



# A Model Cleansing Pipeline for Model-Driven Engineering: Mitigating the Garbage In, Garbage Out Problem for Open Model Repositories

Anđela Đelić

TU Wien, Business Informatics Group  
andjela.djelic@tuwien.ac.at

Syed Juned Ali

TU Wien, Business Informatics Group  
syed.juned.ali@tuwien.ac.at

Charlotte Verbruggen

TU Wien, Business Informatics Group  
charlotte.verbruggen@tuwien.ac.at

Julia Neidhardt

TU Wien, CDL RecSys  
julia.neidhardt@tuwien.ac.at

Dominik Bork

TU Wien, Business Informatics Group  
dominik.bork@tuwien.ac.at

**Abstract**—In data-driven research within Model-Driven Engineering (MDE), the extraction of conceptual models, such as UML diagrams, from software repositories is a crucial step for analyzing software design, evolution, and quality. However, these extracted models often contain inconsistencies, redundancies, and noise because most model repositories are not curated. Without effective data cleansing, the reliability of empirical and machine learning (ML)-based MDE studies working with these repositories is seriously threatened. This paper proposes a data cleansing pipeline designed to effectively cleanse model repositories. Our approach systematically addresses common data quality issues by offering a sequence of automated pre-processing, validation, and filtering steps based on rule-based heuristics and ML techniques. By integrating conceptual modeling-specific data cleansing techniques into an automated pipeline, our approach reduces manual intervention, enhances reproducibility, and supports scalable analysis of model repositories. In an experimental evaluation of open-source UML diagram repositories, we demonstrate the effectiveness of our method in cleansing models. In two reproducibility studies, we further show the statistically significant effect the use of our MCP4CM pipeline has on downstream tasks.

**Index Terms**—Model-driven engineering, Model repositories, Open models, Machine learning, Data cleansing, UML.

## I. INTRODUCTION

Model-Driven Engineering (MDE) has established itself as a fundamental paradigm for managing the complexity of software systems by leveraging high-level abstractions through models [1]. There exists a growing interest in applying data-driven and machine learning (ML) techniques to MDE, aiming to (semi-)automate modeling tasks such as model transformation, repository management and clone detection [2]. ML for MDE (ML4MDE) publications have tripled in the last five years [3]. Recently, advanced AI-based approaches such as Deep Learning (DL) and Natural Language Processing (NLP) have been applied for conceptual modeling to support Partial Model Completion (PMC) [4]–[6], automated domain model extraction [7], [8], model transformation [9], [10], metamodel

classification and clustering (MCC) [11], [12]. A recent survey [13] indicated the emerging role of AI in modeling. However, realizing such applications critically depends on the availability of high-quality datasets of conceptual models that accurately reflect real-world software design.

In contrast to traditional ML domains where structured and curated datasets are readily available (e.g., ImageNet [14], UCI ML Repository [15]), the modeling community lacks large-scale, high-quality model datasets suitable for empirical and ML-based studies. Several efforts have emerged to mine software modeling artifacts from online repositories such as GitHub, GenMyModel [16]–[19], or other model-sharing platforms. While these repositories can potentially serve as a rich source of models for MDE research, they are predominantly uncurated, noisy, and inconsistent, and frequently contain models that are syntactically valid artifacts yet semantically meaningless, incomplete, or redundant.

Poor model quality poses a major challenge, as ML techniques are highly sensitive to input data. Using models extracted from public repositories ‘as is’ without adequate cleansing and validation makes the results of any subsequent data-driven downstream tasks unreliable. This undermines both the validity and the reproducibility of ML-based MDE research. Surprisingly, despite the criticality of this issue, little attention has been paid to this problem, particularly in the context of preparing such data for ML applications.

As the modeling community increasingly embraces data-driven approaches, the need for robust and scalable model cleansing techniques becomes ever more pressing. We propose an automated Model Cleansing Pipeline for Conceptual Models (MCP4CM) to empirically evaluate this claim, combining general and modeling language-specific heuristics to ensure the cleansed model datasets are semantically meaningful. We reproduce two ML-based MDE studies to show the statistically significant effect of using MCP4CM on downstream tasks such as model domain classification. This paper addresses the following research question: *What is the effect of model*

*cleansing on downstream ML4MDE tasks?*

In the remainder of this paper, Section II discusses relevant background and related works. Section III then introduces our novel model cleansing pipeline. In Section IV, we evaluate the necessity and performance of our pipeline using reproducibility studies, where we show the effect of model cleansing on downstream ML4MDE tasks and respond to our research question. A comprehensive discussion of key findings and implications of our work is presented in Section V before we conclude this paper in Section VI.

## II. BACKGROUND AND RELATED WORKS

The subsequent section provides the background information necessary for our model cleansing pipeline and its empirical evaluation. Firstly, an overview of the current landscape of publicly available model repositories is provided, along with an outline of works that utilize these repositories for ML-based tasks in the MDE domain. We also examine prevalent data quality issues encountered in model repositories, as well as how these issues are usually addressed. We outline typical ML tasks and training strategies used in this context. Since one of our reproducibility studies involves explainable ML, the relevant explainability techniques are briefly reviewed as well (see Fig. 3).

### A. Model Repositories and Usage

Numerous open model repositories have been proposed in various modeling languages such as ArchiMate [20] (977 models), Petri Nets [21] (664 models), BPMN [22] (174 models) [23] (25,866 models) [24] (25,590 models), Ecore [25] (555 models), OntoUML [26] (185 models) and UML [18] (10,586 models) [17] (93,000 models). From these, only [18], [23] and [25] provide a labeled model dataset where each model is tagged with relevant tags, e.g. for the domain.

### B. Related Works

Several works have used model datasets to train ML models for modeling tasks. Ali et al. [6] use the OntoUML dataset for training ML models to predict ontological stereotypes. Lopez et al. [12], [27] have used UML and Ecore model datasets [18] for modeling tasks such as domain classification, model clustering, and model completion. Alcaide et al. [28] have also used the ModelSet dataset to apply explainability techniques to interpret the predictions made by a trained ML model for the tasks of dummy detection, domain classification, and multi-label prediction. The following related works have proposed dummy detection and duplicate removal (or clone detection) techniques for data cleansing.

1) *Dummy Filtering*: A problem often encountered in model repositories is the presence of numerous non-informative models, e.g., trivial or placeholder diagrams, and example models auto-generated by modeling tools. These ‘toy’ or ‘dummy’ models usually represent noise for ML models and can affect their quality. In most ML4MDE applications (e.g. AI-assisted modeling environments), they are not meaningful data that ML models should learn from. However, in some

applications (e.g., dummy detection), the user might decide to keep the dummy models in the model dataset. The threat of training an ML model on low-quality data resulting in the ML model learning non-meaningful patterns motivates researchers to find ways to flag these trivial models. One way to do this is by defining heuristics that detect noisy data by focusing on the models’ content. This has been done for BPMN models in [23]. Alternatively, ML models for binary classification can be trained to detect dummy models using a labeled model dataset as training data, as demonstrated in [28]. However, an ML-driven dummy detection approach faces several challenges. It requires a labeled model dataset, which leads to a chicken-and-egg problem whereby a clean model dataset is needed to train an ML model to clean a model dataset, and the trained ML model may not generalize well to new models and overfit for a specific set of models.

2) *Duplicate Filtering*: Allamanis [29] reported that code repositories from sites such as GitHub contain substantial amounts of duplicated code which can accidentally split between training and test set, leading to inflated ML model performance. Problems related to code clones, such as quality degradation and maintenance issues, have been noted in the MDE domain as well [30]. Nikoo et al. [31] and Babur et al. [32] propose four types of clones, characterized by varying degrees of similarity. Model clone detection refers to the process of identifying these duplicate or highly similar models (or model fragments). In the context of ML-based MDE methods, (near) duplicate samples in the model dataset can lead to data leakage and overly optimistic results. Clone detection has been explored in various modeling languages like UML [30], BPMN [31], Simulink [33], and metamodels [32]. Proposed techniques for model clone detection often fall into one of the following categories: structural-based approaches, lexical-based approaches, or a combination of the two.

3) *Language Filtering*: A model dataset can have models created with element names in different languages. Non-English element names can be considered noise for text-based ML models, especially those trained largely on English text. Therefore, filtering a model dataset based on language can be a meaningful step. ModelSet [18] provides a language tag, allowing users to filter out all non-English models. However, since most other model datasets do not have a language label, automatic language detection is needed, as is done in [23].

### C. ML Tasks and Training

**Model Encoding.** Models need to be transformed into suitable encodings for ML. These can include vector representations (e.g., frequency counts or embeddings) based on model elements such as names, types, and attributes. Common encoding techniques include *i*) bag-of-words that creates vectors of the size of the vocabulary with all the indices of the terms present in the model, *ii*) term frequency-inverse document frequency (TF-IDF) matrix, or *iii*) word embeddings, which is a representation of text where each word has an associated vector and words with similar meanings have similar vectors [34].

**ML Model Training.** Model encodings are used to train ML models for task-specific predictions. For dummy detection, the ML model predicts whether the encoding corresponds to a dummy model. For domain classification, it predicts the correct domain class. For multi-label classification, it predicts a probability matrix indicating the belonging of each label to a model. Common classifiers are Feed-Forward Neural Networks (FFNN), Support Vector Machines (SVM), K-nearest Neighbours (KNN), Naïve Bayes (NB), and Random Forest (RF). NB, a probabilistic classifier that assumes feature independence, includes variants such as Gaussian NB (GNB) for continuous data, Multinomial NB (MNB) for word counts, and Complement NB (CNB) for imbalanced data.

**Explainability.** Alcaide et al. [28] provide an approach to explain results based on feature importance using existing explainability approaches. *Global explainability* approaches help understand how a model makes decisions across the entire dataset. A common method is Permutation Feature Importance (PFI) [35], which measures the drop in performance when feature values are randomly shuffled. Larger performance drops indicate more important features. *Local explainability* approaches focus on individual predictions. Local Interpretable Model-agnostic Explanations (LIME) [36] creates a simple model (e.g., linear regression) to approximate how the complex model behaves for one specific prediction by slightly tweaking the input. SHapley Additive exPlanations (SHAP) [37] uses game theory to fairly distribute ‘credit’ among all input features, giving a solid but sometimes computationally expensive explanation. Breakdown [38] works similarly to SHAP but evaluates features step-by-step, making it faster but slightly more order sensitive. Lion Forests (LF) [39] extracts knowledge from Random Forests to provide local explanations.

#### D. Synopsis

While approaches for data cleansing exist, to the best of our knowledge, there is no comprehensive, extensible, and configurable automated pipeline specifically targeting the cleansing of conceptual models for the purpose of enabling ML or large-scale empirical research in MDE. Our work fills a fundamental gap by systematically identifying data quality challenges in model datasets and offering a scalable solution tailored to the unique characteristics of MDE.

### III. THE MCP4CM MODELS CLEANSING PIPELINE

In the following, we introduce MCP4CM<sup>1</sup>, our model dataset cleansing pipeline illustrated in Fig. 1. The top part of the figure shows the steps involved in the model cleansing process where the input to the pipeline is a model repository and the output is a cleansed dataset of models. To implement the entire cleansing process, MCP4CM involves three components, namely, Model Dataset Parsing, Model Dataset Loading, and Model Dataset Filtering. Below we discuss the core data classes for each component. Fig. 1 also has classes with ‘X’ prefixed. These classes underpin the extensibility support

our pipeline offers such that it can be extended with further data cleansing features and parsers for different modeling languages. Duplicate detection and language detection are dependent only on the list of names (and types) of model elements extracted from XMI files. These can be configured for any modeling language, using a proper parser for extracting element names. Heuristics for dummy detection are similarly applied to the list of names, however, they were drafted for UML specifically. They could still be applied to other modeling languages with small adjustments (e.g., different stopwords). The user can adjust heuristics’ hyperparameters or add custom heuristics. All existing thresholds are configurable by the user depending on how rigorous the filtering should be.

#### A. Model Dataset Parsing

The model parser processes a dataset of model files of a given modeling language in one of the serialization formats using the method `parse_model`. The `ModelParser` class is an abstract class that can be extended by the parsers of any modeling language. Currently, our pipeline supports a parser for UML that takes a model file in XMI format and returns an object of a `UMLModel` class. To filter the models based on the extracted data, the data is standardized by applying splitters such as camel case and snake case. This filtering splits the terms into individual lowercase terms.

#### B. Model Dataset Loading

To avoid reading and parsing the model from a file for every operation, MCP4CM stores the model in memory. We store the model and corresponding dataset in a data structure as shown in the *Model Dataset Loading* step of Fig. 1. In general, a model has five attributes, including an identifier (`id`) and name. The `names` attribute stores the names of all elements in the model, e.g., any element like a `Class` or a `Relationship` that has the `name` attribute. We store the JSON or XMI serialization of the model as well in the `Model` object. `UMLModel` specializes `Model` and adds UML-specific properties such as `diagram_type` (e.g., *Activity Diagram* and *Sequence Diagram*) and `names_with_types`, which stores the names of the elements along with their type information in the format `type:name` e.g., for a `Student` class, the name would be “class:Student”. A `ModelDataset` class is the general class for loading and storing model datasets. Our pipeline currently implements `UMLDataset` that extends `ModelDataset`. In some cases, UML datasets can be of different types, e.g., an Ecore-based metamodel that supports only class diagrams being used in `ModelSet` [40] or the complete UML metamodel that supports different diagram types.

#### C. Model Dataset Filtering

Once we have a model dataset loaded, the core step of our pipeline is model filtering. To make our pipeline extensible, we created a general class `ModelCleansingFilter`. In general, a dataset cleansing filter has a `name` and the attribute key (`model_key`) on which the filter will be applied since different filters will require different model information. For

<sup>1</sup>MCP4CM is available on Zenodo: <https://doi.org/10.5281/zenodo.15802551>

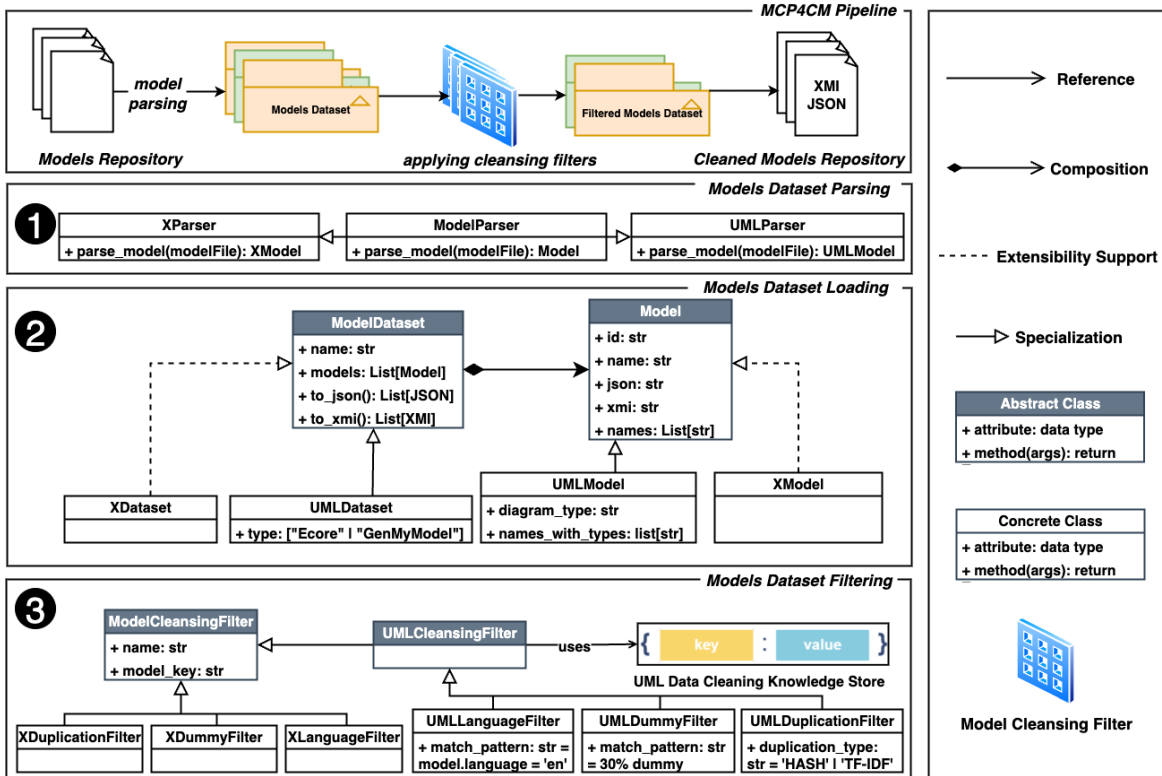


Fig. 1: MCP4CM Overview

example, the dummy removal filter can require just the names of the model elements, in which case we would only need the `model.names` attribute, whereas a filter based on the model elements' type information requires the `model_key` to be `names_with_types`. Different filters can be implemented for a given modeling language. In MCP4CM, we currently support three kinds of filters: *i) Dummy Filtering*, *ii) Duplicate Filtering*, and *iii) Language Filtering*.

1) *Dummy Filtering*: For developing the heuristics, two modeling experts collaboratively defined a set of characteristics indicative of a dummy UML model (e.g., few elements or short element names, similar to the heuristics introduced in [23]). Then, we verified and refined these characteristics by manual inspection of a large sample of the ModelSet models that were detected as dummy by the initial heuristics. The heuristics were iteratively improved until all detected models are considered dummy models. For the detection of dummy models, we defined the following four heuristics. The user can choose to select a subset or all of them.

- **Name length filter** – To exclude models with dummy or meaningless element names (e.g., `a1`, `b2`, or `x y`), we apply a `MIN_NAME_LENGTH` filter with a default value of 2. This filter is used in two ways. First, models are excluded if the `MEDIAN_NAME_LENGTH` of all element names is less than `MIN_NAME_LENGTH`. Second, models are also excluded if more than `MIN_NAME_LENGTH_THRESHOLD` of the element names (default threshold = 0.3) have length less than `MIN_NAME_LENGTH`.

- **Dummy class names** - dummy models often have elements with names like 'class1', 'class A', 'my class'. We put a threshold `DUMMY_CLASSES_THRESHOLD` (default value = 0.3) on the fraction of such names a model can have to be considered a valid model.
- **Stopword keywords** - Based on manual inspection, we curated a list of frequent UML keywords, e.g., 'control flow' and 'opaque action'. If the fraction of element names that are stopwords exceeds `STOPWORDS_THRESHOLD` (default value = 0.4), we consider the model as a dummy model.
- **General Pattern** - Users can define their own regex patterns along with a `GENERIC_PATTERN_THRESHOLD` (default value = 0.3) that can be used to filter out dummy models. E.g., there are several models with names 'att1', 'attr1' therefore we added a pattern that detects such cases as an instance of a general pattern.

Multiple heuristics can be used in combination, and there may be some overlap. It is important to note that our set of heuristics is specific to UML models as they were curated by manual inspection and exploration of UML models. Although they provide practical and effective criteria for identifying dummies in this context, we do not claim that they are exhaustive or applicable to all situations. MCP4CM encourages users to define their own heuristics for dummy filtering based on their dataset and its inspection.

2) *Duplicate Filtering*: As described in Section II, a dataset can contain a lot of duplicate models, therefore, duplicate filtering is a crucial step. Currently, two techniques of duplicate

TABLE I: Cleansing results of each MCP4CM technique applied to ModelSet

Type	Technique	#Detected	% Detected
Dummy Filtering	Name Length Filter	188	3.67%
	Dummy Class Names	288	5.62%
	Dummy Keywords	111	2.16%
	General Pattern Filter	176	3.43%
Duplicate Filtering	Hash	2043	39.90%
	TF-IDF	3754	73.32%
Language Filtering	Exclude Non English	677	13.22%

filtering are implemented in MCP4CM.

- **Hash-based Duplicate Filtering** - this approach enables exact matching of models based on the hash of the data extracted from the `model_key` of a model.
- **Vector-based Duplicate Filtering** - this is an advanced vector-based similarity approach for performing duplicate filtering. Similar models are identified by the cosine similarity between their TF-IDF vectors. Two models with a similarity score greater than a certain threshold `TFIDF_SIMILARITY_THRESHOLD` (adjustable, default value = 0.8) are considered duplicates.

3) *Language Filtering*: For language detection, we used the python library `langdetect`<sup>2</sup>. Since type names in the original XMI files are set by the modeling tool, they are often in English regardless of the language(s) of the model. Therefore, only the element names are used in this step.

Note that all the parameters (e.g., `MIN_NUM_ELEMENTS`, `TFIDF_SIMILARITY_THRESHOLD`) in the filtering steps are configurable parameters in MCP4CM that users can easily change based on their requirements and needs. The default thresholds balance the trade-off between identifying problematic models and avoiding false positives (e.g, a non-dummy model can have a few short element names like ‘TV’).

#### IV. EVALUATION

In the following, we present an empirical evaluation of our framework. First, we evaluate the effectiveness of MCP4CM in cleansing the ModelSet [18] dataset. Afterward, we underpin the necessity of our pipeline by evaluating its impact in the context of two reproducibility studies that largely rely on manually assigned labels to filter out noisy data. For this purpose, two ML4MDE studies from the related literature that also utilize the ModelSet dataset were selected ([27], [28]). A reproducibility package for all experiments is provided as supplementary material for this paper on Zenodo<sup>3</sup>.

##### A. Dataset Description

ModelSet is a labeled dataset of 5,466 Ecore meta-models and 5,120 UML models, specifically created to enable ML research in the MDE domain. The models were collected from GitHub and GenMyModel and labeled using the semi-automated Greedy Methodology for Fast Labeling

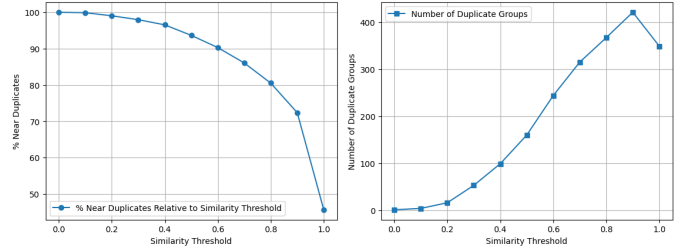


Fig. 2: TF-IDF % Near Duplicate and Duplicate Groups vs Similarity Threshold

(GMFL) [18]. Each UML model is assigned a category (main label) that represents the domain of the model. There are two special cases within the category label: ‘unknown’, used when the category of the model was not identifiable, and ‘dummy’, representing trivial models. Additional label tags contain keywords that provide further context to the main label. ModelSet offers multiple data formats to support different ML approaches, i.e., raw XMI that preserves a model’s original structure and metadata, a .txt file that contains extracted names of the model elements, and a file with a graph representation of the model. Finally, ModelSet contains a table with extracted numerical features that represent the number of times a specific element is present in a particular model.

##### B. Cleansing ModelSet with MCP4CM

Table I shows the result of applying each MCP4CM filter separately on all the 5,120 ModelSet models. In each case, we considered the default values of the configuration parameters. In the case of hash duplicate filtering, which involves an exact match, we see that around **40% of the models are exact duplicates**. This should be of crucial interest to anyone interested in using the ModelSet out of the box. We further see that TF-IDF duplicate filtering with a threshold of 0.8 identifies more than 70% of the models as duplicates. Fig. 2 shows the duplicates found relative to the chosen similarity threshold (left) and the number of groups of similar models (right). The total unique models are the set of models that have no duplicates and models that represent groups of duplicates.

ModelSet includes manual annotations of which models are dummy models. We can compare these annotations with the dummies detected by MCP4CM. Out of the 606 models that are labeled as dummy in ModelSet, 60 models are not flagged by the MCP4CM heuristics. Note that these numbers might contain duplicate models. The heuristics are general rules, so care should be taken to avoid labeling non-dummy models as dummy models (minimizing false positives). With manual inspection of each model, as is done with manual annotations, this is not an issue. Therefore, the heuristics may identify fewer dummy models. On the other hand, the heuristics identify 28 dummy models that are not labeled as dummy in ModelSet. There are several reasons why the heuristics identify these models as dummies; some models were labeled ‘unknown’ in ModelSet, and in other models, only a portion of the elements have dummy names.

<sup>2</sup><https://pypi.org/project/langdetect/>

<sup>3</sup><https://doi.org/10.5281/zenodo.16365460>

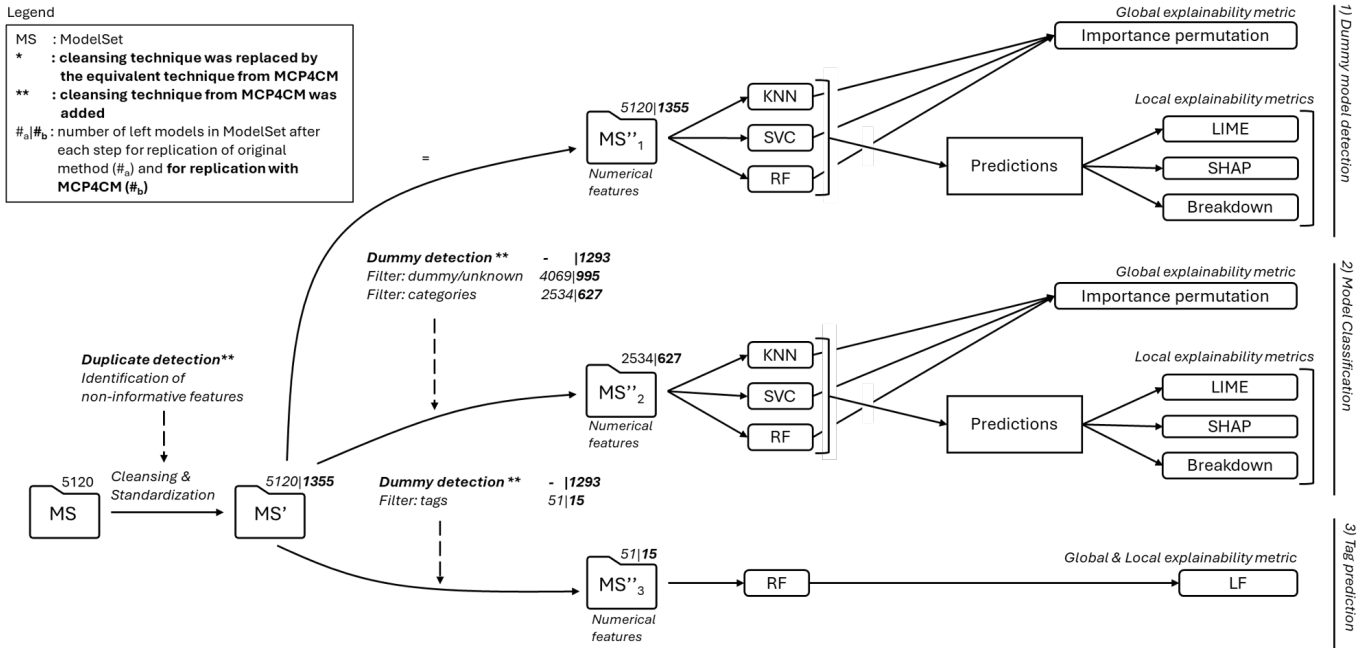


Fig. 3: Methodology for RS<sub>1</sub>

ModelSet also contains manually annotated language tags. Since the model files can contain words in different languages and each model only gets one tag, we compare the tags to the language filtering of MCP4CM. 4,497 models were tagged as English in ModelSet and 4,443 English models were identified by MCP4CM, with a strong overlap of 4,224 models.

Overall, our results show that without systematic data cleansing, a dataset can contain a large number of models of at least questionable quality that can severely affect the training of ML models and, subsequently, the effectiveness of the ML downstream tasks. To demonstrate the need for and the effectiveness of MCP4CM, we reproduce the experiments of two exemplary studies on ML4MDE. Given that the initial version of MCP4CM contains cleansing techniques for UML models, we selected studies that use UML data from ModelSet. First, the experiments are replicated as-is to ensure that the results of the downstream tasks can be reproduced exactly. Then we replicate the tasks after applying MCP4CM to investigate the effect of the pipeline on the final results.

### C. RS<sub>1</sub>: Replication Study 1

**Description.** In this study, the authors train ML models to perform three classification tasks on the ModelSet dataset: Dummy Model Detection (DMD), Model Classification (MC), and Model Tag Prediction (MTP) [28]. After training their ML models, they generate global and local explanations.

**Data Cleansing and Model Training.** Fig. 3 presents an overview of the methodology of this study. The authors perform several data cleansing steps, both in general and specifically for each task. The general data cleansing consists of dropping noninformative features based on an exploratory analysis and a correlation analysis, and standardization of the numerical features using z-score normalization.

For the DMD task, only general cleansing steps were performed, after which three separate classifiers were built (KNN, SVC, and RF). For each, hyperparameters were tuned using a grid search approach with 5-fold cross-validation followed by global and local explainability approaches (global: permutation feature importance; local: LIME, SHAP, and Breakdown).

For the MC task, the dataset was further cleansed by filtering out models categorized as ‘dummy’ or ‘unknown’ and models that belong to infrequent categories. Then, the same classifiers and explainability metrics of the DMD task were applied.

The final MTP task is a multi-label classification task where each model can be associated with more than one tag. To mitigate low label density, further processing of the dataset was needed to remove those models that contained no tags, or only a single tag [28]. In contrast to the other two tasks, there are not many explainability approaches supported for MTP. As a result, only the RF classifier was built due to its compatibility with LionForests (LF) which is capable of generating label-specific explanations for multi-label classification.

After training the classifiers with original data and the data cleansed after applying MCP4CM, we evaluate the significance of the difference in the ML model performance. First a Shapiro test [41] is used to test for normal distribution in 100 runs. If the results are not normally distributed, significance is tested with a Mann-Witney U test [42]. Otherwise, we conduct an independent t-test to check significance.

**Replication Results.** We replicate this study using the provided replication package and obtain the same results. We then replicate the study again, this time including techniques from the MCP4CM pipeline. Specifically, we performed duplicate filtering for all three tasks and dummy filtering for the model classification and tag prediction tasks (see Fig. 3). We do not include the dummy filtering from MCP4CM for the dummy

TABLE II: Comparison of metrics for the Dummy Detection and Model Classification from the  $RS_1$  between the original and MCP4CM cleansed data, averaged on 100 runs. Bold values show statistical significance compared to the original,  $\alpha = 0.05$

Dataset	Task	Classifier	Best params	Balanced Accuracy (Avg $\pm$ SD)	Precision (Avg $\pm$ SD)	Recall (Avg $\pm$ SD)	F1-Score (Avg $\pm$ SD)
Original	Dummy	SVC	C: 300, cw: None, $\gamma$ : 0.1, k: rbf	0.88 $\pm$ 0.02	0.87 $\pm$ 0.03	0.77 $\pm$ 0.03	0.82 $\pm$ 0.02
MCP4CM	Dummy	SVC	C: 1000, cw: None, $\gamma$ : 1.0, k: rbf	<b>0.65 <math>\pm</math> 0.04</b>	<b>0.44 <math>\pm</math> 0.09</b>	<b>0.33 <math>\pm</math> 0.08</b>	<b>0.37 <math>\pm</math> 0.07</b>
Original	Dummy	KNN	ls: 5, nn: 3, p: 1, w: distance	0.88 $\pm$ 0.01	0.89 $\pm$ 0.05	0.78 $\pm$ 0.03	0.83 $\pm$ 0.03
MCP4CM	Dummy	KNN	ls: 5, nn: 2, p: 1, w: distance	<b>0.68 <math>\pm</math> 0.04</b>	<b>0.49 <math>\pm</math> 0.09</b>	<b>0.38 <math>\pm</math> 0.08</b>	<b>0.43 <math>\pm</math> 0.07</b>
Original	Dummy	RF	cw: None, md: None, msl: 1, mss: 5, ne: 300	0.88 $\pm$ 0.01	0.95 $\pm$ 0.02	0.78 $\pm$ 0.03	0.85 $\pm$ 0.02
MCP4CM	Dummy	RF	cw: balanced, md: None, msl: 3, mss: 2, ne: 100	<b>0.70 <math>\pm</math> 0.04</b>	<b>0.41 <math>\pm</math> 0.09</b>	<b>0.44 <math>\pm</math> 0.09</b>	<b>0.41 <math>\pm</math> 0.07</b>
Original	Multiclass	SVC	C: 500, cw: None, $\gamma$ : 0.1, k: rbf	0.76 $\pm$ 0.01	0.79 $\pm$ 0.01	0.76 $\pm$ 0.01	0.78 $\pm$ 0.01
MCP4CM	Multiclass	SVC	C: 10.0, cw: balanced, $\gamma$ : 0.1, k: rbf	<b>0.29 <math>\pm</math> 0.03</b>	<b>0.30 <math>\pm</math> 0.03</b>	<b>0.29 <math>\pm</math> 0.03</b>	<b>0.28 <math>\pm</math> 0.03</b>
Original	Multiclass	KNN	ls: 50, nn: 3, p: 1, w: distance	0.79 $\pm$ 0.01	0.79 $\pm$ 0.02	0.79 $\pm$ 0.01	0.79 $\pm$ 0.02
MCP4CM	Multiclass	KNN	ls: 50, nn: 10, p: 1, w: distance	<b>0.30 <math>\pm</math> 0.02</b>	<b>0.29 <math>\pm</math> 0.03</b>	<b>0.30 <math>\pm</math> 0.02</b>	<b>0.29 <math>\pm</math> 0.02</b>
Original	Multiclass	RF	cw: None, md: None, msl: 1, mss: 2, ne: 200	0.79 $\pm$ 0.01	0.83 $\pm$ 0.02	0.79 $\pm$ 0.01	0.81 $\pm$ 0.01
MCP4CM	Multiclass	RF	cw: balanced, md: 10, msl: 1, mss: 3, ne: 100	<b>0.33 <math>\pm</math> 0.03</b>	<b>0.32 <math>\pm</math> 0.03</b>	<b>0.33 <math>\pm</math> 0.03</b>	<b>0.32 <math>\pm</math> 0.03</b>

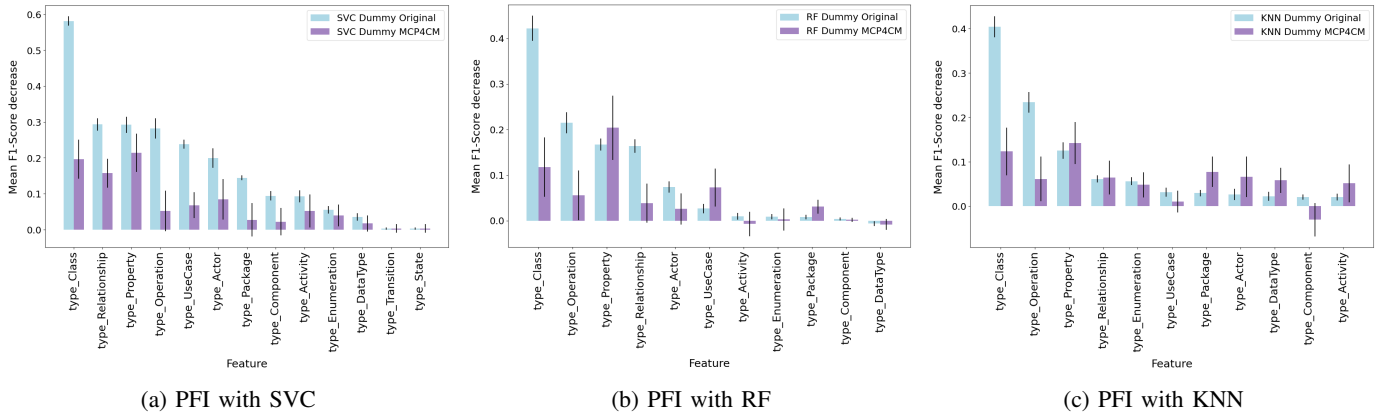


Fig. 4: Comparison for global permutation feature importance results of DMD of the  $RS_1$  study with SVC, RF, and KNN

TABLE III: Comparison of the performance metrics for MTP from the  $RS_1$  between original and MCP4CM cleansed data across different classifiers, averaged on 100 runs. Bold values show statistical significance compared to the original,  $\alpha = 0.05$

Dataset	Best params	Precision Macro (Avg $\pm$ SD)	Recall Macro (Avg $\pm$ SD)	F1-Score Macro (Avg $\pm$ SD)
Original	md: 7, mf: sqrt, b: True, msl: 1, ne: 500	0.43 $\pm$ 0.06	0.42 $\pm$ 0.08	0.41 $\pm$ 0.07
MCP4CM	md: 5, mf: None, b: True, msl: 1, ne: 10	<b>0.37<math>\pm</math>0.11</b>	0.41 $\pm$ 0.11	<b>0.38<math>\pm</math>0.10</b>

detection task because this task aims to learn the dummy detection features from the data itself. Therefore, we need valid and dummy models in the dataset.

We conduct the experiments 100 times with the best found hyperparameters and the results are averaged to account for different train-test splits. Table II presents the reproduced results from the study and the results produced after applying MCP4CM for the DMD and MC tasks. Table III shows the results for the MTP task.

Our results show that for all the trained classifiers, **the difference between not applying and applying the MCP4CM**

**techniques is significant** for DMD and MC tasks. The results consistently show that the performance of the trained ML models decreases while using the cleansed dataset. In the case of MTP, we get significant results except for recall. However, the pattern is still followed, i.e., the scores for the cleansed dataset drop in comparison to the original dataset. These results indicate an inflation of results due to data quality, more specifically data duplication.

We also replicate the explainability approaches as presented in the original study. Regarding global explainability that explains the importance of features in a given ML model performance, Fig. 4 and Fig. 5 show the comparison between the features importance results obtained in the original study and the ones obtained after applying MCP4CM across all three classifiers for DMD and MC, respectively. Fig. 6 shows the same comparison for MTP and RF classifier. The figures show notable differences in the contribution of the features between the two setups in each of the three tasks and for each of the applied classifiers. Fig. 4a shows that when using the MCP4CM cleansed ModelSet compared to the original one, not only the individual importance of a given feature towards F1-score changes (in some cases considerably drops and in

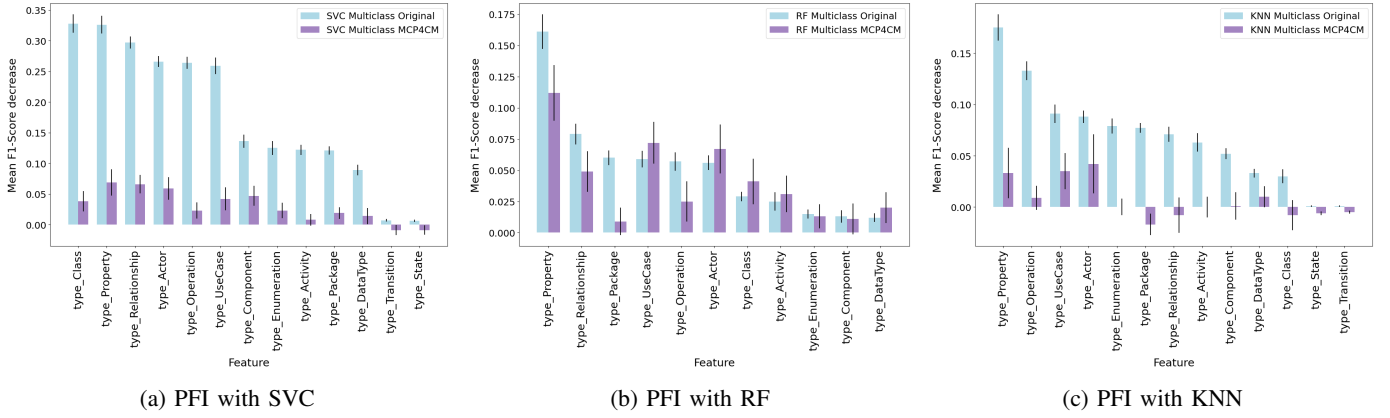


Fig. 5: Comparison for global permutation feature importance results of MC of the  $RS_1$  with SVC, RF, and KNN

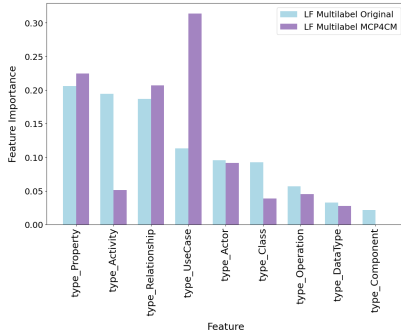


Fig. 6: Comparison for global LF results for MTP of  $RS_1$

some cases increases) but also the relative importance of the features changes. For example., the features ‘type\_Operation’, ‘type\_UseCase’, ‘type\_Actor’ follow a decreasing order of relative importance for the original dataset whereas these features follow an increasing order of relative importance for our cleansed dataset. This indicates the effect of using MCP4CM on the explainability of the results. In the case of explainability of the MTP task, we get a similar pattern in Fig. 6 whereby a cleansed ModelSet leads to a difference in the individual and relative importance of features.

We also replicated the local explainability results. However, these are difficult to compare as the cleansing of the dataset results in different instances being chosen for calculating the local explanations in each setting. Given that the most frequent classes are selected for the classification task, cleansing the dataset changes the distribution of the most frequent classes as well. Therefore, very different classes are selected in our case for the classification in the second and third tasks. Since the results so far already demonstrate the impact of applying additional cleansing techniques, we do not discuss the replication of the local explainability results here. Interested readers can consult the results of this replication study in the supplementary material for more information <sup>3</sup>.

#### D. $RS_2$ : Replication Study 2

**Description** An overview of different ML models for model classification is provided in [27]. The authors investigate

the effects of different ML models, different software model encodings, and duplication in ModelSet. The replicated portion of the methodology of this study is presented in Fig. 7.

**Data Cleansing and Model Training.** The data cleansing consists of three steps. The first step is to filter out models belonging to infrequent categories (with less than seven models), models of the category ‘dummy’ and ‘unknown’, and models that are not in English (based on the tags). The second step consists of detecting and removing duplicates. Duplicates are detected by an algorithm adapted from [29] using the Jaccard similarity to calculate the similarity between models. In the third step, models that are part of infrequent categories (with less than 10 occurrences) are removed.

The resulting model dataset is encoded as numerical feature vectors, and ML models are trained with a 10-fold validation approach. The analysis is conducted on the Ecore and UML dataset with and without duplicates. The results show that FFNN and SVM perform the best in all scenarios. The best-performing encoding techniques are TF-IDF for Ecore models and word embedding for UML. Overall, the accuracy of all methods decreases after dropping duplicate models. *The authors suggest that detecting and removing duplicates is an important avenue for future work in MLAMDE.*

To demonstrate the effect of the MCP4CM cleansing techniques, we replicate the portion of this study that works with the TF-IDF and word embeddings of the deduplicated UML ModelSet and obtain the same results. Then, we replicate the study with the following alterations to the methodology: we substitute the original duplicate detection algorithm with that of MCP4CM and replace tag-based language filtering with MCP4CM’s language detection to filter out non-English models. Additionally, dummy models are filtered using MCP4CM’s dummy detection, and all models labeled as ‘dummy’ or ‘unknown’ in the original ModelSet are excluded to avoid leftover classes irrelevant for model classification. We train the ML models as provided in the original study with the cleansed ModelSet (see Table IV). For this study, we conducted the experiments 30 times with the best hyperparameters. The difference in the number of runs for  $RS_1$  and

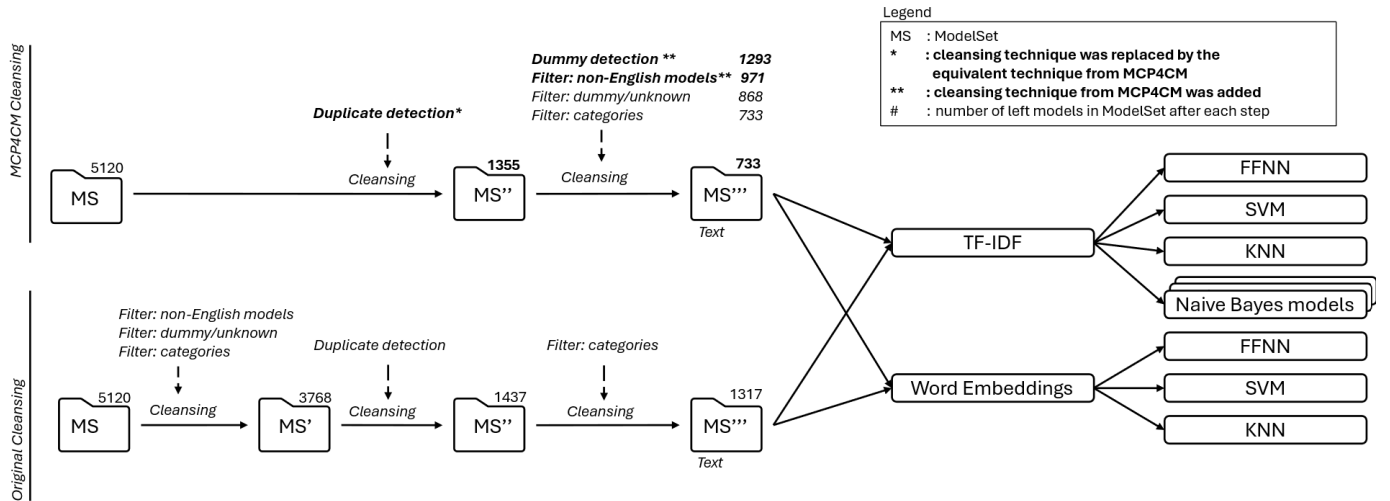


Fig. 7: Methodology for RS<sub>2</sub>

RS<sub>2</sub> comes from the fact that training of neural networks (as done in RS<sub>2</sub>) requires more time than training of traditional ML models (as done in RS<sub>1</sub>). Having in mind that statistically stable results can be expected after 30 runs [43], we consider this sufficient.

**Replication Results.** The original study already notes that accuracy drops when duplicates are removed, compared to when they are retained. Table IV compares RS<sub>2</sub> performance on original vs. MCP4CM-cleansed data across classifiers, showing significant performance changes across all ML models except KNN and GNB. It also highlights the original ranking of each ML model and how these rankings change with MCP4CM cleansing.

It is interesting to note that the accuracy increases on the MCP4CM-cleansed dataset compared to the original. One potential explanation is that the applied cleansing procedures result in a dataset with fewer trivial or noisy models, which can benefit representation learning techniques. TF-IDF and word2vec embeddings derived from less noisy and more consistent input data have the potential to improve the performance of downstream models. In contrast to RS<sub>1</sub>, where numerical features may insufficiently capture model semantics, RS<sub>2</sub> relies on embeddings that can benefit more directly from the quality of input data. In general, the findings of RS<sub>2</sub> demonstrate that cleansing techniques used in a project can significantly impact the results of the model classification task.

## V. DISCUSSION

Despite the growing interest in the field of ML4MDE in recent years, not enough emphasis has been placed on the quality of the model datasets used to train ML models. These datasets are widely leveraged in various ML4MDE studies, yet their overall quality is rarely examined. Data preprocessing efforts tend to be specific to individual studies, offering limited transparency or generalizability. There is a lack of a comprehensive overview of general and modeling language-specific data cleansing approaches for model datasets. In this paper, we address this research gap by proposing the first

iteration of MCP4CM with three components: Model Dataset Parsing, Model Dataset Loading, and Model Dataset Filtering. Currently, MCP4CM has been implemented for UML models. However, the pipeline is constructed to be highly extensible and open-source. It provides support for extending the implementation to other modeling languages in the future.<sup>4</sup>

In RS<sub>1</sub>, we demonstrate the importance of removing duplicates from a model dataset (ModelSet) prior to training ML models, since failing to do so can lead to data leakage and inflated performance metrics. As reported in related work by Nikoo et al. [31] and Babur et al. [32], model clones are not only models that are exact matches, but also models that have a high degree of similarity. Additionally, the same model can be collected from repositories from different modeling tools that each add different meta-data to the XMI files. The MCP4CM duplicate detection algorithms take all these concerns into account. The replication study reveals a significant drop in evaluation metrics for all classifiers across all three tasks, underscoring the impact of removing duplicate models. This leads to our first implication - *Model cleansing has a significant impact on the performance of the ML model in ML4MDE tasks*. Moreover, our results also show that data quality influences not only performance, but also the explainability of model behavior. Depending on the quality of the data, ML models may learn to prioritize different sets of features when assigning class labels. This finding highlights a second implication - *Model cleansing can considerably affect the interpretability of ML outcomes in ML4MDE tasks*.

RS<sub>2</sub> demonstrates how different implementations of cleansing techniques can significantly influence the accuracy of the ML models. In this replication, for a subset of ML models from the original study, we obtain results on the dataset cleansed with the original cleansing methods, as well as with the MCP4CM cleansing methods. Consistent improvements in performance were observed across all experiments, most of which were significant. This provides further empirical

<sup>4</sup>Duplicate detection and language detection depend only on the model element names (and types) and can be configured for any modeling language.

TABLE IV: Comparison of the performance metrics for  $RS_2$  between the original and MCP4CM cleansed data across different classifiers, averaged on 30 runs. Bold MCP4CM values are significantly different from the original,  $\alpha = 0.05$

Model	Encoding	Original Acc.	MCP4CM Acc.	Original Best hyper.	MCP4CM Best hyper.	Original Rank	MCP4CM Rank
FFNN	TFIDF	0.758409 $\pm$ 0.034741	<b>0.782893</b> $\pm$ <b>0.031025</b>	hidden layer = 50	hidden layer = 200	1	3
FFNN	WordE	0.750704 $\pm$ 0.025532	<b>0.791098</b> $\pm$ <b>0.034357</b>	hidden layer = 150	hidden layer = 200	2	1
SVM	WordE	0.738369 $\pm$ 0.032936	<b>0.790480</b> $\pm$ <b>0.038238</b>	kernel = rbf, C = 100	kernel = rbf, C = 10	3	2
SVM	TFIDF	0.730103 $\pm$ 0.030521	<b>0.772314</b> $\pm$ <b>0.038073</b>	kernel = linear, C = 10	kernel = linear, C = 10	4	4
CNB	TFIDF	0.692885 $\pm$ 0.031471	<b>0.747527</b> $\pm$ <b>0.035903</b>	alpha = 0.1	alpha = 0.1	5	5
KNN	TFIDF	0.674414 $\pm$ 0.030267	<b>0.721247</b> $\pm$ <b>0.035361</b>	k = 1	k = 5	6	6
KNN	WordE	0.667139 $\pm$ 0.037451	0.684065 $\pm$ 0.045898	k = 1	k = 4	7	8
GNB	TFIDF	0.618875 $\pm$ 0.038091	0.636232 $\pm$ 0.039672	–	–	8	9
MNB	TFIDF	0.616596 $\pm$ 0.021059	<b>0.713169</b> $\pm$ <b>0.041369</b>	alpha = 0.1	alpha = 0.1	9	7

evidence that model cleansing meaningfully impacts ML outcomes in ML4MDE tasks. While  $RS_1$  particularly highlights the importance of mitigating inflated performance reports via duplicate removal,  $RS_2$  indicates that the quality of the remaining data is also of utmost importance, especially in tasks that use semantic rich encodings, such as TF-IDF or word embeddings. Representation learning methods are sensitive to noise and inconsistencies within the dataset, hence they benefit from cleaner input data, yielding more informative and discriminative feature representations for downstream classification. This finding is consistent with prior research [44], [45], which investigates how factors like accuracy, completeness, and consistency in training and test data are crucial for developing reliable models and shows that data quality, rather than quantity, plays a more important role in enhancing model accuracy. From this, we derive our third implication – *Model cleansing has an important impact on the model encoding, i.e., its vector representation, especially when using techniques such as TF-IDF or word embeddings. The quality of the vector representations subsequently impacts the performance of the ML model in downstream ML4MDE tasks.*

Together, the two replication studies answer our research question. Both studies show (mostly) significantly different results with the MCP4CM pipeline. We did not investigate the individual importance of each cleansing filter. It is therefore not possible to explain how different cleansing techniques affect the accuracy of ML techniques solely based on the reproducibility studies we performed. Further research is required to identify causalities and formulate concrete guidelines.

**Threats to Validity.** There are several factors that may impact the validity of our findings. *i) Conclusion Validity:* The effectiveness of our cleansing pipeline is contingent on the quality of the initial datasets. We noted that around 40% of the models were exact matches, which may not be true for different datasets and thereby can indicate a potential overestimation of our results, *ii) External Validity:* The generalizability of our pipeline to entirely new domains, e.g., models in the medical domain, remains uncertain. Further validation is necessary to confirm its efficacy across broader applications and real-world scenarios and *iii) Construct Validity:* Performance metrics used to evaluate the effect of our cleansing pipeline may not fully capture all dimensions of model improvement and therefore, additional assessments that capture the causality of

our results may be required to obtain a holistic understanding of the benefits of our work.

## VI. CONCLUSION

In this paper, we introduced MCP4CM, a comprehensive, out-of-the-box usable and highly configurable model cleansing pipeline aimed at enhancing the quality of model repositories used to train ML models and thereby enhance the reliability of ML models. By implementing a systematic approach to data cleansing by means of dummy filtering, duplicate removal, and language-based filtering, MCP4CM ensures robustness in several downstream ML4MDE tasks such as dummy model detection, domain classification, and tag prediction. Our experiments demonstrate a significant impact not just on the ML model’s accuracy but also a clear difference in explainability. Therefore, our results underscore the necessity to mitigate the garbage in, garbage out problem. It is important to note that the objective of our work was not to show our approach outperforms existing works or to even discredit the works we reproduced. Instead, our focus was to show that model cleansing has a measurable impact on ML4MDE research.

Overall, there are many avenues for future work in this domain. In the general domain of ML, there has been research on the effect of data quality and preprocessing on the results of ML approaches [46]. In the domain of ML4MDE, similar research efforts should be made to identify model dataset quality characteristics and cleansing practices. MCP4CM establishes a first step toward a comprehensive, open pipeline of model dataset cleansing. Such a pipeline would facilitate research convergence, enable benchmarking and comparability of ML- and data-driven MDE research. We release MCP4CM open source for researchers to use and extend it <sup>5 6</sup>.

## ACKNOWLEDGEMENTS

This study was supported by the TU Wien Career Grant ‘Recommender Systems in Model-driven Engineering’. J. Neidhardt gratefully acknowledges financial support from the Austrian Federal Ministry of Labour and Economy, the National Foundation for Research, Technology and Development, and the Christian Doppler Research Association.

<sup>5</sup>Python Package: <https://pypi.org/project/mcp4cm/>

<sup>6</sup>Zenodo Repository: <https://doi.org/10.5281/zenodo.16410868>

## REFERENCES

- [1] N. Bencomo, J. Cabot, M. Chechik, B. H. Cheng, B. Combemale, S. Zschaler *et al.*, “Abstraction engineering,” *arXiv preprint arXiv:2408.14074*, 2024.
- [2] M. Brambilla, J. Cabot, and M. Wimmer, “Model-driven software engineering in practice,” *Synthesis lectures on software engineering*, vol. 3, no. 1, pp. 1–207, 2017.
- [3] A. C. Marcén, A. Iglesias, R. Lapeña, F. Pérez, and C. Cetina, “A systematic literature review of model-driven engineering using machine learning,” *IEEE Trans. Software Eng.*, vol. 50, no. 9, pp. 2269–2293, 2024.
- [4] M. B. Chaaben, L. Burgueño, I. David, and H. A. Sahraoui, “On the utility of domain modeling assistance with large language models,” *CoRR*, vol. abs/2410.12577, 2024.
- [5] M. Weyssow, H. Sahraoui, and E. Syriani, “Recommending metamodel concepts during modeling activities with pre-trained language models,” *Software and Systems Modeling*, vol. 21, no. 3, pp. 1071–1089, 2022.
- [6] S. J. Ali and D. Bork, “A graph language modeling framework for the ontological enrichment of conceptual models,” in *International Conference on Advanced Information Systems Engineering*. Springer, 2024, pp. 107–123.
- [7] C. Arora, M. Sabetzadeh, S. Nejati, and L. Briand, “An active learning approach for improving the accuracy of automated domain model extraction,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 28, no. 1, pp. 1–34, 2019.
- [8] R. Saini, G. Mussbacher, J. L. Guo, and J. Kienzle, “Domobot: A modelling bot for automated and traceable domain modelling,” in *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE, 2021, pp. 428–429.
- [9] L. Burgueño, J. Cabot, and S. Gérard, “An lstm-based neural network architecture for model transformations,” in *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2019, Munich, Germany, September 15-20, 2019*, M. Kessentini, T. Yue, A. Pretschner, S. Voss, and L. Burgueño, Eds. IEEE, 2019, pp. 294–299.
- [10] K. Lano, S. Kolahdouz-Rahimi, and S. Fang, “Model transformation development using automated requirements analysis, metamodel matching, and transformation by example,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–71, 2021.
- [11] P. T. Nguyen, D. Di Ruscio, A. Pierantonio, J. Di Rocco, and L. Iovino, “Convolutional neural networks for enhanced classification mechanisms of metamodels,” *Journal of Systems and Software*, vol. 172, p. 110860, 2021.
- [12] J. A. H. López, J. S. Cuadrado, R. Rubei, and D. Di Ruscio, “Modelxglue: a benchmarking framework for ml tools in mde,” *Software and Systems Modeling*, pp. 1–24, 2024.
- [13] J. Michael, D. Bork, M. Wimmer, and H. C. Mayr, “Quo vadis modeling?” *Software and Systems Modeling*, pp. 1–22, 2023.
- [14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.
- [15] A. Asuncion, D. Newman *et al.*, “Uci machine learning repository,” 2007. [Online]. Available: <https://archive.ics.uci.edu/>
- [16] J. Di Rocco, D. Di Ruscio, L. Iovino, and A. Pierantonio, “Collaborative repositories in model-driven engineering [software technology],” *IEEE Software*, vol. 32, no. 3, pp. 28–34, 2015.
- [17] G. Robles, T. Ho-Quang, R. Hebig, M. R. Chaudron, and M. A. Fernandez, “An extensive dataset of uml models in github,” in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 519–522.
- [18] J. A. H. López, J. L. C. Izquierdo, and J. S. Cuadrado, “Modelset: a dataset for machine learning in model-driven engineering,” *Softw. Syst. Model.*, vol. 21, no. 3, pp. 967–986, 2022. [Online]. Available: <https://doi.org/10.1007/s10270-021-00929-3>
- [19] J. A. H. López and J. S. Cuadrado, “An efficient and scalable search engine for models,” *Software and Systems Modeling*, vol. 21, no. 5, pp. 1715–1737, 2022.
- [20] P.-L. Glaser, E. Sallinger, and D. Bork, “The extended ea modelset—a fair dataset for researching and reasoning enterprise architecture modeling practices,” *Software and Systems Modeling*, pp. 1–19, 2025.
- [21] L. M. Hillah and F. Kordon, “Petri nets repository: a tool to benchmark and debug petri net tools,” in *Application and Theory of Petri Nets and Concurrency: 38th International Conference, PETRI NETS 2017, Zaragoza, Spain, June 25–30, 2017, Proceedings 38*. Springer, 2017, pp. 125–135.
- [22] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi *et al.*, “Repository: a repository platform for sharing business process models.” *BPM (PhD/Demos)*, vol. 2420, pp. 149–153, 2019.
- [23] M. Saeedi Nikoo, S. Kochanthara, Ö. Babur, and M. van den Brand, “An empirical study of business process models and model clones on github,” *Empirical Software Engineering*, vol. 30, no. 2, p. 48, 2025.
- [24] I. Compagnucci, F. Corradini, F. Fornari, and B. Re, “Trends on the usage of bpmn 2.0 from publicly available repositories,” in *International Conference on Business Informatics Research*. Springer, 2021, pp. 84–99.
- [25] Önder Babur, “A labeled ecore metamodel dataset for domain clustering,” Mar. 2019. [Online]. Available: <https://doi.org/10.5281/zenodo.2585456>
- [26] P. P. F. Barcelos, T. P. Sales, M. Fumagalli, C. M. Fonseca, I. V. Sousa, E. Romanenko, J. Kritz, and G. Guizzardi, “A fair model catalog for ontology-driven conceptual modeling research,” in *International Conference on Conceptual Modeling*. Springer, 2022, pp. 3–17.
- [27] J. A. H. López, R. Rubei, J. S. Cuadrado, and D. di Ruscio, “Machine learning methods for model classification: a comparative study,” in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 165–175. [Online]. Available: <https://doi.org/10.1145/3550355.3552461>
- [28] F. J. Alcaide, J. R. Romero, and A. Ramírez, “Can explainable artificial intelligence support software modelers in model comprehension?” *Software and Systems Modeling*, pp. 1–26, 2025.
- [29] M. Allamanis, “The adverse effects of code duplication in machine learning models of code,” in *Proceedings of the 2019 ACM SIGPLAN international symposium on new ideas, new paradigms, and reflections on programming and software*, 2019, pp. 143–153.
- [30] H. Störrle, “Towards clone detection in uml domain models,” in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, 2010, pp. 285–293.
- [31] M. S. Nikoo, Ö. Babur, and M. van den Brand, “Clone detection for business process models,” *PeerJ Computer Science*, vol. 8, p. e1046, 2022.
- [32] Ö. Babur, L. Cleophas, and M. Van Den Brand, “Metamodel clone detection with samos,” *Journal of Computer Languages*, vol. 51, pp. 57–74, 2019.
- [33] M. H. Alalfi, J. R. Cordy, T. R. Dean, M. Stephan, and A. Stevenson, “Models are code too: Near-miss clone detection for simulink models,” in *2012 28th IEEE international conference on software maintenance (ICSM)*. IEEE, 2012, pp. 295–304.
- [34] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
- [35] L. Breiman, “Random forests,” *Machine learning*, vol. 45, pp. 5–32, 2001.
- [36] M. T. Ribeiro, S. Singh, and C. Guestrin, “‘‘ why should i trust you?’’ explaining the predictions of any classifier,” in *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016, pp. 1135–1144.
- [37] S. M. Lundberg and S.-I. Lee, “A unified approach to interpreting model predictions,” *Advances in neural information processing systems*, vol. 30, 2017.
- [38] M. Staniak and P. Biecek, “Explanations of model predictions with live and breakdown packages,” *arXiv preprint arXiv:1804.01955*, 2018.
- [39] I. Mollas, N. Bassiliades, I. Vlahavas, and G. Tsoumakas, “Lionforests: local interpretation of random forests,” *arXiv preprint arXiv:1911.08780*, 2019.
- [40] J. A. H. López, J. L. C. Izquierdo, and J. S. Cuadrado, “Using the modelset dataset to support machine learning in model-driven engineering,” in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2022, pp. 66–70.
- [41] S. S. Shapiro and M. B. Wilk, “An analysis of variance test for normality (complete samples),” *Biometrika*, vol. 52, no. 3-4, pp. 591–611, 1965.
- [42] P. E. McKnight and J. Najab, “Mann-whitney u test,” *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.
- [43] A. Arcuri and L. Briand, “A practical guide for using statistical tests to assess randomized algorithms in software engineering,”

in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1–10. [Online]. Available: <https://doi.org/10.1145/1985793.1985795>

- [44] A. Kariluoto, J. Kultanen, J. Soininen, A. Pärnänen, and P. Abrahamsson, “Quality of data in machine learning,” in *2021 IEEE 21st international conference on software quality, reliability and security companion (QRS-C)*. IEEE, 2021, pp. 216–221.
- [45] L. Budach, M. Feuerpfeil, N. Ihde, A. Nathansen, N. Noack, H. Patzlaff, F. Naumann, and H. Harmouch, “The effects of data quality on machine learning performance,” *arXiv preprint arXiv:2207.14529*, 2022.
- [46] J. García-Carrasco, A. Maté, and J. Trujillo, “A data-driven methodology for guiding the selection of preprocessing techniques in a machine learning pipeline,” in *Intelligent Information Systems*, C. Cabanillas and F. Pérez, Eds. Cham: Springer International Publishing, 2023, pp. 34–42.