UNIVERSIDAD DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

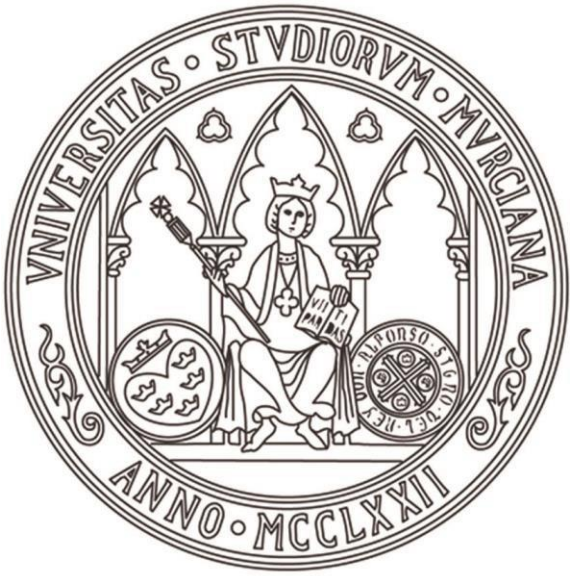
TESIS DOCTORAL

On the application of Machine Learning to Model-Driven Engineering

Aplicación de técnicas de Machine Learning a modelado de software

D. José Antonio Hernández López

2023



UNIVERSIDAD DE MURCIA

ESCUELA INTERNACIONAL DE DOCTORADO

TESIS DOCTORAL

On the application of Machine Learning to Model-Driven Engineering

Aplicación de técnicas de Machine Learning a modelado de software

Autor: D. José Antonio Hernández López

Director/es: D. Jesús Sánchez Cuadrado

UNIVERSIDAD DE MURCIA

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD DE LA TESIS PRESENTADA EN MODALIDAD DE COMPENDIO O ARTÍCULOS PARA OBTENER EL TÍTULO DE DOCTOR

Aprobado por la Comisión General de Doctorado el 19-10-2022

D./Dña. José Antonio Hernández López

doctorando del Programa de Doctorado en

Doctorado en Informática

de la Escuela Internacional de Doctorado de la Universidad Murcia, como autor/a de la tesis presentada para la obtención del título de Doctor y titulada:

On the application of Machine Learning to Model-Driven Engineering / Aplicación de técnicas de Machine Learning a modelado de software

y dirigida por,

D./Dña. Jesús Sánchez Cuadrado
D./Dña.
D./Dña.

DECLARO QUE:

La tesis es una obra original que no infringe los derechos de propiedad intelectual ni los derechos de propiedad industrial u otros, de acuerdo con el ordenamiento jurídico vigente, en particular, la Ley de Propiedad Intelectual (R.D. legislativo 1/1996, de 12 de abril, por el que se aprueba el texto refundido de la Ley de Propiedad Intelectual, modificado por la Ley 2/2019, de 1 de marzo, regularizando, aclarando y armonizando las disposiciones legales vigentes sobre la materia), en particular, las disposiciones referidas al derecho de cita, cuando se han utilizado sus resultados o publicaciones.

Además, al haber sido autorizada como compendio de publicaciones o, tal y como prevé el artículo 29.8 del reglamento, cuenta con:

- *La aceptación por escrito de los coautores de las publicaciones de que el doctorando las presente como parte de la tesis.*
- *En su caso, la renuncia por escrito de los coautores no doctores de dichos trabajos a presentarlos como parte de otras tesis doctorales en la Universidad de Murcia o en cualquier otra universidad.*

Del mismo modo, asumo ante la Universidad cualquier responsabilidad que pudiera derivarse de la autoría o falta de originalidad del contenido de la tesis presentada, en caso de plagio, de conformidad con el ordenamiento jurídico vigente.

En Murcia, a de de 20

Fdo.:

Esta DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD debe ser insertada en la primera página de la tesis presentada para la obtención del título de Doctor.

Información básica sobre protección de sus datos personales aportados	
Responsable:	Universidad de Murcia. Avenida teniente Flomesta, 5. Edificio de la Convalecencia. 30003; Murcia. Delegado de Protección de Datos: dpd@um.es
Legitimación:	La Universidad de Murcia se encuentra legitimada para el tratamiento de sus datos por ser necesario para el cumplimiento de una obligación legal aplicable al responsable del tratamiento. art. 6.1.c) del Reglamento General de Protección de Datos
Finalidad:	Gestionar su declaración de autoría y originalidad
Destinatarios:	No se prevén comunicaciones de datos
Derechos:	Los interesados pueden ejercer sus derechos de acceso, rectificación, cancelación, oposición, limitación del tratamiento, olvido y portabilidad a través del procedimiento establecido a tal efecto en el Registro Electrónico o mediante la presentación de la correspondiente solicitud en las Oficinas de Asistencia en Materia de Registro de la Universidad de Murcia

Agradecimientos

Me gustaría agradecer a todas las personas que me han ayudado y apoyado durante todo este proceso. En especial, debería nombrar a mis pilares fundamentales: María, mi hermana; Juan, mi padre y Rosa, mi madre. Ellos han sido los primeros en estar en los buenos y malos momentos, los que me sirvieron como guía cuando estaba un poco perdido y los que confiaron en mi y en mi esfuerzo.

Profundo agredecimiento a mi director de tesis, Dr. Jesús Sánchez Cuadrado por su infinita paciencia y guía, sin la cual no podría haber conseguido nada de lo que he hecho, ni de lo que hoy soy. Gracias mil y una veces por haber visto el potencial que ni yo había visto en mi mismo, por haber sido como un soporte con el que crecer y aprender. Gracias a ti, no solo soy doctor, sino que también ingeniero de software.

A Jesús, Raquel, Hamisa y Rafa. Ellos han sido fuente de verdadera amistad. Me han aconsejado y ayudado tantas veces que mis agradecimientos nunca serían suficientes.

A mi pareja Ana por su apoyo y amor incondicional. Te agradezco por tantas ayudas y tantos aportes no solo para el desarrollo de mi tesis, sino también para mi vida; eres mi inspiración y mi motivación.

Contents

List of Figures

List of Tables

Resumen

La complejidad de los sistemas que demanda la sociedad actual (p.ej., sistemas distribuidos en la nube, sistemas ciber-físicos, Internet de las cosas, sistemas colaborativos y dependientes del contexto, etc) plantea cada vez dificultades mayores para ser abordada con los paradigmas de programación mayoritarios. Aumentar el nivel de abstracción con el que se desarrolla el software es un elemento esencial para abordar esta complejidad. La Ingeniería Dirigida por Modelos (MDE; Model-Driven Engineering) propone el uso activo de modelos durante todas las fases del desarrollo de un sistema software. Así, los modelos no son solo documentación pasiva, sino que se utilizan para describir, simular, probar y generar el código de un sistema, a través de transformaciones de modelos que automatizan este proceso. En MDE, el foco del desarrollo está puesto en la creación de los modelos que representan las diferentes partes del sistema a un mayor nivel de abstracción, en lugar de simplemente en la codificación utilizando uno o más lenguajes de programación. Es habitual que los modelos se construyan utilizando lenguajes específicos de dominio (DSLs, por sus siglas en inglés). El MDE se ha utilizado con éxito en muchos ámbitos [6, 7], sin embargo la calidad y la facilidad de uso de las herramientas de modelado sigue siendo un factor limitante para una adopción más amplia [8].

Por otra parte, el aprendizaje automático (Machine Learning; ML) es un paradigma que intenta abordar problemas que son muy complicados de *programar a mano* (p.ej., análisis de sentimientos [9], reconocimiento de caras [10], traducción de textos [11], etc). En general, dado un problema y datos referentes a dicho problema el ML busca (o aprende) una función matemática que asocia el conjunto de entradas con el conjunto de salidas empleado algún algoritmo de aprendizaje. Dependiendo de la naturaleza de los datos, tenemos los siguientes tipos de aprendizaje [12]: supervisado (tenemos muestras de entradas con sus respectivas salidas), no supervisado (solo disponemos de las entradas) y por refuerzo (podemos obtener alguna medida de la calidad de una salida siguiendo la entrada asociada). Las técnicas de ML han sido aplicadas con éxito en varios campos de la Ingeniería del Software. En particular, es importante destacar los avances recientes de los entornos de desarrollo integrados (Integrated Development Environments; IDEs) para código gracias al ML. Los IDEs actuales incluyen funcionalidades que aumentan la productividad de un programador como, por ejemplo, generación de documentación, detección de defectos en código, o incluso generación de tests [13, 14].

Recientemente, las técnicas de ML han empezado a ser utilizadas para resolver tareas en el contexto de MDE. Por ejemplo, Nguyen et al. [15] emplea redes neuronales para clasificar meta-modelos en categorías de dominio; Osman et al. [16] extrae varias características de los modelos UML y los clasifica en *reverse-engineered* o *forward-engineered* empleando un enfoque supervisado. El agrupamiento no supervisado o clustering de modelos también ha sido estudiado en trabajos previos. Basciani et al. [17] tratan los modelos como si fueran documentos de texto y emplean clustering jerárquico para extraer grupos. Babur et al. [18] proponen usar la estructura de grafo de los modelos considerando $n-$grams para llevar a cabo el clustering. Finalmente, en

el contexto de aprendizaje por refuerzo, Barriga et al. [19] emplean técnicas de aprendizaje por refuerzo para reparar artefactos software.

A diferencia de los IDEs de código, las herramientas de modelado todavía no aprovechan los algoritmos de ML para mejorar su funcionalidad y hay aún varias cuestiones que tienen que explorarse para conseguir aplicar todo el potencial de ML en el dominio del modelado. Por ejemplo, un factor limitante encontrado en la literatura es la falta de conjuntos de datos (o datasets) de modelos. El no tener un buen dataset limita la generalización de los modelos ML e imposibilita la aplicación de técnicas de aprendizaje profundo (Deep Learning) ya que este tipo de maquinaria requiere grandes cantidades de muestras. En la literatura encontramos pocos datasets para MDE y, los que hay, son o muy pequeños [20] o no están curados (*curated*) [21, 22]. Por otro lado, es importante destacar que los modelos ML no pueden recibir como entrada un artefacto software en bruto, sino que es necesario transformar el mismo a una representación adecuada y legible por el modelo ML. Relacionado con la representación, han habido varias propuestas como kernels de grafos [23], modelos como documentos de texto [15, 17], o incluso modelos como árboles [24]. Sin embargo, aún no se ha estudiado sistemáticamente cuál es la mejor codificación de un modelo software para una tarea dada y un modelo ML. Finalmente cabe destacar que la cantidad de tareas MDE cubiertas por trabajos previos con técnicas de ML es pequeña y los enfoques empleados han sido principalmente simplistas, por lo que no existe un buen cuerpo de conocimiento práctico sobre cómo usar ML en MDE.

Esta tesis aborda los problemas anteriores e intenta cubrir la brecha actual entre las técnicas de ML y el MDE. El objetivo general de la tesis es explorar la aplicación de técnicas de ML para mejorar MDE y contribuir con artefactos prototípicos que la comunidad de modelado pueda usar para mejorar el estado del arte de ML aplicado a MDE. En particular, se pretende abordar los dos objetivos específicos siguientes:

Objetivo 1: Construir datasets de modelado para fomentar la aplicación de ML a MDE.

Se plantea construir tanto datasets etiquetados como no etiquetados para permitir tanto aprendizaje supervisado como no supervisado.

Objetivo 2: Aplicar algoritmos de ML para solucionar tareas MDE con el objetivo de mostrar el potencial del ML en el contexto del MDE.

En particular, en esta tesis se abordan dos tareas de MDE: clasificación de modelos (aprendizaje supervisado) y generación de modelos realistas (aprendizaje no supervisado).

En lo referente a la metodología llevada a cabo en esta tesis, es importante tener en cuenta que el actual estado del arte de ML aplicado a MDE no está bien establecido. Esto es esencialmente debido a la falta de datasets MDE y frameworks que permitan la aplicación de ML a MDE. Así pues, la metodología ha estado principalmente dirigida a suplir dichas carencias y después se ha puesto esfuerzo en construir aplicaciones complejas. Para cumplir los dos objetivos de la tesis, la metodología estuvo dividida en cuatro fases:

- **Recolección de modelos.** En esta fase se recolectaron modelos de diferentes repositorios tales como GitHub[1] o GenMyModel[2]. Entre los tipos de modelos que se recolectaron se incluyen Ecore, UML, RDS, Yakindu, etc.

[1] `https://github.com/`

[2] `https://www.genmymodel.com/`

- **Organización de los modelos.** Todos estos artefactos fueron analizados, almacenados e indexados en un buscador de modelos que ofrece varias facilidades de consulta y navegación. De esta forma, se obtuvo un dataset no etiquetado de modelos sobre el que se puede aplicar aprendizaje no supervisado.
- **Construcción de un dataset etiquetado.** En esta fase se consideró un subconjunto de los modelos Ecore y UML del buscador de modelos anterior y se etiquetaron. Para llevar a cabo dicha labor de etiquetado, se ideó una metodología que explota las facilidades de consulta del motor de búsqueda de modelos. Se consideraron varios tipos de etiquetas en el etiquetado: categoría, tags, propósito, notación, herramienta y diagrama principal.
- **Machine Learning aplicado a MDE (ML4MDE).** Finalmente, se utilizaron los datasets construidos en las fases anteriores para abordar dos tareas en el contexto de MDE usando ML. En particular, se han abordado la clasificación de modelos empleando un enfoque supervisado y la generación de modelos realistas usando un enfoque no supervisado.

Con las tres primeras fases, se cumple el primer objetivo (i.e., construcción de datasets curados) y, con la última fase se cumple el segundo objetivo (i.e., aplicación de ML a MDE). Como resultado de aplicar dicha metodología, esta tesis tiene cuatro contribuciones principales:

Un buscador de modelos. Para aplicar ML a MDE, primero necesitamos una cantidad importante de modelos. Así pues, el primer paso en esta tesis ha sido recolectar y organizar modelos para construir posteriormente datasets. Los repositorios de modelos son la forma común de guardar y organizar modelos. Sin embargo, los repositorios actuales, o bien no están disponibles (p.ej., MDEForge [25]), no ofrecen formatos de consultas específicos para modelos (p.ej., GenMyModel) o no soportan el rastreo y análisis de modelos. Motivados por estas necesidades, se construyó el buscador de modelos MAR [1, 2]. MAR es capaz de manejar e indexar cualquier tipo de modelo si su meta-modelo es conocido. Además, ofrece dos facilidades de consulta: búsqueda por palabras clave y búsqueda por ejemplos. En particular, en este último tipo de consulta, se le pide al usuario que introduzca un fragmento de modelo y el sistema recupera modelos que se parecen a dicho fragmento. Para considerar la estructura de los modelos a la hora de resolver la consulta, MAR emplea la noción de *bolsa de caminos*. Ésta es una codificación novedosa de modelos que se basa en la construcción de un multiconjunto de caminos entre elementos del modelo. Para manejar consultas de manera eficiente, se ideó un esquema de HBase para almacenar las bolsas de caminos. Además, MAR emplea Lucene como motor de búsqueda para resolver las consultas por palabras clave. Para que los usuarios puedan realizar consultas, se ha implementado una interfaz de usuario y un conjunto de servicios disponibles en `http://mar-search.org`. Finalmente, MAR tiene indexados más de 500.000 modelos de diferente tipo (Ecore, UML, BPMN, etc), los cuales están disponibles para descarga. Esto lo convierte, en este momento, en el dataset disponible más grande para aprendizaje no supervisado.

Un dataset etiquetado de modelos. Con el resultado anterior, tenemos a nuestra disposición miles de modelos que pueden ser usados para aplicar aprendizaje no supervisado. Así pues, para permitir la aplicación de aprendizaje supervisado, consideramos un subconjunto de los modelos de MAR y lo etiquetamos. Como resultado se presenta ModelSet [3]. Este dataset está compuesto de 5.466 modelos Ecore y 5.120 modelos UML etiquetados con su categoría

como etiqueta principal y además otras etiquetas secundarias de interés. Para llevar a cabo el proceso de etiquetado, ideamos una metodología llamada *Greedy Methodology for Fast Labelling* (GMFL). Dicha metodología está pensada para facilitar la exploración y etiquetado de datasets de modelos. Esta metodología fue implementada en un plugin Eclipse que puede ser usado para etiquetar datasets de modelos. Para mostrar la utilidad de ModelSet, se entrenaron varios modelos ML y se integraron en MAR para mejorar el *pipeline* de procesamiento de modelos. Finalmente, se ha construido una librería de Python [26] para permitir a los usuarios desarrollar aplicaciones ML utilizando Modelset. En particular, esta librería permite al usuario cargar ModelSet en Python y aplicar pipelines de ML estándar (p.ej., usando scikit-learn).

Clasificación de modelos. El problema de clasificación de modelos no es muy demandante desde el punto de vista de ML y se espera que la mayoría de modelos ML puedan obtener buenos resultados en dicha tarea. De hecho, hasta se podría decir que esta tarea está resuelta ya que trabajos previos obtienen puntuaciones casi perfectas en los conjuntos de prueba [15, 27]. Sin embargo, tras un estudio en profundidad de estos trabajos se observó que los resultados podrían estar sesgados. En primer lugar, el dataset objetivo empleado es relativamente pequeño y el problema de clasificación contiene pocas clases. En segundo lugar, la tarea abordada realmente es la de clasificación de meta-modelos y otros tipos de artefactos no han sido considerados. Y, en tercer lugar, no se ha tenido en cuenta la presencia de duplicados y casi-duplicados. Normalmente, los datasets cuyos modelos vienen de repositorios como GitHub pueden tener una cantidad no trivial de duplicados como ocurre en el caso de código [28]. Por otro lado, cuando se aplica ML a MDE, un obstáculo común a enfrentar es determinar la mejor combinación de modelo ML y codificación de artefactos. Este problema no ha sido todavía investigado sistemáticamente. Motivados por esta necesidad y el posible sesgo de trabajos previos, se llevó a cabo un estudio comparativo [4] de varias técnicas de ML y codificaciones de artefactos software en el contexto de la clasificación de modelos. Se empleó un dataset más grande que trabajos previos y se tuvo en cuenta el efecto de la duplicación. En particular, se utilizó como dataset objetivo ModelSet y la categoría como principal etiqueta a predecir. Entre los resultados obtenidos cabe destacar que: los mejores modelos fueron las redes neuronales y las máquinas de soporte vectorial, los vectores semánticos funcionan bien en UML pero no en Ecore, la estructura de los modelos no es relevante para el problema de clasificación y el rendimiento de los modelos ML empeora cuando los cuasi-duplicados son eliminados.

Generación de modelos realistas. En esta parte de la tesis, el objetivo ha sido emplear arquitecturas de aprendizaje profundo para resolver problemas complejos en el contexto del MDE. En particular, se ha abordado el problema de generar modelos realistas. Los generadores de modelos son herramientas que reciben como entrada unas especificaciones (escritas usando OCL [29] o patrones de grafos [30]) que describen cómo los modelos deben ser y se encargan de producir modelos que satisfacen dichas especificaciones. Varró [31] estableció cuatro características deseables que deberían cumplir todos los generadores: consistencia con respecto a las especificaciones, diversidad de formas en los modelos resultado, escalabilidad del generador con respecto al tamaño de los modelos y realismo de los modelos producidos. De esas propiedades, la propiedad del realismo no había sido matemáticamente definida. De hecho, las métricas propuestas para evaluar dicha característica son incorrectas o fácilmente trampleables [32]. Así pues, primero se definió matemáticamente la noción de realismo [32] ideando correspondencias entre generadores y distribuciones de probabilidad de modelos. Posteriormente se propuso un generador de modelos, llamado ModelMime (M2) [5], centrado en satisfacer la propiedad de realismo y se observó que no sólo obtiene resultados de vanguardia en la propiedad en la que fue

entrenado, sino que también obtiene resultados competitivos en las tres propiedades restantes (consistencia, diversidad y escalabilidad).

Esta tesis es resultado del compendio de varios artículos que describen los cuatro resultados previamente explicados, que han sido publicados en revistas y conferencias relevantes del área. Además de las publicaciones, cabe destacar que, como valor añadido, MAR es actualmente el mayor motor de búsqueda propuesto con más modelos indexados (aproximadamente medio millón de artefactos software); ModelSet es el mayor dataset etiquetado en el contexto de MDE; y ya hay otros investigadores que está empleando los modelos de MAR y ModelSet para construir aplicaciones interesantes [33–35]. Como trabajo futuro, se planea mantener MAR y ModelSet incorporando más tipos de modelos y rastreando más fuentes de modelos. En particular, se planea extender la metodología de etiquetado usando un enfoque colaborativo. Por otro lado, con respecto a las aplicaciones de ML a MDE, actualmente se está trabajando en un sistema de recomendación que emplea M2 como componente central para recomendar la siguiente operación de editado. Además, en lo referente a la generación de modelos, se pretende combinar M2 y VIATRA para alcanzar la propiedad de diversidad y consistencia mientras se mantiene realismo.

Abstract

Model-Driven Engineering (MDE) is a Software Engineering methodology which attempts to raise the level of abstraction of software development by promoting the use of models as first-class artifacts. In this way, models are no longer used only to document software, but they are also employed to describe, simulate, and generate code. This paradigm has been proven to be successful in a lot of scenarios where the complexity of the systems is not trivial and cannot be tackled by traditional software engineering practices. However, the quality and ease of use of the modelling tools still remain a limiting factor that prevents MDE from being used more extensively in practice.

On the other hand, Machine Learning (ML) is an Artificial Intelligence paradigm that has been applied to solve complex tasks in a wide variety of application domains. ML algorithms learn a mathematical function that maps a set of inputs to a set of outputs. This function is usually learnt from the data and, depending on the data's shape, these algorithms can be roughly classified into supervised, unsupervised, and reinforcement learning. ML has been successfully applied in the Software Engineering field. Particularly, it has been used to improve integrated development environments (IDEs) for code. This has been done by incorporating ML models trained on code auto-completion, documentation generation, defect detection, and test-case generation.

Unlike code IDEs, modelling tools have not been able yet to take advantage of ML techniques. One reason is the absence of high-quality and extensive curated datasets, either labelled or unlabelled, to train ML models. This aspect limits the generalization of such ML models and prevents the application of deep learning technologies. Moreover, ML models cannot receive as input a raw modelling artifact and it must be transformed into a representation suitable to feed the ML model (e.g., numeric vectors). There have been several proposals to address this, but this aspect has not been systematically studied for a given MDE task and ML model. Finally, the number of MDE tasks addressed with ML is scarce and the approaches used are simplistic.

This thesis tries to overcome the aforementioned issues by bridging the gap between ML and MDE. The first result of the thesis is the MAR search engine. This search engine collects a large amount of software artifacts of different types, analyses and stores them in a centralized database, providing query facilities to get relevant models. From an ML point of view, this result can be seen as an extensive unlabelled dataset that can be used for unsupervised learning. In addition, MAR is currently the model search engine with the largest collection of models. To address supervised learning scenarios, a set of UML and Ecore models were taken from MAR and labelled with their main category and secondary labels of interest. As a result, the ModelSet dataset was created, which is the largest corpus of labelled models in the context of ML for MDE. This dataset enables interesting applications of supervised learning to MDE. In particular, using ModelSet as the target dataset, a comparative study was carried out to compare different model encodings and ML algorithms in the context of model classification.

This study has showed that for model classification simple representations based on text (e.g., TF-IDF) and simple models (e.g., SVM or feed-forward neural networks) are good choices. Also, it shows semantics embeddings can also improve the performance of these models in the case of UML. Finally, deep learning architectures have been employed to solve a complex MDE problem: the generation of realistic models. In particular, a model generator based on autoregressive deep neural networks is proposed, which achieves state-of-the-art performance in terms of realism.

Chapter 1

Introduction

The design and implementation of current systems (e.g., cloud systems, cyber-physical systems, Internet of Things, etc) are becoming increasingly more difficult due to their complexity. In this context, raising the level of abstraction has always been a key strategy to address the inherent complexity of software development. In particular, the Model-Driven Engineering (MDE) paradigm proposes to use models as an active element in software development. In this paradigm, models are not only a way to document the software, but can also be used to describe, simulate, and generate code. The MDE approach relies on creating a model for each part of the system that represents a high-level view of that part. MDE solutions have been successfully employed in many scenarios [6, 7]. However, the quality and ease of use of modelling tools remain a limiting factor for wider adoption of MDE, making their improvement critical [8].

Machine Learning (ML) is an Artificial Intelligence paradigm that tries to solve problems which are *difficult to code by hand* (e.g., sentiment analysis [9], face recognition [10], text translation [11], etc) by learning from data about the problem/domain. Broadly speaking, given a problem and data associated with this problem, an ML algorithm learns a mathematical function that maps a set of inputs to a set of outputs. Depending on the data, the ML algorithms can be classified into three groups [12]: supervised learning (we have access to inputs and outputs), unsupervised learning (we only have inputs), and reinforcement learning (we have access to a quality metric that guides the learning process). ML has been successfully applied to Software Engineering. In particular, it is important to mention the recent advances in integrated development environments (IDEs) for code thanks to the ML field. Current IDEs include several features that increase the productivity of the programmer such as code auto-completion, documentation generation, defect detection, or test-case generation [13, 14].

Recently, ML has started to be applied to solve some MDE tasks. In the context of supervised learning, Nguyen et al. [15] use neural networks to classify meta-models into domain categories. Osman et al. [16] extract several features from UML models and classify them into reverse-engineered or forward-engineered by using a supervised approach. The clustering of software models has also been tackled in previous works. Basciani et al. [17] deal with models as if they were documents and implement hierarchical clustering using common document similarity measures. Babur et al. [18] propose to use the graph structure of the models by considering $n-$grams to perform clustering. Finally, in the context of reinforcement learning, Barriga et al. [19] rely on reinforcement learning techniques to automatically repair software models.

Despite these efforts, modelling tools cannot fully take advantage of the ML algorithms and

there are still several topics that have to be explored to unleash the full potential of ML in the modelling domain. For instance, one important issue identified in the literature is the lack of high-quality and extensive datasets, which severely limits the generalization the of ML models. There are few datasets for MDE, and they are either too small [20] or not curated [21, 22]. This issue makes it difficult to apply deep learning architectures to tackle MDE tasks (e.g., Graph Neural Networks [36] or Transformers [11]), as their machinery requires a large number of samples. Enabling the application of deep learning models by providing datasets could boost the application of deep learning to more complex MDE tasks beyond classification and clustering. Another important aspect is that ML models cannot receive a raw modelling artifact as input. That artifact must be transformed into a suitable representation, readable by the ML model. In this line, there have been several proposals such as graph kernels [23], models as documents [15, 17], or even models as trees [24]. However, it has not been systematically assessed what is the best encoding for a given MDE task and an ML model. Finally, the amount of MDE tasks addressed with ML techniques is still scarce and, in general, simplistic approaches have been used.

This thesis tackles the aforementioned issues and tries to bridge the gap between ML and MDE. In particular, the thesis contributes four main results:

- The first result is the MAR search engine. It is a search engine specifically designed for models that has processed, analysed, and stored hundreds of thousands of software artifacts. From an ML point of view, this platform provides several datasets for unsupervised learning.

- The second result is the ModelSet dataset. It is the largest labelled dataset in the context of MDE. It was built by labelling a subset of the models collected by MAR with several types of labels. This dataset enables interesting applications of supervised learning for MDE.

- The third result corresponds to a systematic comparative study in the context of model classification. The main aim was to study which is the best combination of ML model and artifact representation in order to address the model classification task.

- The fourth result of the thesis was to address the generation of realistic models using a deep learning perspective. The generation of realistic models is a complex MDE problem that has not been solved until now and was tackled using deep autoregressive models.

Chapter 2

Objectives and methodology

2.1 Objectives

The general objective of this thesis is to explore the application of ML techniques to enhance MDE approaches and to contribute prototypical artifacts which the modelling community could use to improve the state-of-the-art of ML applied to MDE. In particular, the two specific objectives of this thesis have been the following:

> **Objective 1**: Building datasets of software models to enable the development of ML applications in the context of MDE.

Particularly, the aim is to build labelled and non-labelled datasets of software artifacts to enable supervised and unsupervised ML applications.

> **Objective 2**: Applying ML algorithms to address MDE tasks to show the potential of ML in the context of MDE.

In this thesis, ML techniques have been applied to solve two MDE tasks: model classification (supervised task) and generation of realistic models (unsupervised task). These tasks have been implemented as reusable libraries (whose source code is available), which can be used directly or adapted to address similar tasks.

Figure 2.1 shows the relationship between the methodology, objectives, and results of this thesis. The upper part shows the objectives, which are used to drive the methodology (middle part of the figure). The lower part represents the obtained results after carrying out the methodology.

2.2 Methodology

The state-of-the-art in the context of ML for MDE is only starting to be established. This was mainly caused by the lack of MDE datasets and frameworks to enable the application of ML to MDE. Therefore, the methodology followed in this thesis was primarily oriented to tackle such shortcomings and then to build more complex applications. Thus, to fulfil the two objectives of this thesis, the methodology was divided into four phases:

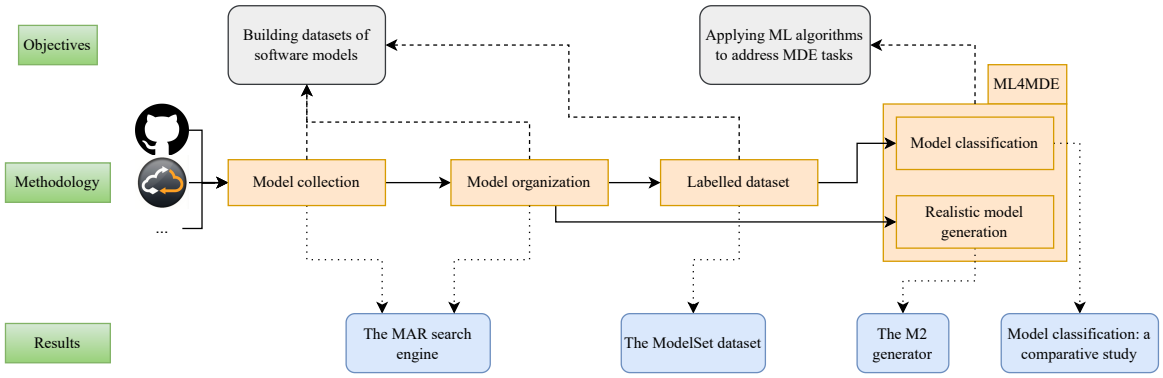


Figure 2.1: Objectives, methodology, and results.

- **Model collection.** In this phase, software models were collected from different sources by using dedicated crawlers. The main sources considered were GitHub[1] and GenMyModel[2] (a cloud modelling service). As target models, Ecore, UML, RDS, and Yakindu, to name a few, were crawled.
- **Organization of collected models.** All these artifacts were analysed and stored in a search engine with several query and browsing facilities. In this way, a (non-labelled) dataset that can be used for unsupervised learning was obtained.
- **Labelled dataset construction.** In this phase, a subset of the Ecore and UML models (collected in the previous phases) was labelled. To do so, a labelling methodology that exploits the query facilities of the search engine was devised.
- **Machine Learning applied to MDE (ML4MDE).** Finally, the datasets obtained in the previous phases were used to tackle two MDE tasks using ML approaches. In particular, the model classification problem with a supervised approach and the generation of realistic software artifacts with an unsupervised approach.

As shown in Fig. 2.1, the goal of the first three phases was to fulfil the first objective of the thesis (i.e., building and curating datasets) and the goal of the last phase was to fulfil the second objective (i.e., addressing MDE tasks with ML).

[1]https://github.com/
[2]https://www.genmymodel.com/

Chapter 3

Results

This chapter presents a summary of each of the four main results of the thesis, which are depicted in the lower part of Figure 2.1.

The first result is the MAR search engine, an efficient search engine for models. It stores all the crawled models in a centralised way and provides several query facilities. This result is derived from the collection and organization phases of the methodology.

Then, the collected models were labelled to build a labelled dataset of about $10,000$ models. This dataset, named ModelSet, is the second result of the thesis and it is the largest labelled dataset of software models to date.

Finally, ModelSet was used to perform a comparative study in the model classification task to assess several combinations of model encodings and ML algorithms, and a subset of the models available in MAR was used to build a realistic model generator.

3.1 Collecting and organizing models for unsupervised learning: the MAR search engine

In order to apply ML techniques to MDE, it is necessary to have available a large amount of software models. Thus, the first step in this thesis has been to collect and organize models which can be used to build such datasets. Model repositories are a common way to store and organize MDE models. However, current model repositories are not available (such as MDEForge [25]), do not have model-specific query mechanisms to find relevant models (such as GenMyModel), or do not support crawling and model analysis procedures which are key factors in building curated datasets. Motivated by this, the MAR search engine [1, 2] was created. It is a scalable search engine specially designed for models. MAR is able to handle and index any type of model if its meta-model is known. This search engine provides two query mechanisms: query-by-example and keyword-based queries. In the first one, the user is asked for a fragment of a software model and the system retrieves similar models. The structure of the model is considered using the notion of *bag of paths*, which is a novel encoding for models based on the construction of a multiset of paths between model elements. To handle queries efficiently, a special schema built on top of HBase was devised to store bags of paths. Regarding the keyword-based query, MAR relies on Lucene[1] to index the models using their identifiers.

[1] `https://lucene.apache.org/`

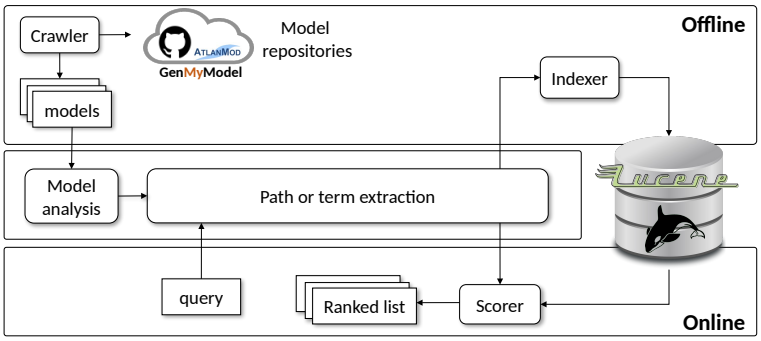


Figure 3.1: Architecture of MAR. Extracted and adapted from [1],

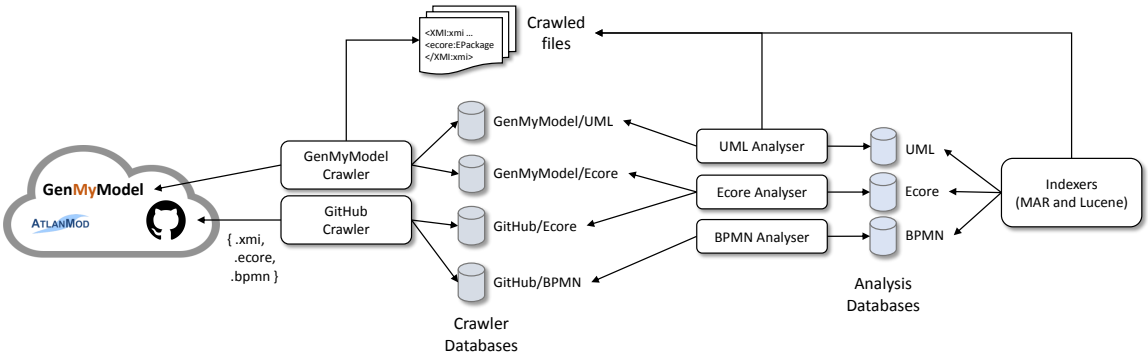


Figure 3.2: Crawling and analysing pipeline. Extracted from [2].

The architecture of MAR

The architecture of MAR is depicted in Fig. 3.1 and has two main processes: off-line and on-line. In the former, MAR uses dedicated crawlers to discover software models stored in different data sources. These models are analysed and indexed in two indices enabling the two types of query mechanisms: by-example and keyword-based. In the later, the search engine takes the user's query (keywords or software model), MAR accesses the corresponding index, and retrieves a ranked list of models.

Indexed models

An added value of this work is that MAR is currently the only model search engine available that contains hundreds of thousands of artefacts. Previous model search engines stored at most a few thousand of models, and such models were never made available. MAR currently contains $\sim$ 500k models (see Table 3.1) of different types, including Ecore, UML, BPMN, and Archimate, to name a few. The main two sources of models were GitHub and GenMyModel. Once these models are retrieved from such sources, they are analysed (meta-data extraction, duplication detection, statistics, validation, etc) and stored in the indices. Figure 3.2 shows this pipeline.

Encoding software models as bags of paths

One of the main contributions of MAR is the notion of *bag of paths* to encode models. This encoding allows query-by-example to be performed and return relevant software models based

	Source	Crawled	Duplicates	Failed	Indexed
Ecore	GitHub	67,322	46,199	341	20,782
	GenMyModel	3,987	3	27	3,957
	AtlanMod	304	1	4	299
UML	GitHub	53,082	7,282	1,699	44,101
	GenMyModel	352,216	143	23,836	328,237
BPMN	GenMyModel	21,285	0	200	21,085
Archimate	GitHub	496	77	106	313
PNML	GitHub	3,291	1,576	1,044	671
Sculptor	GitHub	188	88	0	88
RDS	GenMyModel	91,411	108	515	90,788
Simulink	Dataset [37]	200	0	0	200
Total	-	593,582	55,477	27,972	510,321

Table 3.1: Summary of the type of models crawled by MAR. Extracted from [2].

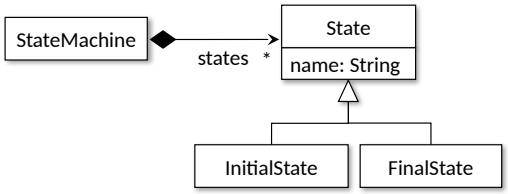


Figure 3.3: Example of Ecore query. Extracted from [2].

on the similarity of paths between the indexed models and the user query. To illustrate this, let us consider the Ecore query model represented in Fig. 3.3. This query means that the user is interested in Ecore meta-models representing a state machine, but those meta-models in which the different kinds of states are modelled as subclasses will be ranked first. The query and the models in the repository are transformed into multigraphs. The multigraph associated with the running example query is shown in Fig. 3.4. In this graph, two types of nodes can be distinguished: attribute values (circled nodes) and "object class" (rounded rectangles). The edges are labelled using the references and attributes names.

Once the graph is computed, the paths are extracted generating a multiset named *bag of paths*. However, extracting all paths from the graph is not feasible. Therefore, by default, MAR extracts three types of paths: paths of length zero for objects without attributes (e.g., $\left(\text{EPackage}\right)$), paths of length one which start from attribute values (e.g., $\left(\text{states}, \overrightarrow{\text{name}}, \text{EReference}\right)$), and simple paths with lengths less than or equal to a threshold (normally 3 or 4) between attributes and between attributes and objects without attributes (e.g., $\left(\text{StateMachine}, \overrightarrow{\text{name}}, \text{EClass}, \overrightarrow{\text{eStructuralFeatures}}, \text{EReference}, \overrightarrow{\text{name}}, \text{states}\right)$).

To perform the search, given the bags of paths from the repository and the bag of paths associated with the query, an adaptation of Okapi BM25 [38, 39] (which is a ranking function commonly used in document retrieval) is used to compute the ranking scores. Each model of the repository is compared with the query model and a ranked list of models is returned.

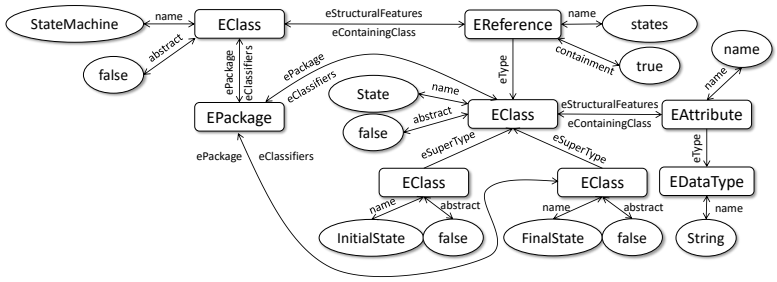


Figure 3.4: Multigraph associated with the Ecore query in Fig. 3.3. Extracted from [2].

HBase database schema for efficient model retrieval

Given a query, computing its associated ranked list of results using a naive implementation of the Okapi BM25 scoring function would be too slow. To speed up the search process, a special inverted index built on top of HBase has been devised. The associated HBase schema is another contribution of this work and it is illustrated in Fig. 3.5. Broadly speaking, the HBase data model consists of tables. One cell (or value) in a table is identified by a row key, a column family, a column qualifier, and a timestamp. Therefore, given a table, a particular cell has four dimensions:

$$(\text{Row, Column Family, Column Qualifier, Timestamp}) \longrightarrow \text{Value}$$

In the proposed schema, each cell of the inverted index has one path associated. To get the row key and the column qualifier, each different path is split into two parts: the prefix and the rest (the split point depends on the type of path). The row key will be set as the prefix and the column qualifier will be the rest. The value of each cell is a serialized map that contains information needed to compute the ranking score when there is a path match (such as the identifiers of the models that contain that path). Finally, the timestamp dimension is used to support incremental indexing. That is, to enable the option of introducing more models to the inverted index. Altogether, the schema has the following form:

$$(\text{Prefix, Column Family, Rest, Timestamp}) \longrightarrow \text{Serialized map}$$

This schema has two main advantages: 1) the number of accesses to the database per query is reduced as the database is queried per different prefixes rather than different paths, and 2) the size of the stored inverted index is also reduced.

Evaluation of MAR

MAR is evaluated from three different angles: search precision, query response time, and indexing time.

Search precision and query response time. To carry out the first two evaluation procedures, a dataset of pairs (query, relevant model) is considered. Each pair is computed by taking a model from a crawled dataset of Ecore models and mutating it to simulate a query made by a user that is interested in that model. To measure the search precision, the Mean Reciprocal Rank (MRR) is used. This metric is calculated as $\frac{1}{r}$ where r is the rank position of the relevant model. Table 3.2 shows the results of the search precision evaluation. In this

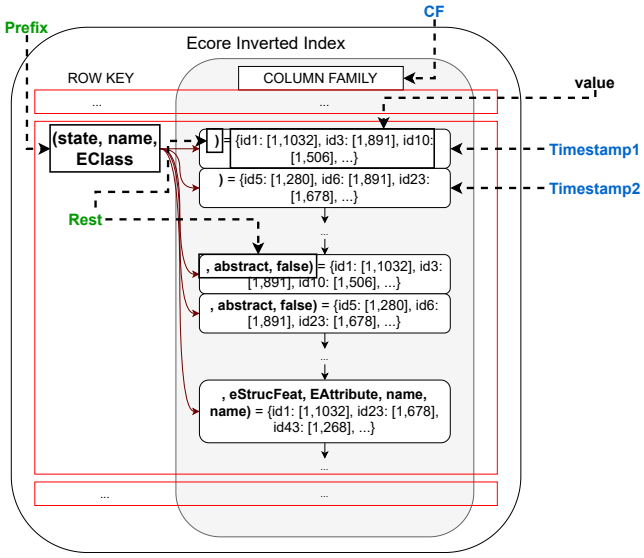


Figure 3.5: HBase schema extracted from [2].

analysis, six different configurations of MAR are considered. The motivation is to study the effect of considering more attributes and more model structure in the search. Three of the configurations are called *Names-X* and only consider the names inside the model. The other three, called *All-X*, consider more attributes such as cardinalities, whether a class is abstract or not, etc. The X attached to the configurations' name is an integer that indicates the maximum length of the paths considered and controls the amount of structure that is taken into account when performing the search. In the experiments, $X \in \{2, 3, 4\}$. Moreover, the keyword-based search is used as a baseline to compare MAR and it can be observed that all configurations of MAR outperform the keyword-based search. Finally, to assess the query response time, the seconds that each query takes to be satisfied is annotated. Table 3.3 shows the results of the query response time evaluation. The queries are split in three groups depending on their sizes (small, medium, and large). With the most accurate configuration (i.e., Names-4), MAR is able to handle large queries in less than a second on average.

Indexing time. To assess this aspect of the MAR search engine, a large corpus of 17k crawled Ecore models is considered. MAR is able to index such a large dataset in 12 minutes. Furthermore, this process grows linearly with respect to the number of models to be indexed. The indexing procedure is fully parallelizable as it is implemented using Apache Spark [2].

3.2 Labelling models for supervised learning: the ModelSet dataset

The MAR search engine provides thousands of models which can be used for unsupervised learning approaches. In order to enable the application of supervised learning in the MDE field, a subset of the MAR repository was taken and labelled. As a result, the ModelSet [3] dataset was obtained.

ModelSet is composed of 5,466 Ecore meta-models and 5,120 UML models labelled with its category as the main label plus additional secondary labels of interest. To carry out the

[2] `https://spark.apache.org/`.

Configurations	MRR	Differences in MRR
Text search	0.668	-
Names-2	0.734	p-value < 0.001 / $+0.066$
Names-3	0.742	p-value < 0.001 / $+0.074$
Names-4	**0.757**	p-value < 0.001 / $+0.089$
All-2	0.742	p-value < 0.001 / $+0.089$
All-3	0.752	p-value < 0.001 / $+0.084$
All-4	0.702	p-value < 0.001 / $+0.034$

Table 3.2: Mean Reciprocal Rank (MRR) of key-word based search of all configurations of MAR. Extracted from [2].

	Small		Medium		Large	
Configurations	Mean	Max	Mean	Max	Mean	Max
All-2	0.13	0.25	0.24	0.51	0.35	1.82
All-3	0.29	1.02	0.66	2.07	0.97	3.63
All-4	0.98	2.75	2.36	6.60	3.47	9.18
Names-2	0.03	0.09	0.06	0.16	0.09	1.08
Names-3	0.05	0.21	0.11	0.41	0.19	1.64
Names-4	0.08	0.41	0.28	1.41	0.66	3.85

Table 3.3: Query response time in seconds. Extracted from [2].

labelling process in an effective manner, a Greedy Methodology for Fast Labelling (GMFL) was devised. This methodology was designed to facilitate the exploration and labelling process of datasets of models. An implementation of the GMFL algorithm has been released through an Eclipse plugin which can be used to label datasets of models. To show the usefulness of ModelSet, several simple classifiers have been trained to label models gathered by the MAR search engine. Moreover, the ML models have been integrated as part of the user interface of MAR to support faceted search. Finally, to enable users to develop ML applications using ModelSet, a Python library [26] has been released. This library allows the user to load ModelSet in Python and to seamlessly apply standard ML pipelines (e.g., with scikit-learn) using models as training data.

Labelling methodology

Labelling software models is a very time-consuming activity due to the nature of the input data. The reason is that, to label a single model, the person in charge of labelling needs domain expertise in order to understand it and propose a suitable label. Many times, this implies looking for information about the model in sources like the origin of the model or directly on the Internet.

To address this shortcoming and make the labelling process more efficient, the methodology *Greedy Methodology for Fast Labelling* (GMFL) has been proposed. Its algorithm performs a user-driven clustering and is shown in Algorithm 1. Let us consider a dataset of models $\mathcal{M} = \{m_1, \ldots, m_t\}$. The aim is to label such dataset i.e., assign each model a label. Thus, at the end of the labelling process, two sets are obtained: a set of labels $\mathcal{L} \neq \emptyset$ and a set of tuples (i.e., the original dataset $\mathcal{M}$ labelled) $\mathcal{T} = \{(m_1, l_1), \ldots, (m_t, l_t)\}$ where $l_i \in \mathcal{L}$ for all $1 \leq i \leq t$. The philosophy of GMFL is to maximize the sizes of the *labelling streaks*, that is, to maximize the number of models with the same label assigned in a row. In Algorithm 1, three parts can be customized (marked as comments in the algorithm) depending on the type of the target models and the available technology:

- Exploration order. It sets the order in which the user will be presented with new unlabelled models. To build ModelSet, those models that have a lot of similar models (as a notion of similarity, the TD-IDF approach was used together with the cosine similarity) are first explored.
- Similar model retrieval. It establishes how the set $\mathcal{F}$ of models similar to the currently explored model m is chosen. The expectation is that the models in $\mathcal{F}$ can be labelled with the same label as m. To build ModelSet, the MAR search engine has been used to retrieve similar models.
- Model retrieval refinement. This step aims to maintain the labelling streak by adding more models that can be labelled with the same label as the current streak. To build ModelSet, the already labelled models are used as queries to the search engine and these results are added to a queue of models to be labelled.

This labelling methodology can be seen as a DBSCAN clustering approach but interactive and without the need of setting epsilon and min points. The main advantage of using GMFL rather than other clustering methods is that the user does not need to set hyperparameters (e.g., number of clusters) beforehand. Also, one disadvantage of traditional clustering techniques is

Algorithm 1 Sketch of GMFL extracted from [3]. $\mathcal{V}$ represents the unvisited models.

Data: $\mathcal{M}$: Models in the dataset
Result: $\mathcal{T}$: set of labelled models along their labels
$\mathcal{T} \leftarrow \emptyset$
while $\mathcal{M}$ *has unlabelled models* **do**
  `// Exploration order`
  $m \leftarrow$ pick unlabelled model from $\mathcal{M}$

  Manually inspect m to assign label l
  $\mathcal{T} \leftarrow \mathcal{T} \cup \{(m, l)\}$
  $\mathcal{V} \leftarrow \{m\}$
  while *not isEmpty($\mathcal{V}$)* **do**
    $m \leftarrow pop(\mathcal{V})$
      `// Similar model retrieval`
    $\mathcal{F} \leftarrow$ search for non-labelled models $m_1, ..., m_n$ sorted by similarity to m;

    Manually inspect $m_1, ..., m_n$ to assign label l
    $\mathcal{A} = \{(m_1, l), ..., (m_n, l)\}$
    $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{A}$

    `// Model retrieval refinement`
    m' $\leftarrow$ pick relevant models from $\mathcal{A}$
    add m' to $\mathcal{V}$
  end
end

that they require splitting the data into groups up front. In other words, one needs to know in advance the unrealistic number of labels.

The ModelSet dataset

ModelSet was built by applying the GMFL methodology to a subset of the Ecore and UML models available in MAR. During the labelling process, the following types of labels were considered:

- Category. This is the main label and indicates the main application domain. Fig. 3.6 shows the top 15 categories of ModelSet in the case of Ecore and UML. As can be observed, the categories of the Ecore models represent mainly technical and specific domains, whereas, in the case of UML, the categories represent non-technical domains.

- Tags. A set of keywords that characterizes the model and often specializes the value of the category. For instance, in UML, a model categorized as *computer-videogames* may include the tag *poker* to reflect the type of game.

- Purpose. This label only applies to Ecore models and indicates the intended usage of the model (e.g., assignment, for models used in teaching; or benchmark, for models specifically created for benchmarking).

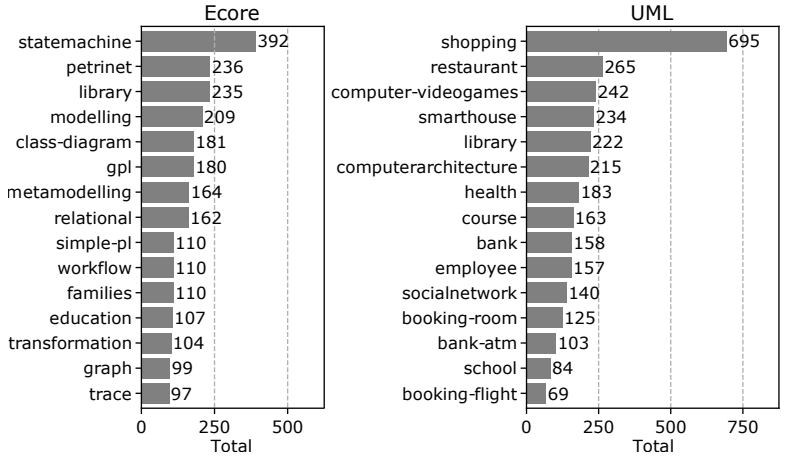


Figure 3.6: Top 15 categories of Ecore and UML models in ModelSet. Extracted from [3].

- Notation. This label only applies to Ecore models and specifies whether there is an associated concrete syntax and the tool used to create it (e.g., xtext or sirius).
- Tool. This label only applies to Ecore models and indicates whether the meta-model is part of a tool.
- Main-diagram. This label only applies to the UML models and indicates which diagrams were used to derive the category of the model.

Usefulness of ModelSet

To show the usefulness of ModelSet, several classifiers were trained to improve the processing pipeline of MAR and its browsing facilities. Fig. 3.7 shows three new steps added to the MAR pipeline.

- **Dummy model identification (ⓐ).** This is the first step in the new pipeline. A *dummy* model is a model labelled with the category *dummy*, whose intent is to identify models containing mostly mock data and created for testing purposes. Thus, they should not be indexed in the search engine as they are not relevant for any query. The dummy model identification is tackled as a binary classification problem. To perform the classification, several relevant features from the training models are extracted, namely counts of the number of elements, median of the characters in string attributes, and count of dummy names. Considering several ML models and the one that yielded, the best results were C5.0 with test set F_1 scores of 0.79 and 0.96 in ModelSet-Ecore and ModelSet-UML respectively.
- **Category identification (ⓑ).** Given a new crawled model, the aim is to assign it a category. This problem is faced as a multi-class classification. To perform the classification, the input features were encoded by a TF-IDF approach using the identifiers of the models, and three ML models ($k-$NN, neural network, and SVM) were considered. The best models were the neural network and SVM achieving accuracies close to 0.95 in the test set. However, as we will show later, these almost perfect results were due to the presence of quasi-duplicates.

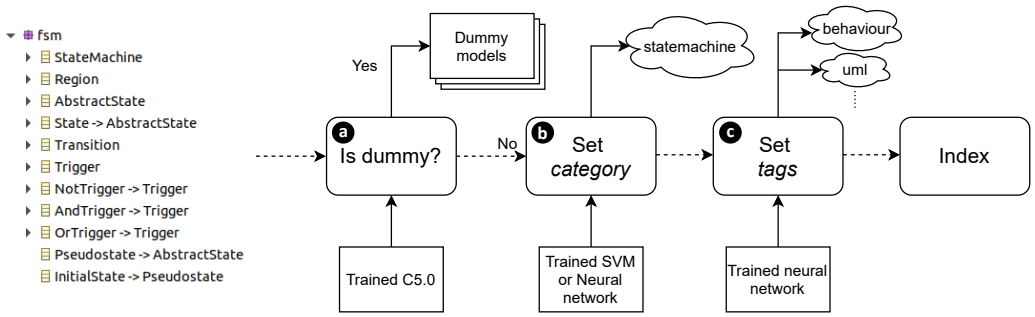


Figure 3.7: MAR's pipeline enriched with several classifiers trained with ModelSet. Extracted from [3].

- **Tagging (c).** Given a new crawled model, the aim is to assign it a set of tags. This problem is faced as a multi-label and multi-classification problem. To perform such classification, a two-layer neural network has been devised (Fig. 3.8). This network receives as input a TF-IDF representation of the model and, given a tag, it outputs the probability of labelling the model with that tag. This neural model achieves F_1 scores of 0.86 and 0.91 in ModelSet-Ecore and ModelSet-UML respectively.

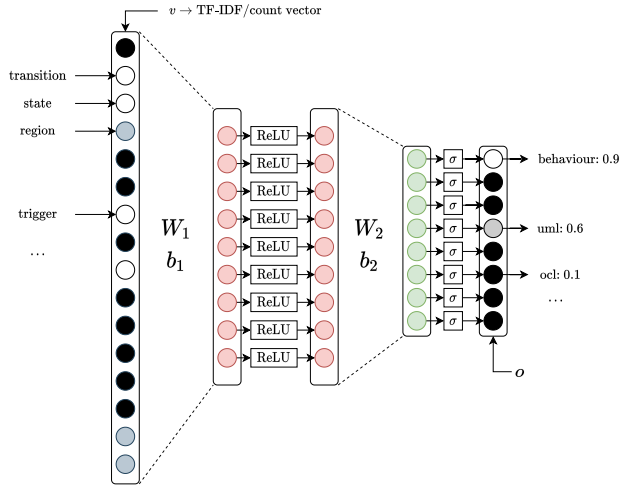


Figure 3.8: Architecture of the neural network for tag inference. Extracted from [3].

Finally, Fig. 3.9 shows how the previously trained ML models have been integrated into MAR. First, the identified dummy models are not indexed. Thus, they are not present in the results panel. Each model of the ranked list of results is labelled with the main category (a) and a set of tags (b) (some of them are inferred by the tagging model and others come from the GitHub repository). Finally, the user can filter the results according to the category and the tags (c).

It is worth noting that the application of an ML algorithm to this problem has allowed us to tackle the problem of classifying models at scale. The huge number of models indexed by MAR makes it impossible to manually label each model in order to show appropriate labels in the search results. In fact, to the best of our knowledge, this is the first integration of an ML model in a public MDE tool.

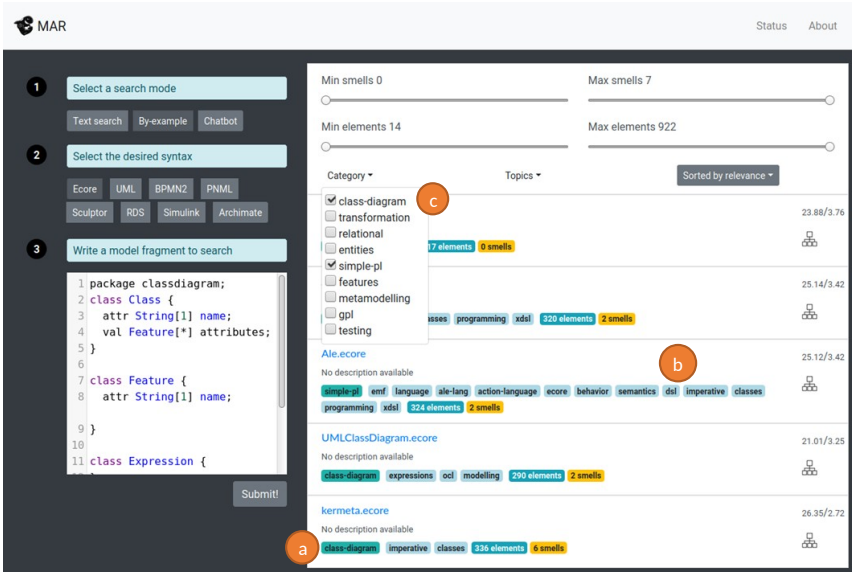


Figure 3.9: Screenshot of MAR. Extracted and adapted from [3].

3.3 Model classification

The model classification task is not very demanding from the ML perspective, and it is expected that most ML models can achieve good performance on it. In fact, according to previous works [15, 27], this task could be considered solved because they have obtained almost perfect scores over the test set. However, it is important to note that these works could be biased and thus report inconclusive results. Firstly, the target dataset is relatively small, and it contains a low number of target classes. Secondly, the actual task that is tackled is only a meta-model classification task and other types of models have not been considered. And thirdly, datasets of models suffer from the presence of duplicates or quasi-duplicates as it occurs in the case of code [28]. On the other hand, when applying ML to MDE, a common obstacle is determining which is the best combination of artifact representation and ML model. This issue has not been investigated in depth. Motivated by this issue and the possible bias of previous works, a study was carried out to systematically compare several ML techniques (such as feed-forward neural networks, graph neural networks, k-nearest neighbours, etc) and model encodings (such as TF-IDF, word embeddings, graphs, etc) in the context of model classification [4] using a larger dataset (ModelSet, with the category as the target label) and taking into account the quasi-duplication of the models in the dataset.

ML models and encodings considered

The combination of ML models and encoding schemes is shown in Table 3.4. The combination choices were driven by previous works in the context of ML for MDE and the compatibility between the ML model and the encodings. Essentially, there are two main families of representations: Bag of Words and graphs. The former includes TF-IDF, 2D TF-IDF, and word embeddings. This family encodes just the terms of the model and does not take into account the structure of the input models. The latter includes graph kernels and Bag of Paths, which not only consider the terms but also the relations between them (i.e. the structure of the model).

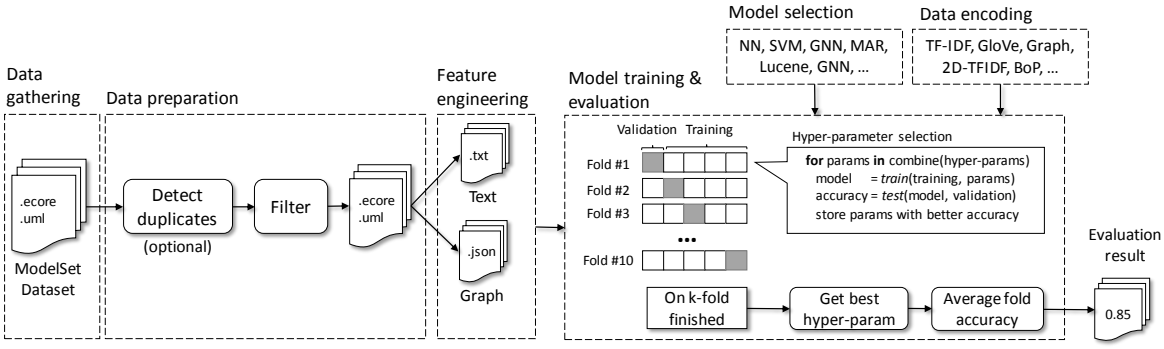


Figure 3.10: Overview of the methodology. Extracted from [4].

Methodology

To systematically compare the different approaches for encoding software models and the associated ML models (see Table 3.4), a methodology which is shown in Fig. 3.10 has been devised. Each phase of the methodology is described below.

- Data gathering. ModelSet has been used as the target dataset because it is the largest state-of-the-art labelled dataset of models. In particular, the addressed problem is the inference of the category label (multi-class classification) for the Ecore and UML models.
- Data preparation. The original ModelSet dataset is pre-processed in the following way. Firstly, the quasi-duplicates are removed and a ModelSet version without quasi-duplicates is built. This is done to study the effect of duplication on the performance of the ML models. ModelSet has a proportion of duplicates as it is composed of models that come from GitHub and GenMyModel. Developers of such platforms tend to *copy-and-paste* instead of making models from scratch. Finally, the categories that have a low number of models are filtered.
- Feature engineering. Two different types of features are generated: textual features and graphs.
- ML model selection. In this work, six ML models are considered (first column of Table 3.4).
- Data encoding. In this phase, the features are mapped to the corresponding encoding. In this work, seven data encoding techniques are considered (second column of Table 3.4).
- Model training and evaluation. To compare each combination of ML model and encoding, $k-$fold cross-validation is employed (as it is done in previous works [15, 40]). As an evaluation metric, the balanced accuracy which is defined as the average recall of each category is used. ModelSet is highly imbalanced and this metric works better in this type of dataset as it gives the same importance to each class.

Research questions

The research questions considered in the comparative study were the following:

- RQ1: Which model achieves a greater performance in the task of model classification?

  This question aims at identifying which ML model is the best choice to tackle the model classification task.

- RQ2: How the chosen encoding of software models affects the task of model classification?

  This question tries to complement RQ1 concerning the importance of the encoding selection.

- RQ3: What is the effect of data duplication on the performance of the ML models?

  In other words, is the performance of the ML model biased due to the presence of duplicates?

To summarize, these three questions try to shed light on which are the best encoding and ML choices and see whether the presence of quasi-duplicates could bias the evaluation.

Discussion

The main conclusions of the comparative study were the following.

- Regarding the first RQ, simple FFNN and SVM are the best ML models as they achieve the best performances.

- Regarding the second RQ, it was found that the structure of the software artefacts is not relevant in this task, since the best results are obtained with encodings which do not use the structure of the model. Related to this RQ, it was also found that word embeddings work well in UML but not in Ecore. This is because the terms used in Ecore are more specific and UML is normally used to model non-technical domains (e.g., banks, shopping centers, etc).

- Finally, regarding the third RQ, it is concluded that the performance of all ML models is reduced when quasi-duplicate models are removed. Something similar occurs with code [28]. One important takeaway here is that, when dealing with a dataset of models, the presence of duplicates should be considered as it can bias the results.

3.4 Generation of realistic software models

This part of the thesis is aimed at using deep learning architectures to solve a complex MDE problem beyond classification and clustering. The goal is to demonstrate the potential of deep learning in the MDE setting. In particular, the problem that is tackled is the generation of realistic models.

Model generation tools or model generators produce software models that follow a set of specifications described by the user. These specifications have normally the form of a metamodel (that the output models have to conform to), constraints (defined as OCL [29] or graph patterns [30]), scope (such as the number of output objects), etc. Previous work [31] established four properties that a given model generator should satisfy: *consistency* with the input specifications, *diversity* of shapes within the produced models, *scalability* concerning the output models size, and *realism* of the output models, that is, whether the produced models are

ML model	Encoding	Impl.	Inspired by/ Adapted from
FFNN	BoW TF-IDF	scikit	[3, 15]
	BoW word embeddings	scikit and gensim	[41]
SVM	BoW TF-IDF	scikit	[3]
	BoW word embeddings	scikit and gensim	[41]
	Graph kernel	scikit and GraKeL	[23, 42]
$k-$NN	BoW TF-IDF	scikit	[3]
	BoW word embeddings	scikit and gensim	[41]
	Raw BoW	Lucene	[43]
	BoP	MAR	[1, 2]
Naive Bayes models	BoW TF-IDF	scikit	[44]
GNN	Raw graph	PyTorch	[32]
CNN	BoW 2D TF-IDF	Keras and TensorFlow	[27]

Table 3.4: Combination of ML models and input features considered in the comparative study. Extracted from [4].

indistinguishable from the human ones. Out of these four properties, the realistic property has been identified as the one that has not been properly and mathematically defined. Moreover, no existing generator was able to generate realistic models.

The state-of-the-art metrics proposed to measure this property are incorrect or easy to cheat on [32]. Therefore, the first step has been to mathematically define what a realistic generator is [32]. Then, on top of this definition, a concrete model generator able to generate realistic models has been designed and implemented [5]. Actually, the generator is intended to address the *structurally realistic property* i.e., generate realistic models but only looking at the typed graph structure and ignoring the attributes. The problem of generating realistic attribute values is orthogonal to generating a realistic typed graph structure [45].

Characterization of the realistic property

In the proposed approach, a key observation to characterize the notion of realism in generators is that a generator can be seen as a probability distribution over models. More formally, let us assume that we have access to a dataset of real models $\{x_1, \ldots, x_n\} \sim Q$ where Q represents the realistic distribution of models. That is, Q represents the distribution of the models made by random humans. Given a generator g, it can be seen as a probability distribution over models P_g and use it to generate $\{y_1, \ldots, y_n\}$. Thus, we have all the ingredients to consider the following hypothesis test

$$H_0 : P_g = Q$$

$$H_1 : P_g \neq Q$$

In this setting, a generator is realistic if H_0 can be accepted. In practice, one is normally interested in comparing two generators and determining which one is the best in terms of realism. To this end, one can employ the p-value or the main statistic of the contrast. Two ways for solving this hypothesis test are proposed:

- C2ST+GNN [32]. This approach uses the Classifier Two Sample Test (C2ST) [46] together with a Graph Neural Network (GNN) [36]. The idea is that if a GNN can be trained to

distinguish between the samples that come from P_g and the ones that come from Q, and it performs well, then the generator is not realistic. More formally, the following set is built:

$$\mathcal{D} = \{(x_i, 0)\}_{i=1}^{n} \cup \{(y_i, 1)\}_{i=1}^{n} = \{(z_i, l_i)\}_{i=1}^{2n}.$$

This set is split into a training set and a test set. The GNN is trained with the train set and is evaluated using the test set. The statistic of the contrast is the accuracy over the test set. The closer to 0.5, the more realistic the generator is. If the accuracy is high, then the generator is not realistic as the GNN is able to distinguish well.

- Maximum Mean Discrepancy [47] over graph statistics [5]. The idea is to relax the hypothesis test by considering graphs distributions instead of the full graphs. From $\{x_1, \ldots, x_n\}$ and $\{y_1, \ldots, y_n\}$, graph distributions are extracted generating $\{g_1, \ldots, g_n\} \sim \boldsymbol{g}$ and $\{g'_1, \ldots, g'_m\} \sim \boldsymbol{g'}$. Each g_i, g'_j are real graph distributions. It is wanted to compare the distributions $\boldsymbol{g}$ and $\boldsymbol{g'}$. To this end, the statistic MMD^2 is computed:

$$MMD^2(\boldsymbol{g}||\boldsymbol{g'}) = E_{g_1,g_2\sim\boldsymbol{g}}\left[k(g_1, g_2)\right] + E_{g'_1,g'_2\sim\boldsymbol{g'}}\left[k(g'_1, g'_2)\right] -$$

$$-2E_{g,g'\sim\boldsymbol{g},\boldsymbol{g'}}\left[k(g, g')\right].$$

Here, k is a kernel that compares univariate distributions. The kernel function is used based on the Wasserstein distance proposed in [48]. Regarding the univariate graph distributions, representative metrics that have been used in previous works to assess the realistic property of model generators [31, 45, 49] are considered: Multiplex Participation Coefficient (MPC), Normalized Node Activity (NNA), and Degree Distribution (DD).

The M2 generator

Once the realistic property is properly characterized, a realistic model generator, named ModelMime (M2), was presented [5]. M2 consists of an autoregressive neural network which is trained to fulfil the realistic property. A key idea behind this work is that software models can be seen as sequences of edit operations, and thus, M2 is trained to predict the next edit operation given the previous ones. Therefore, after the training procedure, this generator can produce software models by just concatenating and predicting edit operations iteratively. M2 has been evaluated in different scenarios (using models that come from MAR) and compared against three state-of-the-art model generators. It is concluded that M2 outperforms the others in terms of realism and that M2 is competitive in the rest of the properties (i.e., diversity, consistency, and scalability).

The main ingredients of M2 are the decomposition of models as sequences of edit operations, the characterization of the optimization problem that the neural network solves, and the architecture of the neural model.

Edit operations

The M2 generator is based on an autoregressive model, that is, the generator builds a new artifact by concatenating construction steps. In this case, the main construction step relies on the notion of *edit operation*. Therefore, the generator will produce models by chaining edit operations. For this problem, an edit operation is defined as an operation that a modeller may perform over an incomplete or partial model that only adds new elements to the model (in the

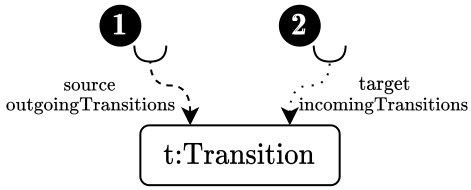


Figure 3.11: *Add transition* edit operation. Extracted from [5].

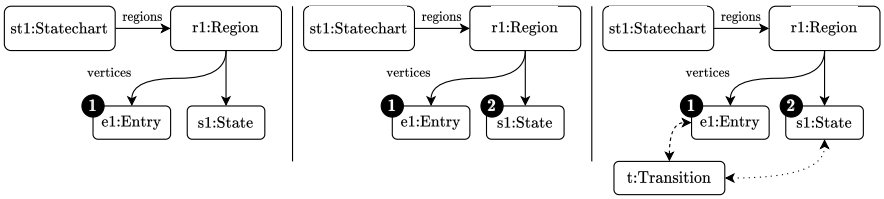


Figure 3.12: Application of *Add transition* on a partial model. Extracted from [5].

form of new nodes and edges in its associated typed graph). An edit operation is defined by a tuple of three elements:

- Unique identifier. It identifies univocally the operation.
- Edit graph. Elements that will be added to the partial model.
- Edit connections. Edges in the edit graph that will be connected to the model under construction. For each edit connection, a node in the partial model has to be selected to link the edit graph.

For instance, the edit operation in the Fig. 3.11 represents the action of adding one transition between two states in a statechart. In this edit operation, the semi-circles (i.e., ◟) represent the edit connections. Fig. 3.12 shows how this edit operation is applied to a partial model. Firstly, the source and target nodes are selected and then the edit graph is added to the model. To train the neural network and perform the decompositions, a set of edit operations have to be defined beforehand. This set can be derived automatically by using the meta-model. The edit operations that are derived automatically are called atomic edit operations. However, this type of edit operations are sometimes too simple and put a lot of pressure on the neural model. Therefore, it is recommendable that the user adds more complex edit operations by combining the previously generated atomic operations.

Optimization problem

Our generator assumes that the user has access to a dataset of real models $\{x_1, \ldots, x_n\} \sim Q$, from which the neural model learns a distribution over models. In other words, it is learnt $P_\theta(x) = P(x|\theta)$[3] to be close to the real distribution Q. To achieve so, the neural network is trained using the maximum likelihood estimation i.e., it is trained to solve the following optimization problem:

$$\hat{\theta} = \text{argmax}_\theta \Pi_{j=1}^{n} P_\theta(x_j) = \text{argmax}_\theta \sum_{j=1}^{n} \log P_\theta(x_j) = \text{argmin}_\theta \sum_{j=1}^{n} -\log P_\theta(x_j).$$

[3] θ represents the parameters (weights) of the neural network

This neural network tackles the model generation as a sequential process. So, given a model x and a possible decomposition of x (named d), the probability can be expressed as

$$P_\theta(x, d) = P_\theta(x_d^1, \ldots, x_d^T) = \Pi_t P_\theta(x_d^{t+1} | x_d^{\leq t})$$

In each step, a partial model is extended by applying an edit operation:

$$x_d^{t+1} = e_{t,d}(x_d^t) \text{ for all } t = 1, \ldots, T-1$$

where $e_{t,d}$ is an edit operation. Therefore, the neural model samples an edit operation $e_{t,d} \sim P_\theta(e_{t,d} | x_d^t, \ldots, x_d^1)$, and then $e_{t,d}$ is applied to x_d^t to get x_d^{t+1}. It is also accepted the Markov assumption, so $P_\theta(e_{t,d} | x_d^t, \ldots, x_d^1) = P_\theta(e_{t,d} | x_d^t)$. Thus,

$$P_\theta(x, d) = \Pi_t P_\theta(x_d^{t+1} | x_d^{\leq t}) = \Pi_t P_\theta(e_{t,d} | x_d^t).$$

Given a model x, the probability $P_\theta(x)$ is expressed as a sum over the probabilities of all possible decompositions:

$$-\log P_\theta(x) = -\log \sum_d P_\theta(x, d).$$

Let us consider $q(d|x)$, a distribution over the possible decompositions given a model x. We can express that sum as an expected value:

$$-\log \sum_d P_\theta(x, d) = -\log \sum_d P_\theta(x, d) \frac{q(d|x)}{q(d|x)} = -\log E_{q(d|x)} \left(\frac{P_\theta(x, d)}{q(d|x)} \right).$$

Using the Jensen's inequality, the logarithm can be introduced into the expectation. However, an upper-bound of the original equation will be minimized.

$$-\log E_{q(d|x)} \left(\frac{P_\theta(x, d)}{q(d|x)} \right) \leq E_{q(d|x)} \left(-\log \frac{P_\theta(x, d)}{q(d|x)} \right).$$

If we sample k random decompositions following $q(d|x)$, it is possible to approximate the last expected value using a Monte Carlo estimator:

$$E_{q(d|x)} \left(-\log \frac{P_\theta(x, d)}{q(d|x)} \right) \approx \frac{1}{k} \sum_{i=1}^{k} -\log \frac{P_\theta(x, d_i)}{q(d_i|x)} = \frac{1}{k} \sum_{i=1}^{k} -\log P_\theta(x, d_i) + S$$

where S is a term that does not depends on θ (so we can ignore it when optimizing). Therefore, putting all together, the optimization problem to be solved is the following:

$$\hat{\theta} = \text{argmin}_\theta \sum_{j \text{ (training samples)}} \sum_{i \text{ (decompositions)}} \sum_{t \text{ (partial models)}} -\log P_\theta(e_{t,d} | x_{j,d}^t)^4.$$

That is, for each training model, k decompositions are sampled, and for each partial model, the neural model is trained to predict the edit operation that generates the next partial model. Finally, it is worth mentioning that the decompositions are generated considering a priority order in the edit operations. By doing this, the number of addends of the sum $\sum_d P_\theta(x, d)$ is reduced.

[4] For the sake of simplicity, not all indices are shown.

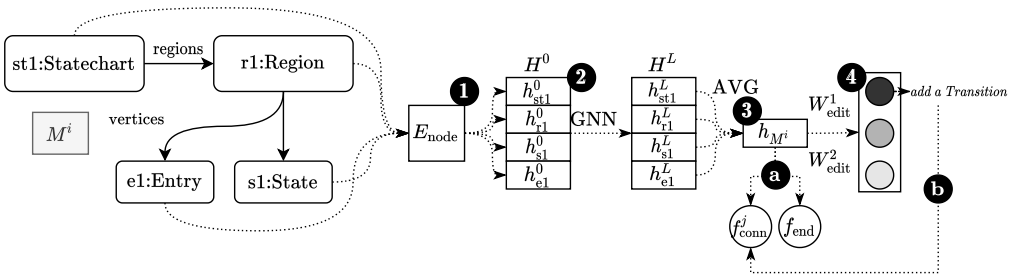


Figure 3.13: Architecture of the edit operation module. Extracted from [5].

Neural model

The neural network architecture combines a GNN to encode the partial graph and a recurrent network to predict the edit connections of the edit operation. It has two main modules:

1. Edit operation selection. This module is in charge of selecting the edit operation that will be applied to the partial model. It is composed of two sub-modules.
   (a) Identifier selection (f_{id}). It selects the edit operation identifier.
   (b) Connectors selection (f^j_{conn}). It selects the nodes in the model under construction to which the edit connections will be linked to.
2. Termination (f_{end}). It determines whether the generation procedure stops or not.

The architecture of the neural model is depicted in Figs 3.13 and 3.14. In particular, Fig. 3.13 represents the edit operation selection module. Firstly ❶, an embedding layer is used to generate initial embeddings for the nodes in the graph. Secondly ❷, these initial embeddings are passed through L GNN layers to get contextualized embeddings. Thirdly ❸, to get the representation of the full graph, the node embeddings are averaged. This average is used by f_{id} (a two-layer neural network followed by a softmax activation) to select the edit operation identifier (❹), by f^j_{conn} as initialization of the recurrent network (ⓐ), and by f_{end} (a two-layer neural network followed by a sigmoid activation) to determine if it is time to stop the generation procedure (ⓐ). Finally, the sampled edit operation is used as input by the f^j_{conn} module (ⓑ). Fig. 3.14 shows the architecture of f^j_{conn}. Firstly ❶, the recurrent model is initialized by the embedding of the input graph and the embedding of the selected edit operation. Finally, the hidden state is concatenated with each one of the node embeddings (❷) and a two-layer neural network followed by a softmax activation is applied to that concatenation (❸) to select the connector node in each time step.

Evaluation of M2

M2 has been evaluated in terms of consistency, diversity, scalability, and realism against three state-of-the-art model generators, namely VIATRA [50], EMF random instantiator [51], and RandomEMF [52]. To this end, four use cases have been evaluated using four datasets. Three of the datasets come from the MAR search engine [1]: Yakindu state-chart models[5], Ecore models, and relational models. The fourth dataset also contains Yakindu models, but they are not taken from a public source but manually constructed as part of an experiment [45].

[5] `https://github.com/Yakindu/statecharts`

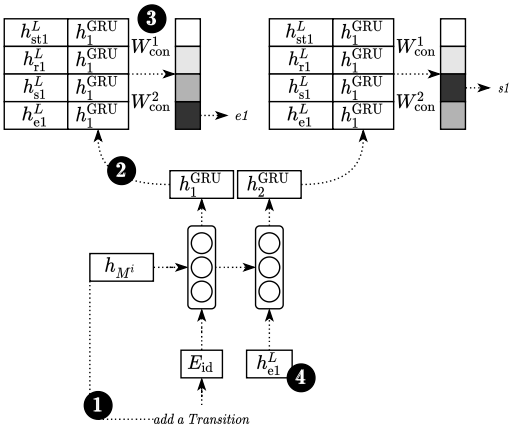


Figure 3.14: Architecture of the connectors' selection sub-module. Extracted from [5].

To assess the consistency, diversity, realism, and scalability the following metrics have been considered respectively: the proportion of consistent models generated, the *internal diversity* metric proposed in [53], the MMD over graph statistics, and the relation between the size of the output models and the generation time. A summary of the results is shown in Table 3.5. The detailed results for each generator, use case, and assessed property are in the paper [5].

Altogether, M2 achieves state-of-the-art results in terms of realism, and it is competitive in terms of consistency, diversity, and scalability. Therefore, M2 is the first generator that supports realism without sacrificing the rest of the properties.

	Consistency	Diversity	Realism	Scalability
RANDOM	○ ○ ○ ○	○ ○ ○ ○	○ ○ ○ ○	● ● ● ●
rEMF	● ● ● ○	● ● ● ●	● ● ○ ○	● ● ● ●
VIATRA	● ● ● ●	● ● ● ●	● ○ ○ ○	● ● ● ○
M2	● ● ● ○	● ● ○ ○	● ● ● ●	● ● ● ○

Table 3.5: Summary of the four properties of each generator. Extracted from [5].

Chapter 4

Conclusion and future work

The main aim of this thesis has been to address fundamental issues for the application of ML to MDE. Particularly, the specific aims have been building datasets and creating application prototypes using ML to address problems in MDE. These objectives have been fulfilled through the following contributions:

- We have proposed the MAR search engine, a scalable search engine designed for software models. MAR offers a novel query-by-example mechanism based on the notion of *bag of paths*. From the point of view of its usefulness to the ML/MDE field, MAR is currently the search engine with the largest collection of models. It indexes more than 500k models of different types (including Ecore, UML, Archimate, etc) which can be used to address unsupervised ML tasks.

- We have built ModelSet, the largest labelled dataset in the context of MDE. ModelSet contains $\sim$ 5k labelled Ecore models and $\sim$ 5k labelled UML models. This has enabled us to build more advanced classification models than the current state of the art and integrate them in MAR.

- We have carried out an extensive comparative study to shed light on the model classification problem, obtaining interesting findings regarding model codification, ML models, and quasi-duplication.

- We have proposed a new definition of realistic model generator using a generative perspective. Furthermore, we propose a model generator M2 trained to satisfy the realistic property. We have compared M2 with three state-of-the-art generators concluding that M2 is the best in terms of realism and is competitive with respect to the other properties.

Moreover, it is worth noting that an added value of the work of this thesis, is that the datasets provided by MAR and ModelSet are being used by other researchers of the modelling community to build interesting ML applications [33–35].

In future work, we plan to maintain MAR and ModelSet by incorporating more types of models and crawling more data sources. In particular, on the ModelSet side, we will extend the labelling methodology to a collaborative setting. Regarding the application of ML to MDE, we would like to tackle the model recommendation task by using M2 as the core component that recommends the next edit operation. We plan also to combine M2 and VIATRA to reach the diversity property and consistency while maintaining realism.

Chapter 5

Publications composing the doctoral thesis

The following PhD Thesis is a compilation of the next published articles, being the PhD student the main author in all of them:

- **José Antonio Hernández López** and Jesús Sánchez Cuadrado. An efficient and scalable search engine for models. Software and Systems Modeling, 21(5):1715–1737, 2022.
- **José Antonio Hernández López**, Javier Luis Cánovas Izquierdo, and Jesús Sánchez Cuadrado. Modelset: a Dataset for Machine Learning in Model-Driven Engineering. Software and Systems Modeling, pages 1–20, 2021.
- **José Antonio Hernández López**, Javier Luis Cánovas Izquierdo, and Jesús Sánchez 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, pages 66–70, 2022.
- **José Antonio Hernández López**, Riccardo Rubei, Jesús Sánchez Cuadrado, and Davide 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, pages 165–175, 2022.
- **José Antonio Hernández López** and Jesús Sánchez Cuadrado. Towards the Characterization of Realistic Model Generators Using Graph Neural Networks. In International Conference on Model-Driven Engineering Languages and Systems, pages 58–69, 2021.
- **José Antonio Hernández López** and Jesús Sánchez Cuadrado. Generating structurally realistic models with deep autoregressive networks. IEEE Transactions on Software Engineering, pages 1–16, 2022.

5.1 An efficient and scalable search engine for models

Title:	An efficient and scalable search engine for models
Authors:	José Antonio Hernández López, Jesús Sánchez Cuadrado
Journal:	Software and Systems Modeling
Impact factor:	2,211 JCR Q3 (2021)
Publisher:	Springer
Year:	2021
Month:	December
DOI:	https://doi.org/10.1007/s10270-021-00960-4
State:	Published
Credits:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing, Visualization, Project administration

Abstract. Search engines extract data from relevant sources and make them available to users via queries. A search engine typically crawls the web to gather data, analyses and indexes it and provides some query mechanism to obtain ranked results. There exist search engines for websites, images, code, etc., but the specific properties required to build a search engine for models have not been explored much. In the previous work, we presented MAR, a search engine for models which has been designed to support a query-by-example mechanism with fast response times and improved precision over simple text search engines. The goal of MAR is to assist developers in the task of finding relevant models. In this paper, we report new developments of MAR which are aimed at making it a useful and stable resource for the community. We present the crawling and analysis architecture with which we have processed about 600,000 models. The indexing process is now incremental and a new index for keyword-based search has been added. We have also added a web user interface intended to facilitate writing queries and exploring the results. Finally, we have evaluated the indexing times, the response time and search precision using different configurations. MAR has currently indexed over 500,000 valid models of different kinds, including Ecore meta-models, BPMN diagrams, UML models and Petri nets. MAR is available at `http://mar-search.org`.

5.2 ModelSet: a dataset for machine learning in model-driven engineering

Title:	ModelSet: a dataset for machine learning in model-driven engineering
Authors:	José Antonio Hernández López, Javier Luis Cánovas Izquierdo, Jesús Sánchez Cuadrado
Journal:	Software and Systems Modeling
Impact factor:	2,211 JCR Q3 (2021)
Publisher:	Springer
Year:	2021
Month:	October
DOI:	https://doi.org/10.1007/s10270-021-00929-3
State:	Published
Credits:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing, Visualization, Project administration

Abstract. The application of machine learning (ML) algorithms to address problems related to model-driven engineering (MDE) is currently hindered by the lack of curated datasets of software models. There are several reasons for this, including the lack of large collections of good quality models, the difficulty to label models due to the required domain expertise, and the relative immaturity of the application of ML to MDE. In this work, we present ModelSet, a labelled dataset of software models intended to enable the application of ML to address software modelling problems. To create it we have devised a method designed to facilitate the exploration and labelling of model datasets by interactively grouping similar models using off-the-shelf technologies like a search engine. We have built an Eclipse plug-in to support the labelling process, which we have used to label 5,466 Ecore meta-models and 5,120 UML models with its category as the main label plus additional secondary labels of interest. We have evaluated the ability of our labelling method to create meaningful groups of models in order to speed up the process, improving the effectiveness of classical clustering methods. We showcase the usefulness of the dataset by applying it in a real scenario: enhancing the MAR search engine. We use ModelSet to train models able to infer useful metadata to navigate search results. The dataset and the tooling are available at `https://figshare.com/s/5a6c02fa8ed20782935c` and a live version at `http://modelset.github.io`.

5.3 Using the ModelSet dataset to support machine learning in model-driven engineering

Title:	Using the ModelSet dataset to support machine learning in model-driven engineering
Authors:	José Antonio Hernández López, Javier Luis Cánovas Izquierdo, Jesús Sánchez Cuadrado
Conference:	ACM / IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS)
Core:	A (2021)
Year:	2022
Month:	October
DOI:	https://doi.org/10.1145/3550356.3559096
State:	Published
Credits:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing, Visualization, Project administration

Abstract. The availability of curated collections of models is essential for the application of techniques like Machine Learning (ML) and Data Analytics to MDE as well as to boost research activities. However, many applications of ML to address MDE tasks are currently limited to small datasets. In this demo paper, we will present ModelSet, a dataset composed of 5,466 Ecore models and 5,120 UML models which have been manually labelled to support ML tasks (`http://modelset.github.io`). ModelSet is built upon the models collected by the MAR search engine (`http://mar-search.org`), which provides more than 500,000 models of different types. We will describe the structure of the dataset and we will explain how to use the associated library to develop ML applications in Python. Finally, we will describe some applications which can be addressed using ModelSet.

5.4 Machine learning methods for model classification: a comparative study

Title:	Machine learning methods for model classification: a comparative study
Authors:	José Antonio Hernández López, Riccardo Rubei, Jesús Sánchez Cuadrado, Davide di Ruscio
Conference:	ACM / IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS)
Core:	A (2021)
Year:	2022
Month:	October
DOI:	https://doi.org/10.1145/3550355.3552461
State:	Published
Credits:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing, Visualization, Project administration

Abstract. In the quest to reuse modeling artifacts, academics and industry have proposed several model repositories over the last decade. Different storage and indexing techniques have been conceived to facilitate searching capabilities to help users find reusable artifacts that might fit the situation at hand. In this respect, machine learning (ML) techniques have been proposed to categorize and group large sets of modeling artifacts automatically. This paper reports the results of a comparative study of different ML classification techniques employed to automatically label models stored in model repositories. We have built a framework to systematically compare different ML models (feed-forward neural networks, graph neural networks, k-nearest neighbors, support version machines, etc.) with varying model encodings (TF-IDF, word embeddings, graphs and paths). We apply this framework to two datasets of about 5,000 Ecore and 5,000 UML models. We show that specific ML models and encodings perform better than others depending on the characteristics of the available datasets (e.g., the presence of duplicates) and on the goals to be achieved.

5.5 Towards the characterization of realistic model generators using graph neural networks

Title:	Towards the characterization of realistic model generators using graph neural networks
Authors:	José Antonio Hernández López, Jesús Sánchez Cuadrado
Conference:	ACM / IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)
Core:	A (2020)
Year:	2021
Month:	October
DOI:	https://doi.org/10.1109/MODELS50736.2021.00015
State:	Published
Award:	ACM Distinguished Paper Award
Credits:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing, Visualization, Project administration

Abstract. The automatic generation of software models is an important element in many software and systems engineering scenarios such as software tool certification, validation of cyber-physical systems, or benchmarking graph databases. Several model generators are nowadays available, but the topic of whether they generate realistic models has been little studied. The state-of-the-art approach to check the realistic property in software models is to rely on simple comparisons using graph metrics and statistics. This generates a bottleneck due to the compression of all the information contained in the model into a small set of metrics. Furthermore, there is a lack of interpretation in these approaches since there are no hints of why the generated models are not realistic. Therefore, in this paper, we tackle the problem of assessing how realistic a generator is by mapping it to a classification problem in which a Graph Neural Network (GNN) will be trained to distinguish between the two sets of models (real and synthetic ones). Then, to assess how realistic a generator is we perform the Classifier Two-Sample Test (C2ST). Our approach allows for interpretation of the results by inspecting the attention layer of the GNN. We use our approach to assess four state-of-the-art model generators applied to three different domains. The results show that none of the generators can be considered realistic.

5.6 Generating structurally realistic models with deep autoregressive networks

Title:	Generating structurally realistic models with deep autoregressive networks
Authors:	José Antonio Hernández López, Jesús Sánchez Cuadrado
Journal:	IEEE Transactions on Software Engineering
Impact factor:	9,322 JCR Q1, D1 (2021)
Publisher:	IEEE
Year:	2022
Month:	December
DOI:	https://doi.org/10.1109/TSE.2022.3228630
State:	Published
Credits:	Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Resources, Data Curation, Writing, Visualization, Project administration

Abstract. Model generators are important tools in model-based systems engineering to automate the creation of software models for tasks like testing and benchmarking. Previous works have established four properties that a generator should satisfy: consistency, diversity, scalability, and structural realism. Although several generators have been proposed, none of them is focused on realism. As a result, automatically generated models are typically simple and appear synthetic. This work proposes a new architecture for model generators which is specifically designed to be structurally realistic. Given a dataset consisting of several models deemed as real models, this type of generators is able to produce new models which are structurally similar to the models in the dataset, but are fundamentally novel models. Our implementation, named ModelMime (M2), is based on a deep autoregressive model which combines a Graph Neural Network with a Recurrent Neural Network. We decompose each model into a sequence of edit operations, and the neural network is trained in the task of predicting the next edit operation given a partial model. At inference time, the system produces new models by sampling edit operations and iteratively completing the model. We have evaluated M2 with respect to three state-of-the-art generators, showing that 1) our generator outperforms the others in terms of the structurally realistic property 2) the models generated by M2 are most of the time consistent, 3) the diversity of the generated models is at least the same as the real ones and, 4) the generation process is scalable once the generator is trained.

Chapter 6

Research stays and other publications

To close this thesis, it is worth mentioning the research stays performed by the PhD student and other publications not included in the compendium published in the MDE and Software Engineering fields.

6.1 Research stays

- Research stay at the University of Montreal (Montreal, Canada). It lasted 4 months. The main aim of this research stay was to determine whether large pre-trained language models of code are able to understand the syntax of a programming language. This research improves the understanding of the internals of neural networks, which are used pervasively in the thesis. This research stay resulted in a paper [54] published at the Automated Software Engineering (ASE) Conference.

- Research stay at IncQuery Labs[1] (Budapest, Hungary). It lasted 2 months. The main aim of this research stay was to improve the main product of the company by developing a compiler that transforms MagicDraw Structured Expressions[2] to Viatra Queries[3]. This stay helped the PhD student to understand software modelling more in-depth and to be more in touch with that field in an industrial context.

6.2 Other publications

- **José Antonio Hernández López** and Jesús Sánchez Cuadrado. MAR: a structure-based search engine for models. In International Conference on Model-Driven Engineering Languages and Systems, pages 57–67, 2020.

- **José Antonio Hernández López**, Martin Weyssow, Jesús Sánchez Cuadrado, and Houari Sahraoui. AST-Probe: Recovering abstract syntax trees from hidden representations of pre-trained language models. In 37th IEEE/ACM International Conference on Automated Software Engineering, pages 1–11, 2022.

[1] `https://incquery-group.com/`

[2] `https://docs.nomagic.com/display/MD185/Specifying+criteria+for+querying+model`

[3] `https://www.eclipse.org/viatra/documentation/query-language.html`

Bibliography

[1] José Antonio Hernández López and Jesús Sánchez Cuadrado. MAR: a structure-based search engine for models. In *Int. Conf. on model driven engineering languages and systems*, pages 57–67, 2020.

[2] José Antonio Hernández López and Jesús Sánchez Cuadrado. An efficient and scalable search engine for models. *Software and Systems Modeling*, 21(5):1715–1737, 2022.

[3] José Antonio Hernández López, Javier Luis Cánovas Izquierdo, and Jesús Sánchez Cuadrado. Modelset: a Dataset for Machine Learning in Model-driven Engineering. *Software and Systems Modeling*, pages 1–20, 2021.

[4] José Antonio Hernández López, Riccardo Rubei, Jesús Sánchez Cuadrado, and Davide 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*, pages 165–175, 2022.

[5] José Antonio Hernández López and Jesús Sánchez Cuadrado. Generating structurally realistic models with deep autoregressive networks. *IEEE Transactions on Software Engineering*, pages 1–16, 2022. doi: 10.1109/TSE.2022.3228630.

[6] Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE software*, 31(3):79–85, 2013.

[7] John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *Proceedings of the 33rd international conference on software engineering*, pages 471–480, 2011.

[8] Antonio Bucchiarone, Jordi Cabot, Richard F Paige, and Alfonso Pierantonio. Grand challenges in model-driven engineering: an analysis of the state of the research. *Software and Systems Modeling*, 19(1):5–13, 2020.

[9] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[10] Grigorios G Chrysos, Stylianos Moschoglou, Giorgos Bouritsas, Yannis Panagakis, Jiankang Deng, and Stefanos Zafeiriou. P-nets: Deep polynomial neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7325–7335, 2020.

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[12] Sebastian Raschka and Vahid Mirjalili. *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2*. Packt Publishing Ltd, 2019.

[13] Miltiadis Allamanis, Earl T Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4): 1–37, 2018.

[14] Tushar Sharma, Maria Kechagia, Stefanos Georgiou, Rohit Tiwari, and Federica Sarro. A survey on machine learning techniques for source code analysis. *arXiv preprint arXiv:2110.09610*, 2021.

[15] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, Alfonso Pierantonio, and Ludovico Iovino. Automated classification of metamodel repositories: A machine learning approach. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 272–282. IEEE, 2019.

[16] Mohd Hafeez Osman, Truong Ho-Quang, and Michel Chaudron. An automated approach for classifying reverse-engineered and forward-engineered uml class diagrams. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 396–399. IEEE, 2018.

[17] Francesco Basciani, Juri Di Rocco, Davide Di Ruscio, Ludovico Iovino, and Alfonso Pierantonio. Automated clustering of metamodel repositories. In *International Conference on Advanced Information Systems Engineering*, pages 342–358. Springer, 2016.

[18] Önder Babur, Loek Cleophas, and Mark van den Brand. Metamodel clone detection with samos. *Journal of Computer Languages*, 51:57–74, 2019.

[19] Angela Barriga, Adrian Rutle, and Rogardt Heldal. Personalized and automatic model repairing using reinforcement learning. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 175–181. IEEE, 2019.

[20] Önder Babur. A Labeled Ecore Metamodel Dataset for Domain Clustering. URL `https://zenodo.org/record/2585456`.

[21] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel RV Chaudron, and Miguel Angel Fernandez. An Extensive Dataset of Uml Models in GitHub. In *Int. Conf. on Mining Software Repositories*, pages 519–522. IEEE, 2017.

[22] Mathias Weske, Gero Decker, Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. Model Collection of the Business Process Management Academic Initiative, April 2020. URL `https://doi.org/10.5281/zenodo.3758705`.

[23] Robert Clarisó and Jordi Cabot. Applying graph kernels to model-driven engineering problems. In *Proceedings of the 1st International Workshop on Machine Learning and Software Engineering in Symbiosis*, pages 1–5, 2018.

[24] Loli Burgueño, Jordi Cabot, and Sébastien Gérard. An lstm-based neural network architecture for model transformations. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 294–299. IEEE, 2019.

[25] Basciani Francesco, Iovino Ludovico, Pierantonio Alfonso, et al. Mdeforge: an extensible web-based modeling platform. In *Proceedings of the 2nd International Workshop on Model-Driven Engineering on and for the Cloud co-located with the 17th International Conference on Model Driven Engineering Languages and Systems, CloudMDE@ MoDELS 2014, Valencia, Spain, September 30, 2014*, volume 1242, pages 66–75. CEUR-WS, 2014.

[26] José Antonio Hernández López, Javier Luis Cánovas Izquierdo, and Jesús Sánchez 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*, pages 66–70, 2022.

[27] Phuong T Nguyen, Davide Di Ruscio, Alfonso Pierantonio, Juri Di Rocco, and Ludovico Iovino. Convolutional neural networks for enhanced classification mechanisms of metamodels. *Journal of Systems and Software*, 172:110860, 2021.

[28] Miltiadis 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*, pages 143–153, 2019.

[29] OMG Available Specification. Object constraint language, 2006.

[30] Gábor Bergmann, Zoltán Ujhelyi, István Ráth, and Dániel Varró. A graph query language for emf models. In *International Conference on Theory and Practice of Model Transformations*, pages 167–182. Springer, 2011.

[31] Dániel Varró, Oszkár Semeráth, Gábor Szárnyas, and Ákos Horváth. Towards the automated generation of consistent, diverse, scalable and realistic graph models. In *Graph Transformation, Specifications, and Nets*, pages 285–312. Springer, 2018.

[32] José Antonio Hernández López and Jesús Sánchez Cuadrado. Towards the Characterization of Realistic Model Generators Using Graph Neural Networks. In *Int. Conf. on Model Driven Engineering Languages and Systems*, pages 58–69, 2021.

[33] Martin Weyssow, Houari Sahraoui, and Eugene Syriani. Recommending metamodel concepts during modeling activities with pre-trained language models. *Software and Systems Modeling*, 21(3):1071–1089, 2022.

[34] Meriem Ben Chaaben, Lola Burgueño, and Houari Sahraoui. Towards using few-shot prompt learning for automating model completion. *arXiv preprint arXiv:2212.03404*, 2022.

[35] Lissette Almonte, Sara Pérez-Soler, Esther Guerra, Iván Cantador, and Juan de Lara. Automating the synthesis of recommender systems for modelling languages. In *Proceedings of the 14th ACM SIGPLAN International Conference on Software Language Engineering*, pages 22–35, 2021.

[36] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1): 61–80, 2008.

[37] Shafiul Azam Chowdhury, Lina Sera Varghese, Soumik Mohian, Taylor T Johnson, and Christoph Csallner. A curated corpus of simulink models for model-based empirical studies. In *2018 IEEE/ACM 4th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*, pages 45–48. IEEE, 2018.

[38] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389, 2009.

[39] ChengXiang Zhai and Sean Massung. Text data management and analysis: A practical introduction to information retrieval and text mining. 2016.

[40] Phuong T Nguyen, Juri Di Rocco, Ludovico Iovino, Davide Di Ruscio, and Alfonso Pierantonio. Evaluation of a machine learning classifier for metamodels. *Software and Systems Modeling*, 20(6):1797–1821, 2021.

[41] Loli Burgueño, Robert Clarisó, Sébastien Gérard, Shuai Li, and Jordi Cabot. An nlp-based architecture for the autocompletion of partial domain models. In *International Conference on Advanced Information Systems Engineering*, pages 91–106. Springer, 2021.

[42] Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Phuong T Nguyen. A gnn-based recommender system to assist the specification of metamodels and models. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 70–81. IEEE, 2021.

[43] Riccardo Rubei, Juri Di Rocco, Davide Di Ruscio, Phuong T. Nguyen, and Alfonso Pierantonio. A lightweight approach for the automated classification and clustering of metamodels. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 477–482, 2021. doi: 10.1109/MODELS-C53483.2021.00074.

[44] Claudio Di Sipio, Riccardo Rubei, Davide Di Ruscio, and Phuong T Nguyen. A multinomial naïve bayesian (mnb) network to automatically recommend topics for github repositories. In *Proceedings of the Evaluation and Assessment in Software Engineering*, pages 71–80. 2020.

[45] Oszkár Semeráth, Aren A Babikian, Boqi Chen, Chuning Li, Kristóf Marussy, Gábor Szárnyas, and Dániel Varró. Automated generation of consistent, diverse and structurally realistic graph models. *Software and Systems Modeling*, pages 1–22, 2021.

[46] David Lopez-Paz and Maxime Oquab. Revisiting classifier two-sample tests. *arXiv preprint arXiv:1610.06545*, 2016.

[47] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *The Journal of Machine Learning Research*, 13(1):723–773, 2012.

[48] Jiaxuan You, Rex Ying, Xiang Ren, William Hamilton, and Jure Leskovec. Graphrnn: Generating realistic graphs with deep auto-regressive models. In *International Conference on Machine Learning*, pages 5708–5717. PMLR, 2018.

[49] Gábor Szárnyas, Zsolt Kővári, Ágnes Salánki, and Dániel Varró. Towards the characterization of realistic models: evaluation of multidisciplinary graph metrics. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 87–94, 2016.

[50] Oszkár Semeráth, András Szabolcs Nagy, and Dániel Varró. A graph solver for the automated generation of consistent domain-specific models. In *Proceedings of the 40th International Conference on Software Engineering*, pages 969–980, 2018.

[51] AtlanMod Team (Inria, Mines-Nantes, Lina), EMF random instantiator. URL `https://github.com/atlanmod/mondo-atlzoo-benchmark/tree/master/fr.inria.atlanmod.instantiator`.

[52] Markus Scheidgen. Generation of large random models for benchmarking. *BigMDE@ STAF*, 1406:1–10, 2015.

[53] Oszkár Semeráth, Rebeka Farkas, Gábor Bergmann, and Dániel Varró. Diversity of graph models and graph generators in mutation testing. *International Journal on Software Tools for Technology Transfer*, 22(1):57–78, 2020.

[54] José Antonio Hernández López, Martin Weyssow, Jesús Sánchez Cuadrado, and Houari Sahraoui. Ast-probe: Recovering abstract syntax trees from hidden representations of pre-trained language models. In *37th IEEE/ACM International Conference on Automated Software Engineering*, pages 1–11, 2022.