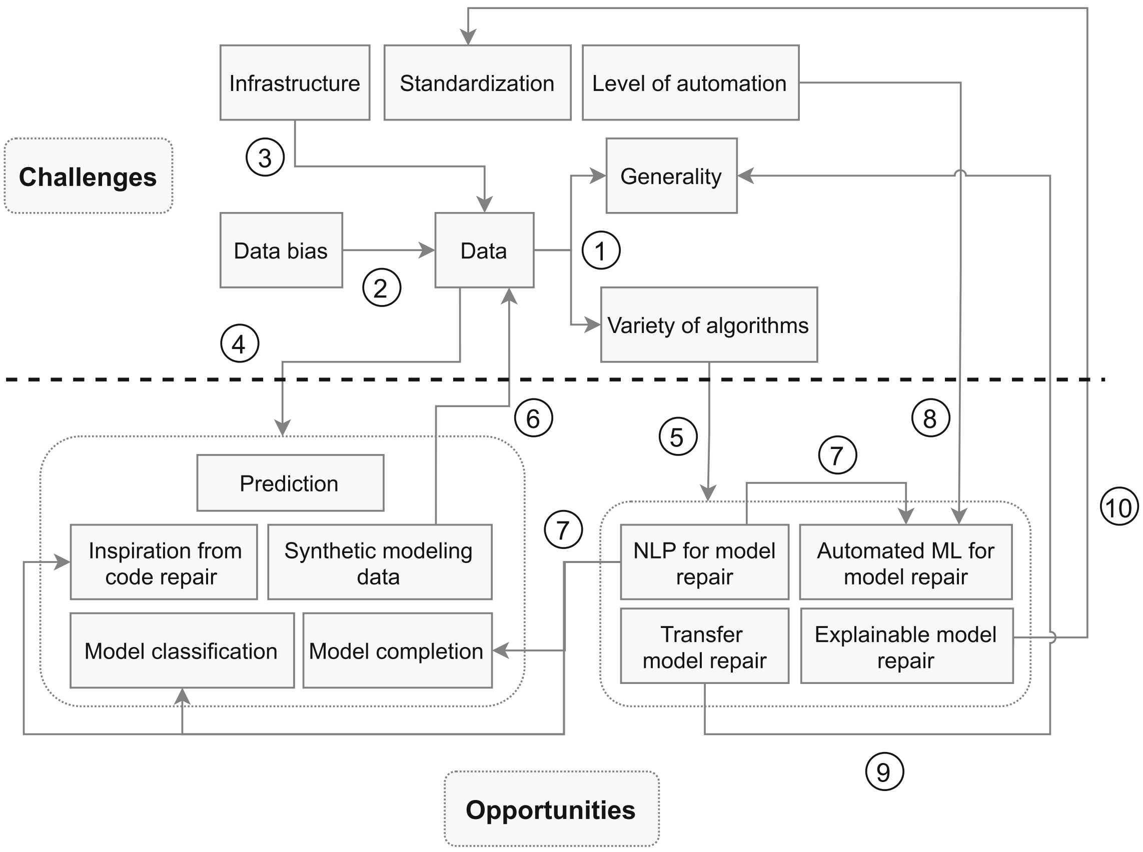


**Fig. 8** Relationship between the challenges and opportunities

inductive learning programming. Most work has been done in the tree learning category. Except for works related to NNs, all the other approaches have in common the use of ML techniques that do not require large amounts of data or labeled datasets to work.

This is aligned with the main challenge presented in Sect. 4.1: the limited amount and current state of the data existing in the modeling community. It is promising that researchers have been able to produce interesting research despite this challenge. However, it is necessary to address it and work to produce more datasets and repositories. Otherwise, only a very limited range of the AI field will be researched, reducing the potential benefits it could provide to model repair.

*Challenges* Then, in Sect. 4, we presented research challenges for the community within the AI-powered model repair field. Here, we identified seven categories of challenges: data, infrastructure, standardization, generality, data bias, level of automation, and variety of algorithms.

As stated in the previous paragraphs, those challenges within data are the most important and urgent to solve. The lack of proper data is the main reason why AI is not so adopted in the MDE field in general [46,51], not only in model repair. The community is already doing efforts in increasing the interest in combining AI and MDE. Focusing on data, there are workshops like the Workshop on Analytics and Mining of Model Repositories (AMMoRe) [108]; however, there is a need for more emphasis on the problem of data within the field. With the increase in the amount of data, we envision as a consequence an increase in research in the AI-powered model repair.

In this direction, it would be interesting to create contests, challenges, and benchmarks like the ones already existing in the MDE community (e.g., Transformation Tool Contest [109], The MULTI Process Challenge [110] and benchmarks for OCL [70] and bidirectional transformations [111]) focusing on, for example, the creation of new repositories and benchmarks, labeling new or already existing data, or testing different repair tools with the same benchmark.

Regarding the rest of the challenges, there is a need for creating common protocols, standards, and guidelines for how future model repair tools and the results they produce will be. Hence, open discussions and debates should be encouraged. In this direction, initiatives like the Winter Modelling Meeting [112] present a great setting for researchers to reflect and debate about how to build the future of this field.

The identified challenges are not isolated from each other, and dealing with some of them may have an important impact on others. For example, the generality and variety of algorithms challenges, presented in Sects. 4.4 and 4.7 depend heavily on how the data challenge in Sect. 4.1 is addressed (see ① in Fig. 8). More general solutions will be provided once more data is available, and likewise, more algorithms will be feasible to apply. As already stated in this paper, it is not possible to apply most supervised and unsupervised learning algorithms without enough data. Another example is how data bias, as presented in Sect. 4.5 will affect the quality of future data repositories (see ② in Fig. 8). Bias will need to be minimized so that future repositories in the field keep a desirable quality.

In order to count with more datasets and repositories, addressing the data challenge from Sect. 4.1, it is important that researchers in the community keep a sharing attitude and publish their experimentation data. By doing so, the more research is produced, the more repositories can be expected to become available. In this direction, creating infrastructures, as stated in Sect. 4.2, that support modelers in sharing and acquiring data might help in augmenting the number of datasets available (see ③ in Fig. 8).

Additionally, it will also be important to share AI-powered model repair tools, so that researchers can compare them. It is crucial to assure the availability of these tools over time, not only by keeping them online but by documenting their use and installation process and periodically checking that it is still up-to-date. Sharing would also affect future model repair infrastructures. For example, experimenting infrastructures or benchmarking will not be possible without the community sharing their research results.

*Opportunities* Finally, in Sect. 5 we enumerated research opportunities for AI-powered model repair based on the successful usage of AI in comparable fields, such as code repair. Here we identified nine categories: prediction, inspiration from code repair, model completion, explainable model repair, natural language processing for model repair, model classification, transfer model repair, automated ML for model repair, and synthetic modeling data. We identified these categories by grouping existing approaches in the literature.

The similarities of the approaches presented in Sect. 5 with what can be done within model repair are promising, and an indicator of the potential development of the field in the near future. However, for some of these opportunities to become feasible, it is needed again to first overcome the challenge about data presented in Sect. 4.1. From the categories identified, five of them (prediction, inspiration from code repair, model completion, model classification, and synthetic modeling data) require more data available and of greater curation and quality before more research can be conducted (see ④ in Fig. 8), because, as mentioned above, most supervised and unsupervised algorithms require more data in order to work.

Explainable model repair, natural language processing for model repair, transfer model repair, and automated ML for model repair are more independent from this challenge since the opportunities in these categories are more about applying existing ML techniques to model repair-related problems. However, without enough data, the opportunities grouped under explainable model repair, transfer model repair, and automated ML for model repair would be reduced, since they would not be feasible for supervised and unsupervised learning algorithms, hence reducing their research scope (see ⑤ in Fig. 8).

In this direction, the synthetic modeling data opportunity presented in Sect. 5.9 could help to overcome several aspects of the data challenge (see ⑥ in Fig. 8). For example, by using data augmentation, big datasets could be created from a few initial samples, making it easier to obtain larger and more diverse data. Likewise, using automatic label generation could reduce the burden and time of manually labeling data. By combining these techniques: data augmentation and label generation, the community could use already existing datasets and increase their sizes and diversity, and, by labeling a portion of them, the newly augmented data could be labeled automatically, reducing the burden of creating new data.

The identified opportunities can be combined, leading to new ones, similar to the presented approach applying NLP for model completion [86]. Due to the possibilities NLP offers, it could be combined with other opportunities, such as inspiration from code repair, completion, classification, or automated ML for model repair, since, by using natural language, modelers could easily specify the repair they want to perform in the models, consistency rules, preferences, requisites, etc. (see ⑦ in Fig. 8). Additionally, language analysis could improve the quality of the repair actions by, for example, providing comprehensive names to automatically created or renamed elements.

Regarding the automated ML for model repair opportunity presented in Sect. 5.8, it will be important to keep in mind how it is related to the level of automation challenge presented in Sect. 4.6. Automation can be a great tool to democratize the use of AI; however, it is important that modelers remain in control of how their tools work and are configured (see ⑧ in Fig. 8).

As stated in the standardization challenge in Sect. 4.3, some experiments might provide different results in different executions, which can threaten the validity of the research. By using XAI, the technology introduced in the explainable model repair opportunity in Sect. 5.4, experiments could include the reasons for the produced results, and if they change in different executions a more insightful analysis could be provided, addressing their causes and potentially solving this situation (see ⑨ in Fig. 8).

For the transfer model repair opportunity presented in Sect. 5.7 to become feasible, the community will need to have an open attitude toward sharing their research results not only by publishing papers about them, but by making an effort in offering the hyperparameter configuration of their ML architectures, trained architectures, and training weights publicly so that other researchers can make use of them. Finally, using TL might ease the generality challenge introduced in Sect. 4.4 by providing algorithms with the ability to forward previous knowledge to adapt to new solutions (see ⑩ in Fig. 8).

## 8 Threats to validity

In this section, we discuss potential threats that are associated with the validity of our research. The main threat of our research is the criteria followed to selecting the papers cited in Sects. 3, 4, and 5. The scope of the search was restricted to sources which mainly came from our experience in the field when developing our own AI-powered model repair approach. It might be the possibility that we have missed some work done in the area of model repair and AI, and we do not claim to have included all. However, we have been active in the AI-powered model repair community from the start, which has given us a deep insight into the field and its needs. This paper is based on our research and experience from participating in the most significant conferences and workshops in the field, such as the European Conference on Modelling Foundations and Applications (ECMFA), the MODELS Conference, the Workshop on Artificial Intelligence and Model-driven Engineering (MDE Intelligence), the Workshop on Analytics and Mining of Model Repositories (AMMoRe).

Likewise, other challenges and opportunities could have been identified from the selected papers. However, we consider our scope wide enough to have covered the main aspects of the AI-powered model repair field, and we find that our expertise in the area mitigates the threat of missing important challenges and opportunities.

Furthermore, we could have used other classifications of AI approaches, which could change the perspective of the paper and the included research. As stated in Sect. 2.2, we follow a simplified classification to avoid adding unnecessary complexity to the text. We consider this high-level classification provides a good enough understanding of the AI field for the reader, and allows us to introduce a wide range of AI techniques.

## 9 Conclusions and future work

In this paper, we have presented an experience report of the AI-powered model repair field. We have explored existing approaches in the AI-powered model repair field and analyzed and discussed the research challenges and opportunities for the development of the field that we have identified in the literature. Also, we have discussed how current approaches, opportunities, and challenges affect each other, providing a general picture of the AI-powered model repair field. Additionally, we have discussed the lessons we have learned while developing PARMOREL, our AI-powered approach for model repair.

AI is a disruptive technology that can improve how we face the model repair process. There is a lot of potential yet to explore, but for doing so, there are a series of challenges the modeling community needs to address. In concrete, challenges regarding data should be prioritized, since without data we will not be able to take full advantage of the AI potential. The AI-powered model repair field is still at an initial stage and hence, it is in our hands, as researchers, to shape how its future will be.

The purpose of this paper is to serve as a starting point for researchers when studying the AI-powered model repair field or developing new AI-powered model repair approaches. In addition, the identified opportunities and challenges should serve as an inspiration for future research projects.

In future work, we will focus on the data and infrastructure challenges presented in Sect. 4. We will collect and curate diverse modeling data, and additionally, we will research and apply techniques like data augmentation and automatic label generation. When having enough data, we plan to create a benchmark to help researchers to compare their own datasets and tools. Then, by using this benchmark, we plan to perform a comparative study between existing AI-powered model repair tools. Additionally, we would like to compare AI-powered and non-AI-powered tools. We are aware that, due to their tailor-made nature, some tools are currently difficult to compare with others, hence, we will also focus on finding objective evaluation metrics to compare these tools, overcoming their lack of generality.

**Funding** Open access funding provided by Western Norway University Of Applied Sciences

## References

1. Bettini, L., Di Ruscio, D., Iovino, L., Pierantonio, A.: Quality-driven detection and resolution of metamodel smells. IEEE Access **7**, 16364–16376 (2019). https://doi.org/10.1109/ACCESS.2019.2891357
2. Strittmatter, M., Hinkel, G., Langhammer, M., Jung, R., Heinrich, R.: Challenges in the evolution of metamodels: Smells and anti-patterns of a historically-grown metamodel. In: Conference: 10th International Workshop on Models and Evolution (ME) (2016)
3. Feldmann, S., Kernschmidt, K., Wimmer, M., Vogel-Heuser, B.: Managing inter-model inconsistencies in model-based systems engineering: application in automated production systems engineering. J. Syst. Softw. **153**, 105–134 (2019). https://doi.org/10.1016/j.jss.2019.03.060
4. Taentzer, G., Ohrndorf, M., Lamo, Y., Rutle, A.: Change-preserving model repair. In: International Conference on Fundamental Approaches to Software Engineering, pp. 283–299. Springer (2017). https://doi.org/10.1007/978-3-662-54494-5-16
5. Ohrndorf, M., Pietsch, C., Kelter, U., Kehrer, T.: Revision: a tool for history-based model repair recommendations. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, pp. 105–108. ACM (2018)
6. Nassar, N., Radke, H., Arendt, T.: Rule-based repair of EMF models: An automated interactive approach. In: International Conference on Theory and Practice of Model Transformations, pp. 171–181. Springer (2017)
7. Macedo, N., Guimaraes, T., Cunha, A.: Model repair and transformation with echo. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, pp. 694–697. IEEE Press (2013)
8. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying model-driven software engineering. In: Federation of International Conferences on Software Technologies: Applications and Foundations, pp. 154–160. Springer (2017)
9. Shafiq, S., Mashkoor, A., Mayr-Dorn, C., Egyed, A.: Machine learning for software engineering: A systematic mapping. arXiv preprint arXiv:2005.13299 (2020)
10. Chang, R., Sankaranarayanan, S., Jiang, G., Ivancic, F.: Software testing using machine learning (2014). US Patent 8,924,938
11. Chandra, K., Kapoor, G., Kohli, R., Gupta, A.: Improving software quality using machine learning. In: Innovation and Challenges in Cyber Security (ICICCS-INBUSH), 2016 International Conference on, pp. 115–118. IEEE (2016)
12. Malhotra, R.: A systematic review of machine learning techniques for software fault prediction. Appl. Soft Comput. **27**, 504–518 (2015)
13. Friedrichs, O., Huger, A., O'donnell, A.J.: Method and apparatus for detecting malicious software through contextual convictions, generic signatures and machine learning techniques (2015). US Patent 9,088,601
14. Models 2021 Conference Homepage. http://www.modelsconference.org/. Last accessed on 24/03/2021
15. MDE Intelligence Workshop. https://mde-intelligence.github.io/. Last accessed on 24/03/2021
16. Bucchiarone, A., Cabot, J., Paige, R.F., Pierantonio, A.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**(1), 5–13 (2020)
17. Torre, D., Labiche, Y., Genero, M., Elaasar, M.: A systematic identification of consistency rules for uml diagrams. J. Syst Softw. **144**, 121–142 (2018)
18. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R., et al.: Handbook of Model Checking, vol. 10. Springer, Berlin (2018)
19. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C., Smolka, S.A.: Model repair for probabilistic systems. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 326–340. Springer (2011)
20. Finkelstein, A.: A foolish consistency: Technical challenges in consistency management. In: International Conference on Database and Expert Systems Applications, pp. 1–5. Springer (2000)
21. Torres, W., Van den Brand, M.G., Serebrenik, A.: A systematic literature review of cross-domain model consistency checking by

model management tools. Softw. Syst. Model. **20**(3), 897–916 (2021)
22. Macedo, N., Jorge, T., Cunha, A.: A feature-based classification of model repair approaches. IEEE Trans. Softw. Eng. **43**(7), 615–640 (2016). https://doi.org/10.1109/TSE.2016.2620145
23. Ohrndorf, M., Pietsch, C., Kelter, U., Grunske, L., Kehrer, T.: History-based model repair recommendations. ACM Trans. Softw. Eng. Methodol. (TOSEM) **30**(2), 1–46 (2021)
24. Cervantes, A.A., van Beest, N.R., La Rosa, M., Dumas, M., García-Bañuelos, L.: Interactive and incremental business process model repair. In: OTM Confederated International Conferences"On the Move to Meaningful Internet Systems", pp. 53–74. Springer (2017)
25. Barriga, A., Heldal, R., Iovino, L., Marthinsen, M., Rutle, A.: An extensible framework for customizable model repair. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 24–34 (2020)
26. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press, Cambridge (2014)
27. Mohri, M., Rostamizadeh, A., Talwalkar, A.: Foundations of Machine Learning. MIT Press, Amsterdam (2018)
28. Cady, F.: Machine learning classification. In: The Data Science Handbook, pp. 97–120. Wiley, New York (2017)
29. Alpaydin, E.: Introduction to Machine Learning. Adaptive Computation and Machine Learning, 3rd edn. MIT Press, Cambridge (2014)
30. Thrun, S., Littman, M.L.: Reinforcement learning: an introduction. AI Magazine **21**(1), 103–103 (2000)
31. Christiano, P., Leike, J., Brown, T.B., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences. arXiv preprint arXiv:1706.03741 (2017)
32. Mens, T., Van Der Straeten, R., D'Hondt, M.: Detecting and resolving model inconsistencies using transformation dependency analysis. In: International Conference on Model Driven Engineering Languages and Systems, pp. 200–214. Springer (2006)
33. Amelunxen, C., Legros, E., Schürr, A., Stürmer, I.: Checking and enforcement of modeling guidelines with graph transformations. In: International Symposium on Applications of Graph Transformations with Industrial Relevance, pp. 313–328. Springer (2007)
34. Mantz, F., Taentzer, G., Lamo, Y., Wolter, U.: Co-evolving meta-models and their instance models: a formal approach based on graph transformation. Sci. Computer Program. **104**, 2–43 (2015)
35. Egyed, A., Letier, E., Finkelstein, A.: Generating and evaluating choices for fixing inconsistencies in uml design models. In: 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, pp. 99–108. IEEE (2008)
36. Kretschmer, R., Khelladi, D.E., Egyed, A.: An automated and instant discovery of concrete repairs for model inconsistencies. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, pp. 298–299. ACM (2018)
37. Khelladi, D.E., Kretschmer, R., Egyed, A.: Detecting and exploring side effects when repairing model inconsistencies. In: Proceedings of the 12th ACM SIGPLAN International Conference on Software Language Engineering, pp. 113–126 (2019)
38. Reder, A., Egyed, A.: Computing repair trees for resolving inconsistencies in design models. In: Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp. 220–229 (2012)
39. Leonetti, M., Iocchi, L., Stone, P.: A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. Artif. Intell. **241**, 103–130 (2016)
40. Puissant, J.P., Van Der Straeten, R., Mens, T.: Resolving model inconsistencies using automated regression planning. Softw. Syst. Model. **14**(1), 461–481 (2015)
41. Puissant, J.P.: Resolving inconsistencies in model-driven engineering using automated planning. In: Seminar on Advanced Tools & Techniques for Software Evolution (SATToSE), Koblenz, Germany (2012)
42. Barriga, A., Bettini, L., Iovino, L., Rutle, A., Heldal, R.: Addressing the trade off between smells and quality when refactoring class diagrams. J. Object Technol. **20**(3), 1:1–15 (2021). https://doi.org/10.5381/jot.2021.20.3.a1.The 17th European Conference on Modelling Foundations and Applications (ECMFA 2021). URL http://www.jot.fm/contents/issue_2021_03/article1.html
43. Barriga, A., Rutle, A., Heldal, R.: Personalized and automatic model repairing using reinforcement learning. In: 22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019, pp. 175–181 (2019). https://doi.org/10.1109/MODELS-C.2019.00030.
44. Barriga, A., Rutle, A., Rogardt, H.: Improving model repair through experience sharing. J. Object Technol. **19**(1), 897–916 (2020)
45. Iovino, L., Barriga, A., Rutle, A., Rogardt, H.: Model repair with quality-based reinforcement learning. J. Object Technol. **19**(2), 17:1-17:21 (2020). https://doi.org/10.5381/jot.2020.19.2.a17
46. Burgueño, L., Cabot, J., Gérard, S.: An lstm-based neural network architecture for model transformations. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 294–299. IEEE (2019)
47. Sidhu, B.K., Singh, K., Sharma, N.: A machine learning approach to software model refactoring. Int. J. Computers Appl. pp. 1–12 (2020)
48. Uml-ninja. http://models-db.com/. Last accessed on 24/03/2021
49. Wust, J.: Sdmetrics: The software design metrics tool for uml (2005)
50. Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. J. Artif. Intell. Res. **11**, 241–276 (1999)
51. Ghannem, A., El Boussaidi, G., Kessentini, M.: Model refactoring using interactive genetic algorithm. In: International Symposium on Search Based Software Engineering, pp. 96–110. Springer (2013)
52. Seal-Ucla: Seal-ucla/ref-finder. https://github.com/SEAL-UCLA/Ref-Finder. Last accessed on 24/03/2021
53. Alrajeh, D., Kramer, J., Russo, A., Uchitel, S.: Automated support for diagnosis and repair. Commun. ACM **58**(2), 65–72 (2015)
54. Jackson, D.: Software Abstractions: Logic, Language and Analysis. MIT Press, Amsterdam (2012)
55. Fumagalli, M., Sales, T.P., Guizzardi, G.: Towards automated support for conceptual model diagnosis and repair. In: International Conference on Conceptual Modeling, pp. 15–25. Springer (2020)
56. Narain, S., et al.: Network configuration management via model finding. In: LISA, vol. 5, pp. 15–15 (2005)
57. Fumagalli, M., Sales, T.P., Guizzardi, G.: Mind the gap!: Learning missing constraints from annotated conceptual model simulations. In: IFIP Working Conference on The Practice of Enterprise Modeling, pp. 64–79. Springer (2021)
58. Bucchiarone, A., Ciccozzi, F., Lambers, L., Pierantonio, A., Tichy, M., Tisi, M., Wortmann, A., Zaytsev, V.: What is the future of modeling? IEEE Softw. **38**(2), 119–127 (2021)
59. Mussbacher, G., Amyot, D., Breu, R., Bruel, J.M., Cheng, B.H., Collet, P., Combemale, B., France, R.B., Heldal, R., Hill, J., et al.: The relevance of model-driven engineering thirty years from now. In: International Conference on Model Driven Engineering Languages and Systems, pp. 183–200. Springer (2014)
60. France, R.B., Bieman, J.M., Mandalaparty, S.P., Cheng, B.H., Jensen, A.: Repository for model driven development (remodd). In: 2012 34th International Conference on Software Engineering (ICSE), pp. 1471–1472. IEEE (2012)

61. Allilaire, F.: Atl transformations. https://www.eclipse.org/atl/atlTransformations/. Last accessed on 24/03/2021
62. Basciani, F., Di Rocco, J., Di Ruscio, D., Di Salle, A., Iovino, L., Pierantonio, A.: Mdeforge: an extensible web-based modeling platform. In: CloudMDE@ MoDELS, pp. 66–75 (2014)
63. Dirix, M., Muller, A., Aranega, V.: Genmymodel: an online uml case tool (2013)
64. Karasneh, B., Chaudron, M.R.: Online img2uml repository: An online repository for UML. In: EESSMOD@ MoDELS, pp. 61–66 (2013)
65. Barriga, A., Di Ruscio, D., Iovino, L., Nguyen, P.T., Pierantonio, A.: An extensible tool-chain for analyzing datasets of metamodels. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1–8 (2020)
66. Gogolla, M., Cabot, J.: Continuing a benchmark for uml and ocl design and analysis tools. In: Federation of International Conferences on Software Technologies: Applications and Foundations, pp. 289–302. Springer (2016)
67. Nguyen, P.T., Di Ruscio, D., Pierantonio, A., Di Rocco, J., Iovino, L.: Convolutional neural networks for enhanced classification mechanisms of metamodels. J. Syst. Softw. **172**, 110,860 (2021)
68. Burdusel, A., Zschaler, S.: Towards scalable search-based model engineering with mdeoptimiser scale. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 189–195. IEEE (2019)
69. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: Future of Software Engineering (FOSE'07), pp. 37–54. IEEE (2007)
70. Gogolla, M., Büttner, F., Cabot, J.: Initiating a benchmark for uml and ocl analysis tools. In: International Conference on Tests and Proofs, pp. 115–132. Springer (2013)
71. Bertoa, M.F., Burgueño, L., Moreno, N., Vallecillo, A.: Incorporating measurement uncertainty into ocl/uml primitive datatypes. Softw. Syst. Model. **19**(5), 1163–1189 (2020)
72. Mussbacher, G., Combemale, B., Kienzle, J., Abrahão, S., Ali, H., Bencomo, N., Búr, M., Burgueño, L., Engels, G., Jeanjean, P., et al.: Opportunities in intelligent modeling assistance. Softw. Syst. Model. **19**(5), 1045–1053 (2020)
73. Kaplan, A., Haenlein, M.: Rulers of the world, unite! the challenges and opportunities of artificial intelligence. Business Horizons **63**(1), 37–50 (2020)
74. Amershi, S., Cakmak, M., Knox, W.B., Kulesza, T.: Power to the people: the role of humans in interactive machine learning. Ai Magazine **35**(4), 105–120 (2014)
75. Fontana, F.A., Mäntylä, M.V., Zanoni, M., Marino, A.: Comparing and experimenting machine learning techniques for code smell detection. Empirical Softw. Eng. **21**(3), 1143–1191 (2016)
76. Alenezi, M., Akour, M., Al Qasem, O.: Harnessing deep learning algorithms to predict software refactoring. Telkomnika **18**(6), 154–160 (2020)
77. Sheneamer, A.M.: An automatic advisor for refactoring software clones based on machine learning. IEEE Access **8**, 978–988 (2020)
78. Mesbah, A., Rice, A., Johnston, E., Glorioso, N., Aftandilian, E.: Deepdelta: learning to repair compilation errors. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 925–936 (2019)
79. White, M., Tufano, M., Martinez, M., Monperrus, M., Poshyvanyk, D.: Sorting and transforming program repair ingredients via deep learning code similarities. In: 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 479–490. IEEE (2019)
80. Harer, J., Ozdemir, O., Lazovich, T., Reale, C.P., Russell, R.L., Kim, L.Y., Chin, P.: Learning to repair software vulnerabilities with generative adversarial networks. arXiv preprint arXiv:1805.07475 (2018)
81. Moghadam, I.H., Ó Cinnéide, M.: Code-imp: a tool for automated search-based refactoring. In: Proceedings of the 4th Workshop on Refactoring Tools, pp. 41–44 (2011)
82. Selman, B., Gomes, C.P.: Hill-climbing search. Encyclopedia of cognitive science (2006)
83. Svyatkovskiy, A., Zhao, Y., Fu, S., Sundaresan, N.: Pythia: Ai-assisted code completion system. In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 2727–2735 (2019)
84. Li, J., Wang, Y., Lyu, M.R., King, I.: Code completion with neural attention and pointer networks. arXiv preprint arXiv:1711.09573 (2017)
85. Proksch, S., Lerch, J., Mezini, M.: Intelligent code completion with Bayesian networks. ACM Trans. Softw. Eng. Methodol. (TOSEM) **25**(1), 1–31 (2015)
86. Burgueño, L., Clarisó, R., Li, S., Gérard, S., Cabot, J.: A nlp-based architecture for the autocompletion of partial domain models (2020)
87. Gunning, D.: Explainable artificial intelligence (XAI). Defense Advanced Research Projects Agency (DARPA), nd Web (2017)
88. Monperrus, M.: Explainable software bot contributions: Case study of automated bug fixes. In: 2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE), pp. 12–15. IEEE (2019)
89. Joshi, S., Deshpande, D.: Textual requirement analysis for uml diagram extraction by using nlp. Int. J. Computer Appl. **50**(8), 42–46 (2012)
90. Deeptimahanti, D.K., Babar, M.A.: An automated tool for generating uml models from natural language requirements. In: 2009 IEEE/ACM International Conference on Automated Software Engineering, pp. 680–682. IEEE (2009)
91. Lano, K., Fang, S., Umar, M., Yassipour-Tehrani, S.: Enhancing model transformation synthesis using natural language processing. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1–10 (2020)
92. Elallaoui, M., Nafil, K., Touahni, R.: Automatic transformation of user stories into uml use case diagrams using nlp techniques. Proc. Computer Sci. **130**, 42–49 (2018)
93. Sajjad, R., Sarwar, N.: Nlp based verification of a uml class model. In: 2016 Sixth International Conference on Innovative Computing Technology (INTECH), pp. 30–35. IEEE (2016)
94. Weyssow, M., Sahraoui, H., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. arXiv preprint arXiv:2104.01642 (2021)
95. Silva, R.F., Roy, C.K., Rahman, M.M., Schneider, K.A., Paixao, K., de Almeida Maia, M.: Recommending comprehensive solutions for programming tasks by mining crowd knowledge. In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pp. 358–368. IEEE (2019)
96. Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al.: Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771 (2019)
97. Ren, R., Castro, J.W., Santos, A., Pérez-Soler, S., Acuña, S.T., de Lara, J.: Collaborative modelling: chatbots or on-line tools? an experimental study. In: Proceedings of the Evaluation and Assessment in Software Engineering, pp. 260–269 (2020)
98. Pérez-Soler, S., Guerra, E., de Lara, J.: Flexible modelling using conversational agents. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 478–482. IEEE (2019)

99. Pérez-Soler, S., Daniel, G., Cabot, J., Guerra, E., de Lara, J.: Towards automating the synthesis of chatbots for conversational model query. In: Enterprise, Business-Process and Information Systems Modeling, pp. 257–265. Springer (2020)
100. Martínez, S., Wimmer, M., Cabot, J.: Efficient plagiarism detection for software modeling assignments. Computer Sci. Edu. **30**(2), 187–215 (2020)
101. Nguyen, P.T., Di Rocco, J., Di Ruscio, D., Pierantonio, A., Iovino, L.: Automated classification of metamodel repositories: A machine learning approach. In: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 272–282. IEEE (2019)
102. Torrey, L., Shavlik, J.: Transfer learning. In: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, pp. 242–264. IGI Global (2010)
103. Hutter, F., Kotthoff, L., Vanschoren, J.: Automated Machine Learning: Methods, Systems. Challenges. Springer, Berlin (2019)
104. Mougenot, A., Darrasse, A., Blanc, X., Soria, M.: Uniform random generation of huge metamodel instances. In: European Conference on Model Driven Architecture-Foundations and Applications, pp. 130–145. Springer (2009)
105. Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Seidl, M., Schwinger, W., Wimmer, M.: Amor–towards adaptable model versioning. In: 1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS, vol. 8, pp. 4–50 (2008)
106. Anicet Zanini, R., Luna Colombini, E.: Parkinson's disease emg data augmentation and simulation with dcgans and style transfer. Sensors **20**(9), 2605 (2020)
107. Hoermann, S., Henzler, P., Bach, M., Dietmayer, K.: Object detection on dynamic occupancy grid maps using deep learning and automatic label generation. In: 2018 IEEE Intelligent Vehicles Symposium (IV), pp. 826–833. IEEE (2018)
108. Ammore@models'18 (2019). https://modelanalytics.wordpress.com/ammore18/. Last accessed on 24/03/2021
109. Transformation Tool Contest (TTC) 2021. https://www.transformation-tool-contest.eu/. Last accessed on 24/03/2021
110. 7th International Workshop on Multi-level Modelling. https://www.wi-inf.uni-duisburg-essen.de/MULTI2020/challenge. Last accessed on 24/03/2021
111. Anjorin, A., Buchmann, T., Westfechtel, B., Diskin, Z., Ko, H.S., Eramo, R., Hinkel, G., Samimi-Dehkordi, L., Zündorf, A.: Benchmarking bidirectional transformations: theory, implementation, application, and assessment. Softw. Syst. Model. **19**, 1–45 (2019)
112. Winter Modelling Meeting (2020). http://eventmall.info/WMM2020/. Last accessed on 24/03/2021

**Angela Barriga** holds Ph.D. in Computer Science from the Western Norway University of Applied Sciences. She has experience working with machine learning, computer vision, gerontechnology, and pervasive systems. Barriga's thesis was focused on model repair, specially on repairing using reinforcement learning. She has been part of the local organization of iFM 2019 and was involved in STAF 2020-2021. She has also been part of the program committee of the third international workshop on gerontechnology. You can learn more about her at https://angelabr.github.io/ or contact her at abar@hvl.no.

**Adrian Rutle** is a professor at Western Norway University of Applied Sciences. Adrian holds Ph.D. in Computer Science from the University of Bergen, Norway. Rutle is a professor at the Department of Computer Science, Electrical Engineering and Mathematical Sciences at the Western Norway University of Applied Sciences, Bergen. Rutle's main interest is applying theoretical results from the field of model-driven software engineering to practical domains and has expertise in the development of modeling frameworks and domain-specific modeling languages. He also conducts research in the fields of modeling and simulation for robotics, eHealth, digital fabrication, smart systems, and machine learning. Contact him at adrian.rutle@hvl.no.

**Rogardt Heldal** is a professor of Software Engineering at the Western Norway University of Applied Sciences. Heldal holds an honors degree in Computer Science from Glasgow University, Scotland, and a Ph.D. in Computer Science from Chalmers University of Technology, Sweden. His research interests include requirements engineering, software processes, software modeling, software architecture, cyber-physical systems, machine learning, and empirical research. Many of his research projects are performed in collaboration with industry. Contact him at rogardt.heldal@hvl.no.