

Article

SEMKIS-DSL: A Domain-Specific Language to Support Requirements Engineering of Datasets and Neural Network Recognition

Benjamin Jahić [†] , Nicolas Guelfi [†]  and Benoît Ries ^{*,†} 

Department of Computer Science, Faculty of Science, Université du Luxembourg, Technology and Medecine, Campus Belval, L-4365 Esch-sur-Alzette, Luxembourg; nicolas.guelfi@uni.lu (N.G.)

* Correspondence: benoit.ries@uni.lu; Tel.: +352-46-66-44-5267

† These authors contributed equally to this work.

Abstract: Neural network (NN) components are being increasingly incorporated into software systems. Neural network properties are determined by their architecture, as well as the training and testing datasets used. The engineering of datasets and neural networks is a challenging task that requires methods and tools to satisfy customers' expectations. The lack of tools that support requirements specification languages makes it difficult for engineers to describe dataset and neural network recognition skill requirements. Existing approaches often rely on traditional ad hoc approaches, without precise requirement specifications for data selection criteria, to build these datasets. Moreover, these approaches do not focus on the requirements of the neural network's expected recognition skills. We aim to overcome this issue by defining a domain-specific language that precisely specifies dataset requirements and expected recognition skills after training for an NN-based system. In this paper, we present a textual domain-specific language (DSL) called SEMKIS-DSL (Software Engineering Methodology for the Knowledge management of Intelligent Systems) that is designed to support software engineers in specifying the requirements and recognition skills of neural networks. This DSL is proposed in the context of our general SEMKIS development process for neural network engineering. We illustrate the DSL's concepts using a running example that focuses on the recognition of handwritten digits. We show some requirements and recognition skills specifications and demonstrate how our DSL improves neural network recognition skills.

Keywords: neural network; domain-specific language; method; requirements; model-driven engineering



Citation: Jahić, B.; Guelfi, N.; Ries, B. SEMKIS-DSL: A Domain-Specific Language to Support Requirements Engineering of Datasets and Neural Network Recognition. *Information* **2023**, *14*, 213. <https://doi.org/10.3390/info14040213>

Academic Editor: José J. Pazos Arias

Received: 3 March 2023

Revised: 24 March 2023

Accepted: 29 March 2023

Published: 1 April 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Software engineers are increasingly developing neural networks (NN) for customers in various domains such as finance, medicine, and autonomous driving. It goes without saying that these NNs must satisfy customers' needs and requirements. Software engineers rely on methods and tools that support requirements engineering (RE). However, there is a lack of such methods and tools [1] to engineer datasets and NNs satisfying the requirements.

In traditional approaches to dataset engineering, data scientists consider high-level requirements in building datasets for training and testing NNs. They often build datasets using time-consuming and expensive ad hoc approaches [2]. These approaches are often not generic and do not rely on precise data selection criteria based on customer requirements. In addition, they typically utilize manual data collection [3,4] to construct datasets for training NNs. As such, they are error-prone, which can lead to numerous data collection/labeling and NN training iterations due to inaccurate neural networks. These NNs may not accurately recognize some data within a certain equivalence class, failing to meet the requirements. We define an equivalence class [5] as a group of related artifacts that an NN recognizes within its category. As a result, multiple exhaustive and costly

iterations are required to enhance the NN's recognition skills and meet the customer's requirements. We define recognition skills [5] as informal textual descriptions of the NN's learned capacity to correctly map input data to the corresponding equivalence class. To address the need for methods and tools for dataset engineering, we introduce a software engineering (SE) methodology [6] that is designed as an iterative business process and focuses on augmenting datasets with synthetic data to improve NNs. To semi-formally describe the data selection criteria for augmenting these datasets, we introduce the notion of key properties [7] to describe the NN's acquired recognition skills after training. Our initial approach specifically addresses recognition skills requirements and dataset requirements in this domain. Rather than developing an entirely new requirements engineering method, we analyze and experiment with existing general SE methods for traditional software systems.

We define the NN's key properties [5] as quantitative and qualitative attributes that describe the NN's recognition skills. The purpose of the key properties is to help engineers precisely understand the NN's recognition skills and determine the precise data selection criteria for enhancing datasets to improve NNs. We use model-driven engineering (MDE) principles to develop an approach for automated data generation that is based on customer requirements and NN key properties. In MDE, models are used to describe a conceptual representation of a domain problem or system using a limited vocabulary. Moreover, these models can be transformed into other models using model transformation rules (e.g., model-to-text transformation).

In this paper, we introduce a textual domain-specific language (DSL) called SEMKIS-DSL that is designed to help software engineers specify the requirements and recognition skills for neural networks. We first define a metamodel to outline the concepts related to customer requirements and NN key properties. Then, we present the domain-specific language, which is compliant with the metamodel and serves as an MDE tool to support the specification of requirements and key properties. The DSL guides users to create structured, readable, and maintainable requirements and key properties. We also describe how some informal model transformations from these specification models can be transformed into a data synthesizer.

The research topic we address in this paper is how to facilitate requirements engineering in NN-based systems for dataset and recognition capabilities using a domain-specific language. Our aim is to enable the artificial intelligence (AI) community to benefit from the MDE principles for specifying the customer requirements and NN key properties to automate data generation. This contribution is the first step towards a fully automated transformation of the requirements and key-properties model into a data synthesizer implementation. Thanks to the MDE principles, we aim to reduce the costs and efforts required for performing dataset engineering activities.

This paper is organized as follows. Section 2 presents the background materials, methods, and relevant references we use to design and implement the SEMKIS-DSL. In Section 3, we present the results of the SEMKIS-DSL and the basis for the development of the model transformations and demonstrate our concepts. In Section 4, we present the works and relevant references related to (1) the software development process for dataset engineering, (2) MDE for deep learning, and (3) MDE for deep learning using NNs for dataset augmentation. In Section 5, we discuss some limitations, threats to the validity of our approach, and potential research directions. Finally, we present a summary of our work, the main contributions, and some ideas for future works in Section 6.

2. Materials and Methods

In this section, we present the materials and methods that serve as input for the design and implementation of the SEMKIS-DSL. We also present the model-driven engineering principles employed in the SEMKIS-DSL for automating dataset engineering activities. Additionally, we present the SEMKIS methodology of which the SEMKIS-DSL is an integral part. Finally, we present the method used to design the SEMKIS-DSL.

2.1. Material

2.1.1. Model-Driven Engineering Context

Fondement and Silaghi [8] define MDE as an SE methodology that relies on conceptual specifications to describe a particular domain problem and its solution. These conceptual specifications, known as domain models, can be specified at different levels of abstraction. MDE follows a top-down approach to designing software systems by starting with a domain model specification at a high abstraction level and culminating in a refined domain model specification at a lower abstraction level.

A tool that is commonly used to specify domain models is domain-specific languages (DSL). Tomaž et al. [9] define a domain-specific language as a tailor-made and high-abstraction-level language that is based on concepts and properties from a specific application domain. There exist two different types of DSLs: textual and graphical DSLs. Both consist of defined grammar and semantics for specifying domain models within a specific domain problem. DSLs are often designed as intermediate languages that are translated into another domain model (e.g., into a different language) using model transformations. For example, a DSL specification enables the development of an automated code generator that generates source code (e.g., Java, Python). DSLs, with their higher level of abstraction, can be designed to facilitate the specification of domain models that are comprehensible and specifiable by domain experts from diverse IT backgrounds.

Therefore, DSLs help to create a pleasant environment for domain experts who can focus on solving domain problems. Textual editors for DSLs typically offer features such as auto-completion, templates, and error detection to increase the efficiency of the modeling process.

The AI community can benefit from these MDE principles and tools to automatize some dataset engineering activities for improving NNs. Rather than following traditional dataset engineering approaches, we aim to design an MDE process to automate the generation of synthetic data based on precise data selection criteria. In an MDE process, an engineer specifies a data model that presents a specification of certain data selection criteria in collaboration with domain experts. By separating the conceptual specification from the concrete dataset construction, we allow domain experts to validate the targeted dataset early in the development process. Hence, errors can be detected earlier, resulting in fewer corrective development iterations. After the model has been validated, the resulting validated data selection model can be interpreted and transformed into a data synthesizer implementation using model transformation techniques. Finally, a data synthesizer can be used to generate synthetic data to build improved datasets for training and testing NNs.

With the SEMKIS-DSL defined in this paper, we take one step closer to an MDE process (see Figure 1) integrated within our SEMKIS methodology to support software engineers in their deep learning (DL) development projects.

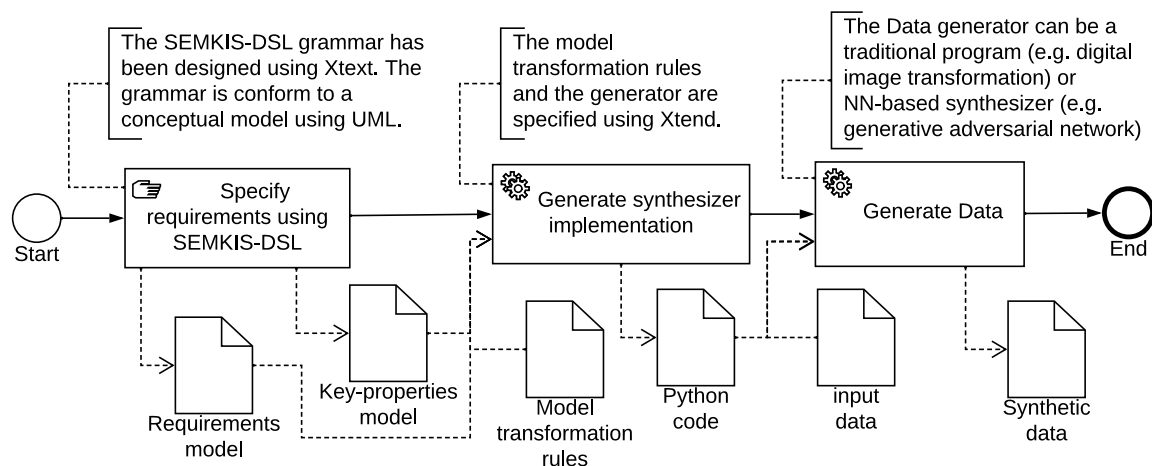


Figure 1. SEMKIS MDE process overview.

2.1.2. SEMKIS Methodology Overview

SEMKIS is an SE methodology that supports software engineers in improving their datasets and NN engineering activities. We designed SEMKIS as an iterative business process [6] that consists of various datasets and NN engineering activities with a few input/output data objects. The aim of this process is to provide precisely defined SE activities that can help software engineers iteratively augment datasets with synthetic data to re-train NNs and improve their recognition skills.

To improve the selection of synthetic data for building datasets, we take into account both the customer's requirements and the NN's acquired recognition skills when defining the precise data selection criteria. As a result, we extend our initial process with RE activities, which include a subprocess for analyzing the test results of a trained NN, and the notion of key properties to describe the NN's recognition skills [5]. The customer's requirements serve to define the NN architecture and certain data selection criteria for creating the initial datasets for training and testing the NN. After testing the NN, the software engineer analyses the test results to specify the NN's key properties by describing its acquired recognition skills during the training. By comparing the requirements and key-properties specification, the engineer can perform the following tasks:

- Determine whether the NN satisfies the requirements.
- Validate the NN's key properties against the customer's requirements.
- Identify new data selection criteria to augment the datasets.

Thus, we use the data selection criteria to synthesize additional data to augment our datasets and re-train the NN. Firstly, the data selection criteria depend on the customer's requirements. For example, we may define some data selection criteria such as (1) incorporating synthetic data of an equivalence class to strengthen its recognition, as the customer has assigned a higher recognition priority to it; (2) reducing the amount of data of an equivalence class as its recognition precision is not critical for the customer; and (3) dividing an equivalence class into two sub-equivalence classes so that the NN can recognize the data separately. Secondly, we can determine some data selection criteria from the key properties to improve the NN's recognition. We may adjust the quantity of data and the variation/balancing of data within specific equivalence classes to improve the recognition. Our process can be executed many times until the customer validates the NN.

We analyzed our SEMKIS activities to improve the engineering of a dataset based on a customer's requirements and an NN's key properties. As a result, we decided to work on an MDE process that supports automated dataset generation from requirements and key-properties specifications. Among the SEMKIS activities, we identified two activities that can be used to specify requirements and key properties with the SEMKIS-DSL:

- *Pre-activity*: This involves specifying the requirements that will serve as input to the process for engineering the initial datasets and NN architecture.
- *Test monitoring data analysis*: This involves analyzing the NN's test results to determine its key properties, which will serve as input to analyze the satisfaction of the requirements and augment the datasets accordingly.

In this paper, we introduce a domain-specific language known as SEMKIS-DSL to support software engineers in specifying these requirements and key properties during the execution of the SEMKIS process. We build the SEMKIS-DSL in the context of our SEMKIS methodology. Our aim is for engineers to benefit from these MDE principles [8] by writing specifications at a higher level of abstraction that focus only on customer requirements and key properties. Thus, we aim to improve their understanding of the customer's requirements and NN's key properties. By refining these abstract specifications using MDE techniques (model transformations), we aim to generate appropriate datasets for engineering improved NNs. In addition, due to its many features (e.g., auto-completion, templates, outline view, hovering, syntax highlighting), our SEMKIS tool supported by the SEMKIS-DSL can help engineers to efficiently write improved specifications. The roles of the SEMKIS stakeholders with respect to the SEMKIS-DSL are as follows:

- The software engineer is responsible for engineering the datasets and the NN, eliciting and specifying the customer's requirements, analyzing the NN's test results and specifying the key properties, and discussing the key properties and the requirements with the customer.
- The dataset engineer is responsible for identifying and suggesting the required data with the software engineer for building the datasets, training the NN to satisfy the customer's requirements, and analyzing the customer's validation feedback, requirements, and key-properties specifications to propose a dataset augmentation strategy for improving the NN.
- The customer is responsible for ordering an NN, participating in meetings with the software engineer to express their requirements, and validating the specified NN's key properties.

2.2. Method

We used a top-down approach to define the different concepts of the SEMKIS-DSL in the context of engineering appropriate datasets to improve NNs. The SEMKIS-DSL was designed in the context of the SEMKIS methodology to support software engineers in improving their datasets and NN engineering activities. The steps for constructing the results for the customer requirements specifications using SEMKIS-DSL were as follows:

1. We designed an academic case study that was small in size but rich enough to support the research process described below.
2. We analyzed the preliminary activity of the dataset augmentation process [5] in the context of the SEMKIS methodology to identify any weaknesses. This preliminary activity focuses on specifying the customer's requirements and defining the target NN.
3. We analyzed the identified weaknesses and found that:
 - Specifying requirements is a complicated, time-consuming, and costly task due to the manual work involved.
 - Our specifications are often handcrafted and unstructured as they are written in basic textual editors (e.g., Word, Excel). These textual editors lack advanced features such as syntax highlighting, code completion, templates, cross-references, and file organization.
 - There are only basic guidelines to help engineers specify the requirements. The engineers still rely on domain experts (AI and dataset engineers), who have a broad knowledge of NN and dataset concepts, to specify the requirements.

4. We designed a metamodel (concept model) that defined the various concepts that were required to specify the requirements for dataset and NN engineering concepts.
5. We designed grammar (Xtext grammar) for tackling the above-mentioned weaknesses to allow for textual specifications of the requirements.
6. We specified some requirements for the MNIST case study [10] to experiment and validate our SEMKIS-DSL.
7. We analyzed the SEMKIS-DSL to propose a basis for developing model-to-text transformations to allow the code-skeleton generation of synthetic data generators or target NN architecture templates.

Similar to the steps for constructing the results for the customer requirements specifications, the steps for constructing the results for the key-properties specifications using SEMKIS-DSL were as follows:

1. We analyzed the activity (Analyze Test Monitoring Data (activity F)) of the dataset augmentation process to identify any weaknesses. This activity is a subprocess comprising four activities for analyzing and specifying the NN's key properties from the collected recognition data (test monitoring data) of the NN's training and testing.
2. We analyzed the identified weaknesses and found, similar to the customer requirements, that:
 - Specifying key properties is a complicated, time-consuming, and costly task due to the manual and handcrafted work involved.
 - Our specifications are often handcrafted and unstructured as they are written in basic textual editors (e.g., Word, Excel). These textual editors lack advanced features.
 - There is a lack of guidelines to help engineers analyze the test monitoring data and specify the key properties. The engineers rely even more on AI and dataset experts to specify the key properties.
 - The specifications have to be manually interpreted to determine new data selection criteria for improving the datasets.
3. We designed a metamodel (concept model) that defined the various concepts that were required to specify the key properties of trained NNs.
4. We designed grammar (Xtext grammar) for tackling the above-mentioned weaknesses to allow for textual specifications of the key properties.
5. We trained an NN to recognize handwritten digits from the MNIST case study [10] and specified some key properties to experiment and validate our SEMKIS-DSL.
6. We analyzed the specifications to propose a basis for developing model-to-text transformations to allow the code-skeleton generation of a synthetic data generator for augmenting our datasets to re-train the NN.

3. Results: SEMKIS-DSL—A Language for Specifying Requirements and Key-Properties of Neural Networks

This section constitutes the core contribution of this paper. In this section, we present the SEMKIS-DSL concepts in the context of the two activities mentioned in the previous section. We introduce a conceptual model (metamodel) for business stakeholders, which we use to engineer the SEMKIS-DSL grammar. The DSL can support software engineers in specifying the customer's requirements (for describing the customer's needs and determining the data selection criteria) and the NN's key properties (for describing and validating the NN's recognition skills). These specifications are used to automatize the generation of improved datasets utilizing an MDE method. We aim to generate additional synthetic data to improve the datasets and, thus, improve an NN until it satisfies the customer's requirements. The SEMKIS-DSL is developed using the Xtext framework [11], which is an open source framework for engineering textual domain-specific languages that provides textual editors for the defined DSLs. The Xtext framework is implemented as a plugin to Eclipse IDE and, consequently, benefits from Eclipse IDE support for languages. The complete SEMKIS-DSL grammar is publicly available on Github [12,13].

3.1. MNIST Running Example

To illustrate the concepts of the SEMKIS-DSL, we use the MNIST case study [10] as a running example. In this paper, we consider a subset of the MNIST dataset and focus on the recognition of the handwritten digits “0” and “7”. We simulate an artificial customer who requires a neural network. The customer meets with a software engineer to outline his needs for the NN. In this context, we instantiate our SEMKIS process to demonstrate the concrete requirements and key-properties specifications. We simulate the engineering of datasets containing images of handwritten digits that are used to train and test the NN.

3.2. Requirements Specifications Using SEMKIS-DSL

In this section, we present the SEMKIS-DSL concepts related to the requirements specifications. We present the requirements concepts to describe the customer’s needs in the context of dataset engineering for NNs. We utilize a traditional approach to define the non-functional and functional requirements, as proposed by Sommerville [14], which we apply in the context of deep learning.

RE is the typical starting point of SE lifecycle models [15]. During the RE phase, a software engineer meets with a customer to discuss their needs concerning the required software. The aim of this phase is to understand the customer’s needs and develop software that satisfies them.

Our process starts with the so-called pre-activity that focuses on RE to describe the customer’s needs for an NN. A software engineer meets with a customer to understand and discuss their needs concerning the required NN. After the first meeting, the software engineer specifies an initial set of requirements based on their understanding. Then, the software engineer meets with the customer again to discuss the specified requirements for validation. If the requirements have been validated, the software engineer proceeds to engineer the datasets. Otherwise, the requirements must be updated and discussed with the customer for validation. The final requirements are used as input in the next activities to engineer the datasets and the target NN.

3.2.1. Concept Model

In this section, we define a metamodel that comprises the concepts related to the RE phase. Figures 2–4 illustrate these concepts in a UML class diagram [16]. We indicate in parentheses the associated UML class for each concept.

The software engineer specifies each of the customer’s requirements (*ctRequirements*) as either a functional or non-functional requirement. Each requirement must be named and described, and a purpose and priority should be specified. Functional requirements (*ctFunctionalRequirement*) describe the expected behaviors that the target NN must offer. The functional requirements describe the targeted recognition skills of the target NN. Each functional requirement defines the expected target NN’s output given a certain target NN’s input. As a result, we specify the following two concepts related to the functional requirements, as illustrated in Figure 3, and the datasets, as illustrated in Figure 2:

- The target NN’s input (*ctTargetNNInput*) describes the customer’s required input data and some technical details about the target NN’s architecture such as the input format and size. The input data (*ctInputData*) represent the data elements (*ctDataElement*) of a dataset (*ctDataset*). Additionally, they define the target NN’s input value format that must be readable by an NN. Datasets consist of data elements, which are classified into equivalence classes (*ctEquivalenceClass*). We define an equivalence class as a group of related artifacts that an NN recognizes within its category. A data element can be a numerical value, image, video, voice, or any other input data that is processable by an NN. In practice, the software engineer starts by specifying the datasets, their data elements, and equivalence classes. Then, the engineer specifies the target NN’s input characteristics and associates the input data with the data elements of the dataset.
- The target NN’s output (*ctTargetNNOutput*) describes the customer’s expected output given a certain input. Given an input data element, an NN typically outputs a

probability distribution over multiple equivalence classes that describes the likelihood that the data element belongs to an equivalence class. Hence, the target NN’s output specifies the output format and size, as well as the accepted value range. Additionally, the output specifies the equivalence classes for which a probability is computed by the NN.

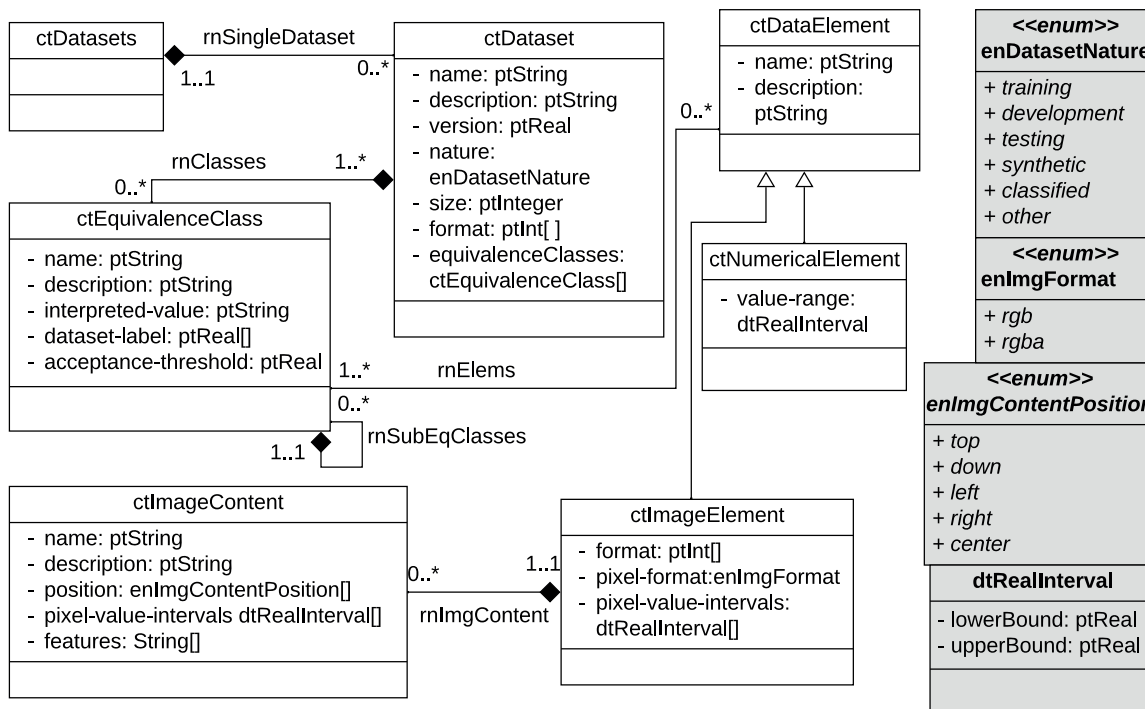


Figure 2. SEMKIS-DSL Metamodel: Dataset specification concepts.

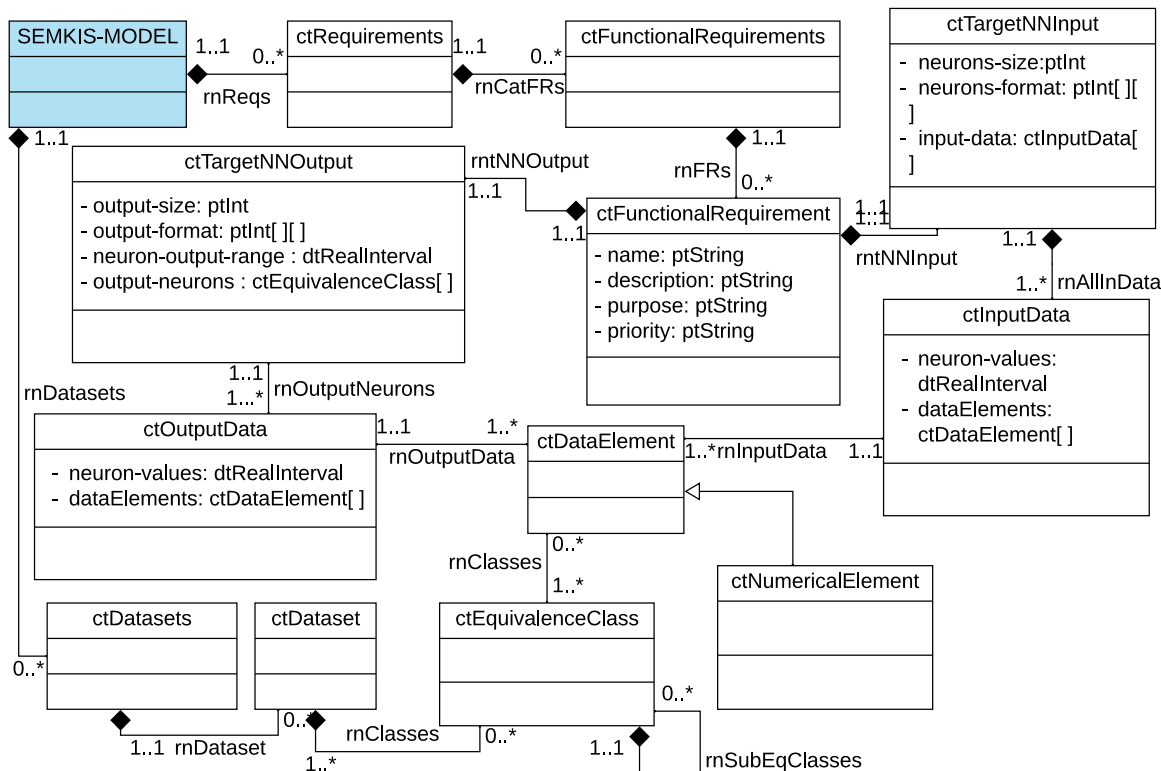


Figure 3. SEMKIS-DSL Metamodel: Functional requirements specification concepts.

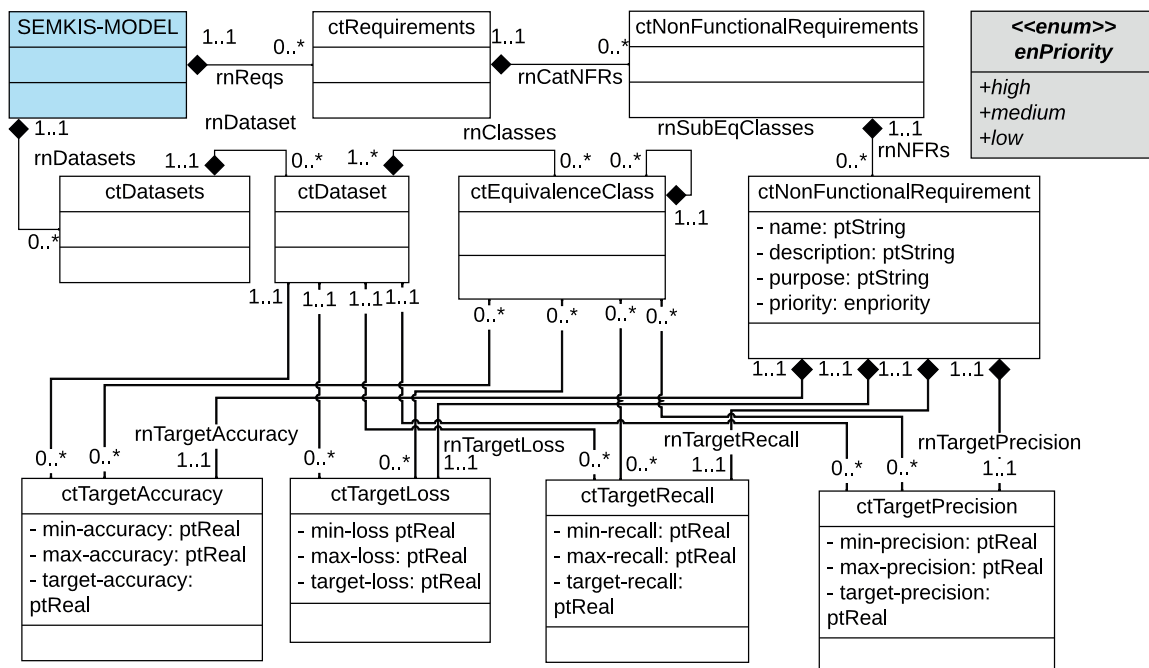


Figure 4. SEMKIS-DSL Metamodel: Nonfunctional requirements specification concepts.

Non-functional requirements (*ctNonFunctionalRequirement*) (see Figure 4) describe the expected characteristics of the system under development. Typically, these characteristics include performance, reliability, maintainability, and robustness, among others. In the context of NN engineering, engineers usually evaluate the performance, robustness, and reliability of an NN based on specific properties. We focus on the following four properties:

- Accuracy (*ctTargetAccuracy*) describes the ratio between the number of correctly recognized data by the target NN and the number of all data elements within the dataset.
- Loss (*ctTargetLoss*) is a numerical value that describes the recognition error between the target NN’s actual output and its expected output.
- Precision (*ctTargetPrecision*) describes the ratio between the number of correctly classified and recognized data and the total number of data elements.
- Recall (*ctTargetRecall*) describes the ratio between the number of correctly classified and recognized data and the number of both correctly/incorrectly classified and recognized data.

3.2.2. Model Transformation—Engineering Datasets and Neural Networks

According to the SEMKIS methodology, a software engineer analyzes the requirements specifications during the dataset and NN engineering activities in two stages. Firstly, the software engineer extracts the relevant data selection criteria from the requirements for engineering the datasets. The data selection criteria serve as input to either collect existing data or synthesize data using a program. Such a program may take the form of a traditional algorithm (e.g., manipulation of pixels) or an NN (e.g., generative adversarial network). The resulting NN can be trained and tested on engineered datasets. Secondly, the software engineer determines the NN’s architectural components (e.g., input/output layer, layer types, activation functions) from the customer’s requirements to engineer the NN’s architecture.

In this subsection, we provide the basics for the development of model-to-text transformations (i.e., generators), which includes the two aforementioned stages. Two transformations are developed:

1. Data synthesizer: generates the skeleton code for synthesizing the data.

2. Target NN architecture synthesizer: generates the skeleton code for the target NN's architecture in Python.

We identify the relevant information (e.g., data selection criteria or architectural components) that can be extracted from the previously presented requirements concepts of the SEMKIS-DSL to be taken as input for the model-to-text transformations.

For the definition of the “data synthesizer” model-to-text transformation, the dataset elements and the equivalence classes can be extracted from the functional requirements specifications. Firstly, the specification of the target NN input, `ctTargetNNInput`, defines the required data elements to be processed by the NN. Thus, we can determine some properties of the datasets such as the dataset size, dataset type, dataset format, and data types (e.g., images, numbers). Other data properties that can be extracted from the data elements include the data type, data format, and data content. Secondly, the target NN's output specification, `ctTargetNNInput`, defines the recognized equivalence classes for a given data input. From the equivalence classes, we can extract the labels of each data element in the dataset or the NN's output, including the NN's output values for formatting the dataset's labels and the interpretation of the NN's output values. The customer's recognition priorities and the recognition precision of the NN can be extracted from the non-functional requirements specifications. We can make some assumptions about the number of required data and the diversity of the data per equivalence class based on the recognition priorities and recognition precision. If an equivalence class must be recognized correctly with a high priority, we can generate additional data to strengthen the NN's recognition.

For the definition of the “Target NN architecture synthesizer” model-to-text transformation, we can determine the NN's architecture from the SEMKIS-DSL requirements specifications. From the functional requirements, we can extract some architectural components (e.g., layers, activation functions, target NN's input and output). From the non-functional requirements, we can extract the training parameters, as well as the parameters for evaluating the trained NN.

3.2.3. Running Example

To demonstrate a requirements specification using SEMKIS-DSL, we perform the SEMKIS requirements pre-activity using a running example. During this activity, the customer meets with a software engineer to order an NN that can recognize the two handwritten digits “0” and “7”. They discuss the functional and non-functional requirements of this NN. After the meeting, the software engineer is in charge of modeling the requirements specifications. Concerning the functional requirements, we specify the NN's input and output data. The functional requirements specified for the handwritten digit “7” are:

- The input data consist of 28×28 images that contain a centered and white handwritten digit “7” on a black background.
- An image is a matrix of pixels, where each pixel is represented by a grayscale value of $[0, 255]$.
- The digit 7 can be curved and may or may not contain a middle dash.

The non-functional requirements specified for the digit “7” are:

- An equivalence class named `ec7` that includes the digit “7”.
- The accuracy for `ec7` on the test dataset should reach a value of $[98\%, 100\%]$
- The accuracy of the test dataset should reach a value of $[98\%, 100\%]$.
- Both accuracies have a high priority and are important for the customer.

In Listing 1, we show some of the concrete functional and non-functional requirements specifications of the handwritten digit “7”. Note that the requirements can be easily modified and adapted to the customer's needs. The customer could wish to place higher importance on recognizing a handwritten digit “7” with a middle dash. In this case, the equivalence class `ec7` could be refined into two separate classes (`ec7-MiddleDash` and `ec7-NoMiddleDash`) to categorize images of a digit 7 with and without a middle dash.

The software engineer then has the option to specify different accuracies for each class. Thus, the number of data and the data diversity can be adapted for each class based on the new accuracies. For example, it could be useful to add more data to the equivalence class ec7-MiddleDash to strengthen the NN's recognition.

Listing 1. Sample requirements specifications for recognition of the digit "7".

```

Requirements {
  functionalRequirements {
    Requirement FR1 {
      NeuralNetworksInput {
        neuronsSize 784
        neuronsFormat [28, 28]
        inputData {
          InputData inputSeven {
            neuronValues [0; 255]
            dataElements imgDigitSeven }}}
    Requirement FR2 {
      NeuralNetworksOutput{
        neuronsSize 10
        neuronsFormat [1, 10]
        outputValueRange [0; 1]
        outputNeurons [seven]
        inputData FR1.inputSeven }}
  }
  nonfunctionalRequirements {
    Requirement NFR1 {
      description "Global accuracy on testing dataset"
      purpose "Tolerated accuracy on testing dataset"
      priority high
      TargetAccuracy globalTestAcc for dataset dstest {
        minAccuracy 98
        maxAccuracy 100
        targetAccuracy 99.7 }}
    Requirement NFR2 {
      description "Equivalence class 7 test accuracy"
      purpose "Tolerated accuracy for ec7 on the testing dataset"
      priority high
      TargetAccuracy tarEcSevenAcc for dataset dstest {
        equivalenceClasses seven
        minAccuracy 99
        maxAccuracy 100
        targetAccuracy 99.9 }}
  }
}

```

3.3. Key-Properties Specifications Using SEMKIS-DSL

In this section, we present the SEMKIS-DSL concepts related to the key- properties specifications. Key properties are used to describe the NN's acquired recognition skills after training.

3.3.1. SEMKIS Process-Test Monitoring Data Analysis

The testing phase is one of the last steps in an SE lifecycle that occurs just before the software is deployed into production for a customer. During this phase, software engineers test and analyze the software to verify if the functional and non-functional requirements have been met based on the initial specifications. The main objective of this phase is to evaluate the software's behavior to obtain customer validation for deploying the software.

In SEMKIS, we defined two activities for the testing phase. In the first activity, a software engineer tests a target NN to collect data for identifying its recognition skills. We refer to this collected data as the test monitoring data. During the testing of the NN, the testing dataset is given as input to the target NN and then the target NN’s output is analyzed. The test monitoring data may comprise quantitative values (e.g., accuracy, loss), (in-)correctly recognized equivalence classes, or graphical visualizations (e.g., confusion matrices, heat maps, categorized recognized images). In the second activity, an engineer reviews the test monitoring data and makes some observations about the target NN’s acquired recognition skills. The purpose of these observations is to understand the recognition skills and precisely specify the key properties using the SEMKIS-DSL. Then, the specifications are analyzed to verify that they are consistent with the customer’s requirements. The purpose of the key properties is to define the target NN’s recognition strengths and weaknesses, which are presented to the customer who then validates the produced target NN. If the customer validates the target NN, the software engineer proceeds to deploy the target NN. Otherwise, the software engineer proceeds to analyze the requirements and key properties to define the precise data selection criteria for preparing a dataset augmentation. Hence, the augmented datasets are used to re-train the target NN to improve its recognition skills during an additional SEMKIS process iteration. The process is iterated until the customer validates the target NN.

3.3.2. Concept Model

We introduce the concept of key properties to precisely describe the acquired recognition skills of a target NN after training. Figures 5 and 6 present the concepts related to the key properties in a UML class diagram [16]. We indicate in parentheses the associated UML class of each key-property concept presented in this section.

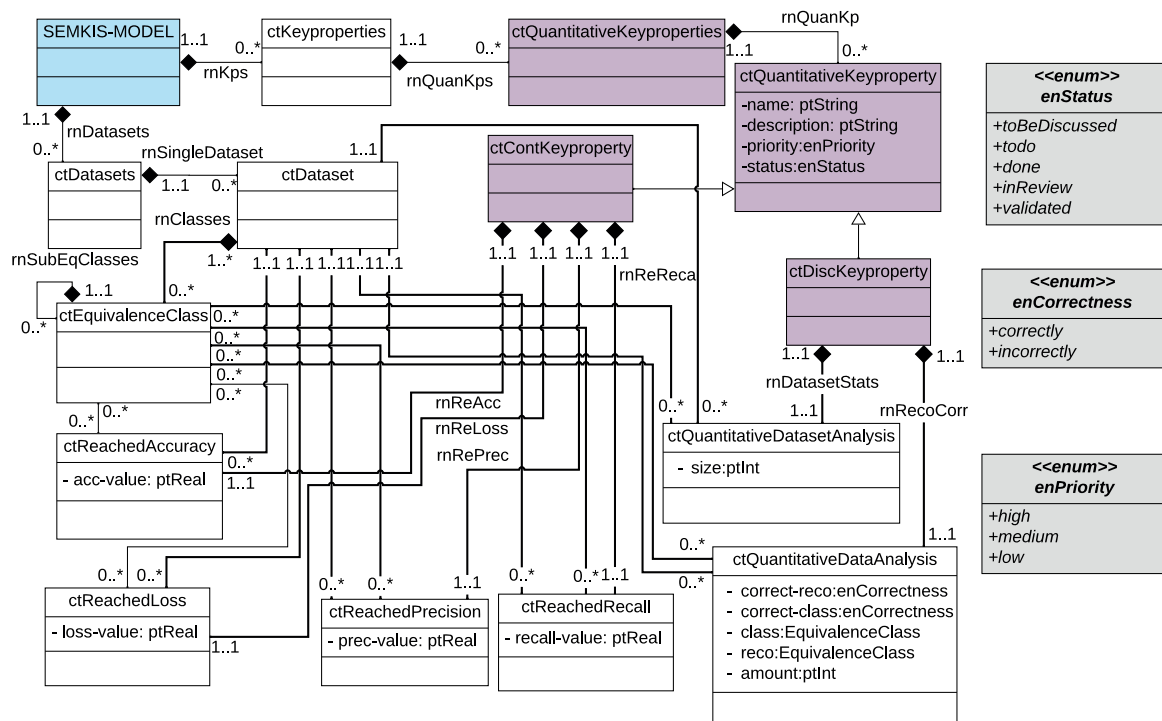


Figure 5. SEMKIS-DSL Metamodel: Quantitative key-properties specification concepts.

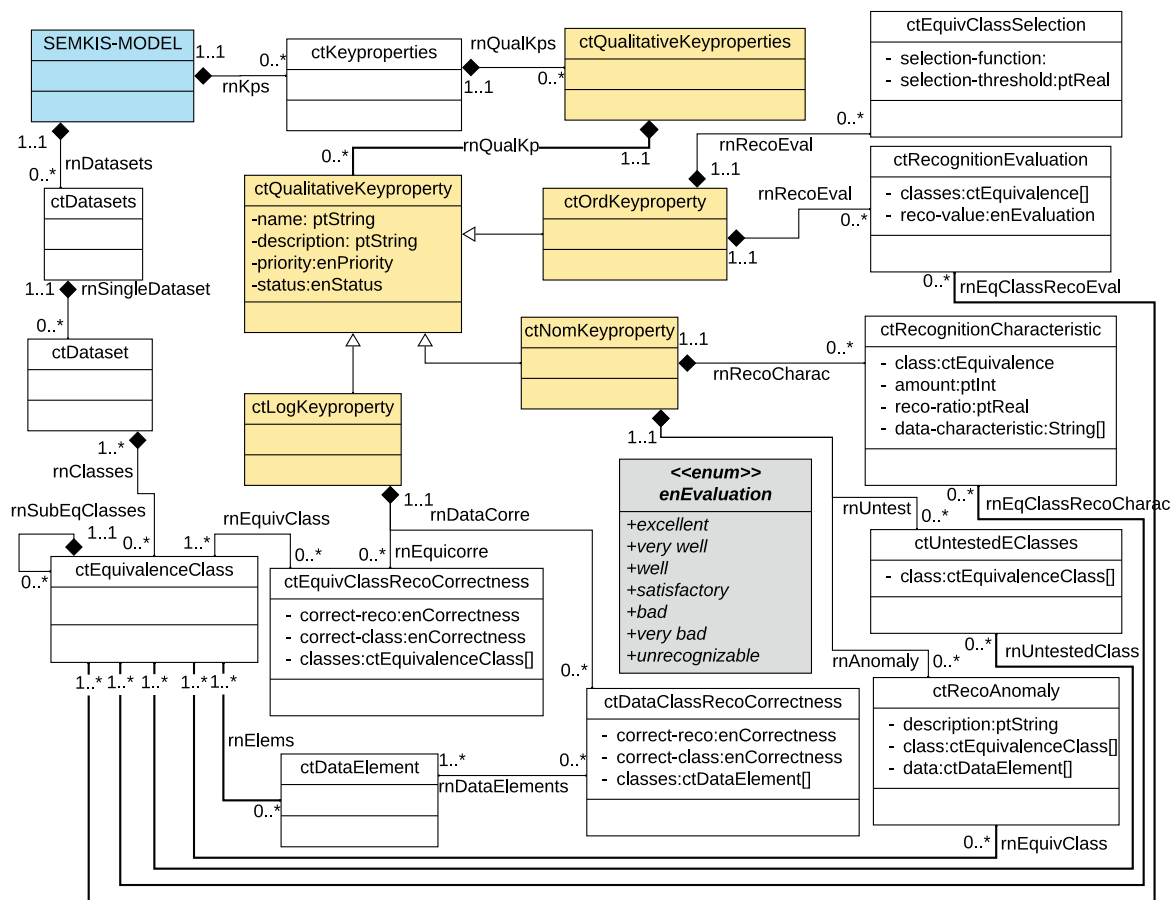


Figure 6. SEMKIS-DSL Metamodel: Qualitative key-properties specification concepts.

A software engineer specifies the target NN’s key properties (*ctKeyproperties*) as qualitative or quantitative. Each key property must be named, given a detailed description, a state (i.e., toBeDiscussed, todo, done, inReview, validated), and a priority. The quantitative key properties (*ctQuantitativeKeyproperty*) describe the characteristics of an NN that are specified as numerical values (e.g., accuracy, loss, precision). We illustrate the conceptual model related to the quantitative key properties in Figure 5. Software engineers use the quantitative key properties to describe the statistical details of the target NN’s recognition skills. We identified two types of quantitative key properties, continuous and discrete key properties.

The continuous key properties (*ctContKeyProperty*) are numerical values in a non-countable set (e.g., accuracy, loss, precision, and recall) that characterize the target NN’s recognition skills using. These characteristics specify the metrics of the acquired target NN’s recognition skills after training. We use the four non-functional requirements (i.e., accuracy, loss, etc.) to define the continuous key properties, accuracy (*ctReachedAccuracy*), loss (*ctReachedLoss*), precision (*ctReachedPrecision*), and (*ctReachedRecall*). However, at this stage, we specify the measurements of the achieved recognition skills rather than the targeted recognition skills (accuracy, loss).

The discrete key properties (*ctDiscKeyProperty*) are numerical values in a countable set (e.g., dataset size, categorizations, number of (in-)/correctly classified/recognized data) that describe a statistical analysis of the recognition of the specific data elements, datasets, and subsets, as well as data within an equivalence class. The software engineer can specify the number of data elements (in-)/correctly recognized within groups of data elements. Therefore, we defined two types of discrete key properties: quantitative data analysis (*ctQuantitativeDataAnalysis*) and quantitative dataset analysis (*ctQuantitativeDatasetAnalysis*) that include these specifications.

The qualitative key properties (*ctQualitativeKeyProperties*) are textual descriptions or boolean expressions that describe the target NN's recognition skills. We illustrate the conceptual model related to the qualitative key properties in Figure 6. The quantitative key properties precisely describe the characteristics and attributes of the target NN's acquired recognition skills. For example, they can precisely describe the recognized objects in an image, insufficiently recognized equivalence classes, any recognition anomalies, or untested equivalence classes. Software engineers can use the qualitative key properties to precisely describe the different aspects (achieved recognitions, anomalies, incorrect recognitions) of the target NN's recognition. We defined three types of quantitative key properties: nominal, ordinal, and logical.

The nominal key properties (*ctNomKeyProperty*) are textual descriptions that characterize the recognition of specific data elements or their content (e.g., images and image content). Firstly, the nominal key properties specify equivalence classes that have not been (or insufficiently been) tested. Thus, the software engineer may indicate that further tests are required to evaluate the target NN's recognition skills on an equivalence class. Secondly, the nominal key properties specify the identified recognition anomalies within the test data elements. The software engineer specifies the different characteristics that describe a problem with recognizing concrete data elements, their content, or an entire equivalence class. Thirdly, the nominal key properties specify the concrete recognition skills of the target NN. The software engineer precisely specifies the recognition of the data elements and characterizes the detected and accepted correct recognitions. Therefore, we defined three types of nominal key properties: recognition characteristics (*ctRecognitionCharacteristic*), untested equivalence classes (*ctUntestedClasses*), and recognition anomalies (*ctRecoAnomaly*).

The ordinal key properties (*ctOrdKeyProperty*) specify the different characteristics of the equivalence classes within the dataset. Firstly, the ordinal key properties are used to specify the selection criteria for a recognized equivalence class when the target NN processes a data element. The target NN often outputs a probability distribution that describes the probability that a data element has been recognized within an equivalence class. Therefore, it is necessary to define a selection criterion (e.g., maximal probability) to select the recognized equivalence class. It is important to understand how the equivalence class has been selected, as the selection criteria may have an impact on the evaluation of the target NN's recognition skills. There may be some recognition issues due to poorly chosen selection criteria. Secondly, the ordinal key properties specify the evaluation of the target NN's recognition. The software engineer specifies the different evaluation criteria (i.e., very well, bad, satisfactory, unrecognizable, etc.) that describe the level of recognition of a set of data elements. To describe these concepts, we defined two types of ordinal key properties: selection of a recognized equivalence class (*ctEquivClassSelection*) and recognition evaluation (*ctRecognitionEvaluation*).

The logical key properties (*ctLogKeyProperty*) are boolean-based values that describe the recognition accuracy of the equivalence classes, categories of data elements, entire datasets, or individual data elements. Firstly, the logical key properties are used to specify a general evaluation of the recognition of an equivalence class. The software engineer specifies whether the data elements within this equivalence class have been correctly classified and recognized. Secondly, the logical key properties are used to specify an evaluation of the recognition of specific data elements (or groups of data elements). Similarly, the software engineer specifies whether the data elements within these groups have been correctly classified and recognized. Therefore, we defined two types of ordinal key properties: the accuracy of recognizing an equivalence class (*ctEquivClassRecoCorrectness*) and the accuracy of recognizing a set of data elements (*ctDataClassRecoCorrectness*).

The resulting key-properties specifications describe the recognition strengths and weaknesses of the target NN. These strengths and weaknesses are validated by the customer in order to deploy the target NN into production. Otherwise, we proceed to dataset augmentation to re-engineer the datasets to improve the recognition skills of the target NN.

During each iteration, the key-properties specifications are updated to describe the target NN's recognition in order to be validated by the customer.

3.3.3. Model Transformation—Dataset Augmentation

According to the SEMKIS methodology, dataset augmentation is performed if the customer is yet to validate the target NN's key properties. In this case, the software engineer analyzes the key-properties specifications during the dataset augmentation specification activity. The software engineer compares the requirements and key-properties specifications to define the data selection criteria for augmenting the datasets. The purpose of the data selection criteria is to implement the data synthesizer (i.e., a traditional synthesizer program or an AI-based data synthesizer). The aim of dataset augmentation is to improve the recognition skills of the target NN by training it on the augmented dataset to meet the customer's requirements.

We do not intend to tackle the problem of dataset augmentation in machine learning in detail. Rather, we informally describe a model transformation that produces an output related to the dataset augmentation activity. The requirements and key-properties specifications serve as input for the model-to-text (M2T) transformation, which generates the skeleton code for a dataset augmentation synthesizer. Both specifications are used to extract the relevant data to be fed as input to the M2T transformation. Below, we identify the relevant input data that can be extracted to establish the basics for the development of the M2T transformation.

From the untested classes (*ctUntestedClasses*), we can extract the equivalence classes that are to be tested further. We generate additional data elements for the testing dataset to improve the evaluation of the target NN's recognition skills for an equivalence class. From the nominal key properties that characterize the recognition evaluation (*ctRecognitionEvaluation*), we can extract the poor or unrecognizable equivalence classes. We generate variations of the data elements for the training dataset to train the target NN and improve its recognition skills.

From the non-functional requirements and quantitative key properties, we can extract the amount of correctly and incorrectly recognized categories of data elements (e.g., recognized data elements within an equivalence class). The amount of correctly and incorrectly recognized data elements helps us to estimate the probability that a target NN performs a correct or incorrect recognition. Based on these results, we can generate additional data within a sufficiently correctly recognized equivalence class to strengthen its recognition. Otherwise, we can generate additional variations of the data elements to improve their recognition within the incorrectly recognized equivalence classes. Our objective is to improve the target NN's recognition in order to meet the customer's requirements. Therefore, we generate enough specific data elements by considering the recognition priorities specified in the requirements.

From the functional requirements and nominal and ordinal key properties, we can determine the recognizability of specific data elements and their content. We can compare the targeted functional requirements to the achieved recognition skills and determine the required data for improving specific recognition skills. Thus, we can generate additional data for sample data elements with specific content (e.g., images containing a specific object) to strengthen the NN's recognition. These specifications help us to focus on the prioritized requirements and, therefore, generate additional data to strengthen the NN's recognition. Finally, we can also determine the NN's architecture. From the *equivalence class selection criteria*, we can define the target NN's output (selected recognized equivalence class). Thus, when changing the selection criteria or adding more equivalence classes due to refinement, we can generate another NN architecture skeleton that is compliant with the updated requirements.

3.3.4. Running Example

To demonstrate a key-properties specification using SEMKIS-DSL, we perform the SEMKIS test monitoring data analysis activity using a running example. The test monitoring data comprise collected information about the NN's recognition after training. Figure 7 shows some results of the NN's initial training on the left-hand side. The engineer has to follow the concepts defined in the SEMKIS-DSL to identify the key properties in the testing data. Thus, he is required to qualitatively and quantitatively analyze the test monitoring data from which he can determine the key properties.

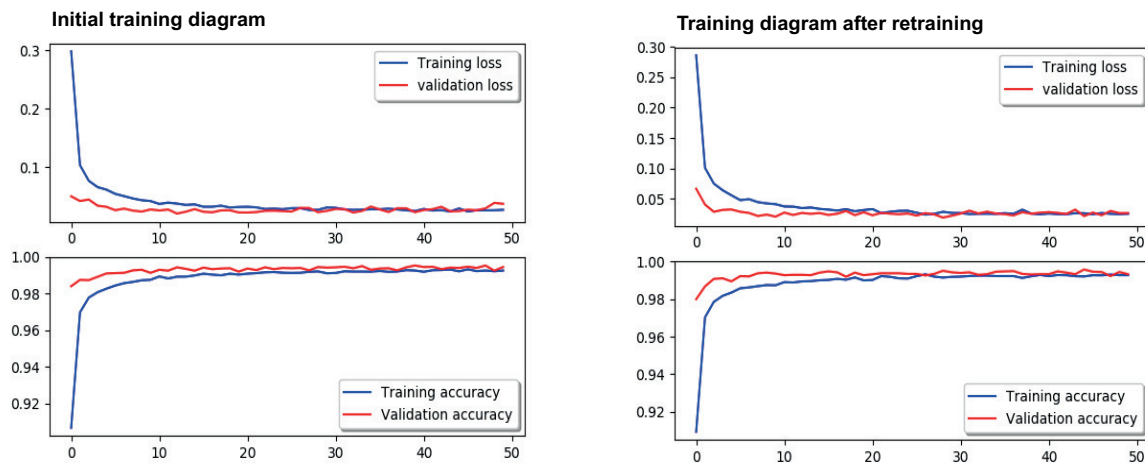


Figure 7. Training diagram of NN recognition after initial and second training.

During this activity, the software engineer analyzes the test monitoring data to make some observations about the recognition of the handwritten digits "0" and "7". From the confusion matrix (quantitative results) on the left-hand side in Figure 8, we can see that there were some issues with the recognition of the handwritten "7", whereas the recognition of "0" was acceptable. The target NN incorrectly recognized a handwritten "7" nine times and a handwritten "0" twice. On the left-hand side in Figure 9, we illustrate the incorrectly recognized images for qualitatively analyzing the NN's recognition skills. The images are ordered from "0" to "9". By analyzing the incorrectly recognized images on the left-hand side, we can confirm that "7" was often incorrectly recognized. Additionally, in Figure 10, we can see that a handwritten "2" was often recognized as a "7". We specify these results in parallel using the SEMKIS-DSL.

Concerning the incorrectly recognized "0", we can see that one image (id = 1 in left Figure 9) was incorrectly classified as it did not contain a "0". We consider this image to be unrecognizable and it will be removed from the dataset. The other image has been incorrectly recognized. Hence, the target NN resulted in only 1 error for "0", which we accept as it is compliant with the customer's requirements.

Concerning the incorrectly recognized "7", we can see that they were often recognized as the equivalence class 2 or 1. Thus, the target NN could not sufficiently distinguish between images containing a handwritten "7", "2", or "1". Since the customer requires the recognition of "7", we must improve the target NN's recognition skills. Thus, we generate additional images of a handwritten "7" to improve the target NN. The above-mentioned key properties are specified in parallel using the SEMKIS-DSL.

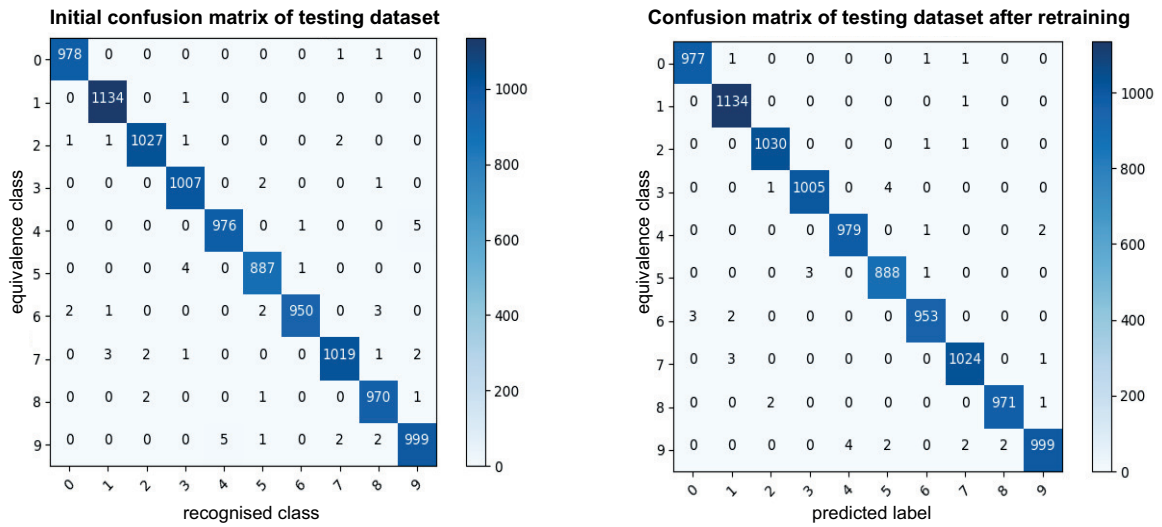


Figure 8. Confusion matrices comparing the performance of the initial and second training on the testing dataset.

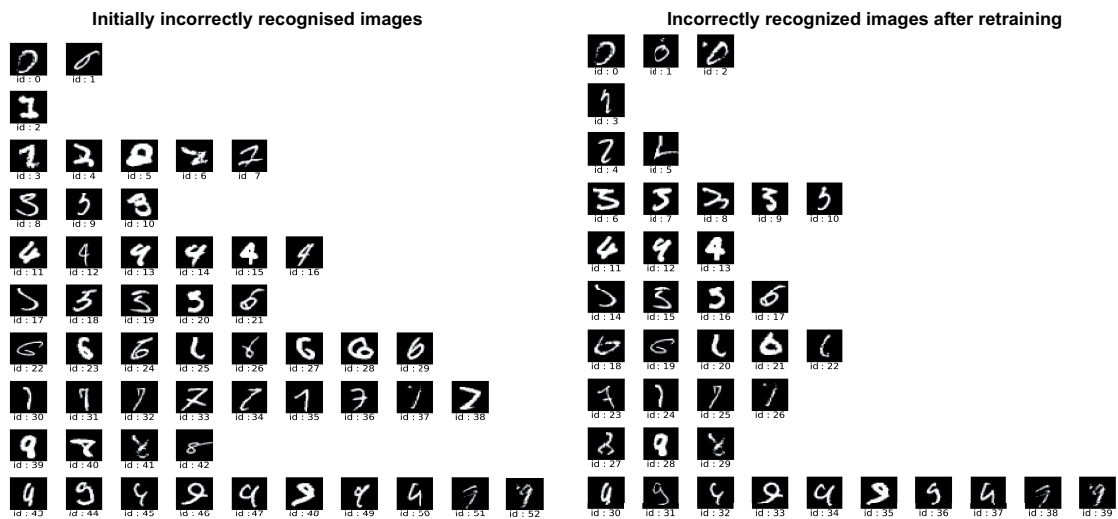


Figure 9. Incorrectly recognized test images after initial and second training.

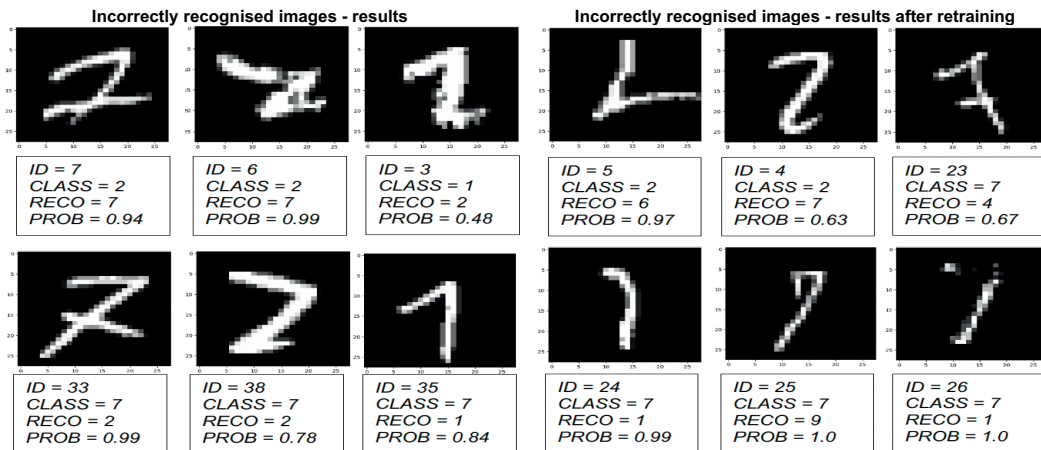


Figure 10. Details of incorrectly recognized test images after initial and second training.

In Listing 2, we show some concrete key-properties specifications for the handwritten “7”. According to the requirements and key-properties specifications, we need to improve the recognition of the handwritten “7”. Therefore, we need to augment the dataset with additional images of “7” to improve the target NN. We thus iterate the process and re-train the target NN until the target NN satisfies the customer’s requirements. The right-hand sides of Figures 7–10 show the results after re-training the NN on the augmented datasets. We can confirm that we improved the recognition of the handwritten “7”, as well as the recognition of the handwritten “2”. These new key properties can then be specified in the SEMKIS-DSL to perform an additional reiteration to improve the NN’s recognition. An example would be to improve the recognition of the handwritten “9”, which is also often incorrectly recognized.

Listing 2. Sample key-properties specifications for recognition of “7”.

```

Key Properties {
  Qualitative Key Properties {
    Nominal Key Property nkp1 {
      Recognition Anomaly recoanomaly_seven {
        description "Seven recognized often as 2 or 1"
        equivalenceClasses seven }}
    OrdinalKeyProperty okp1 {
      RecognitionEvaluation recoeval_seven {
        equivalenceClasses seven
        recognitionValue bad }}
    Logical Key Property lkp1 {
      EquivalenceClassRecognitionCorrectness erc {
        recognitionCorrectness incorrectly
        classificationCorrectness correctly
        equivalenceClasses seven }}
  }
  QuantitativeKeyProperties {
    ContinuousKeyProperty ckp2 {
      name "Recognition of equivalence class 7"
      description "Accuracy of the equivalence class 7"
      priority high
      status toBeDiscussed
      ReachedAccuracy acc_test_seven {
        equivalenceClasses seven
        accuracyValue 0.9}}
    Discrete Key Property dkp2 {
      name " Digit 7 incorrect recognition "
      description " Number of incorrectly recognized digit 7."
      priority high
      status toBeDiscussed
      incorrectly recognized QuantitativeDataAnalysis inc_reco for dataset dctest {
        equivalenceClasses seven
        amount 9}}
  }
}

```

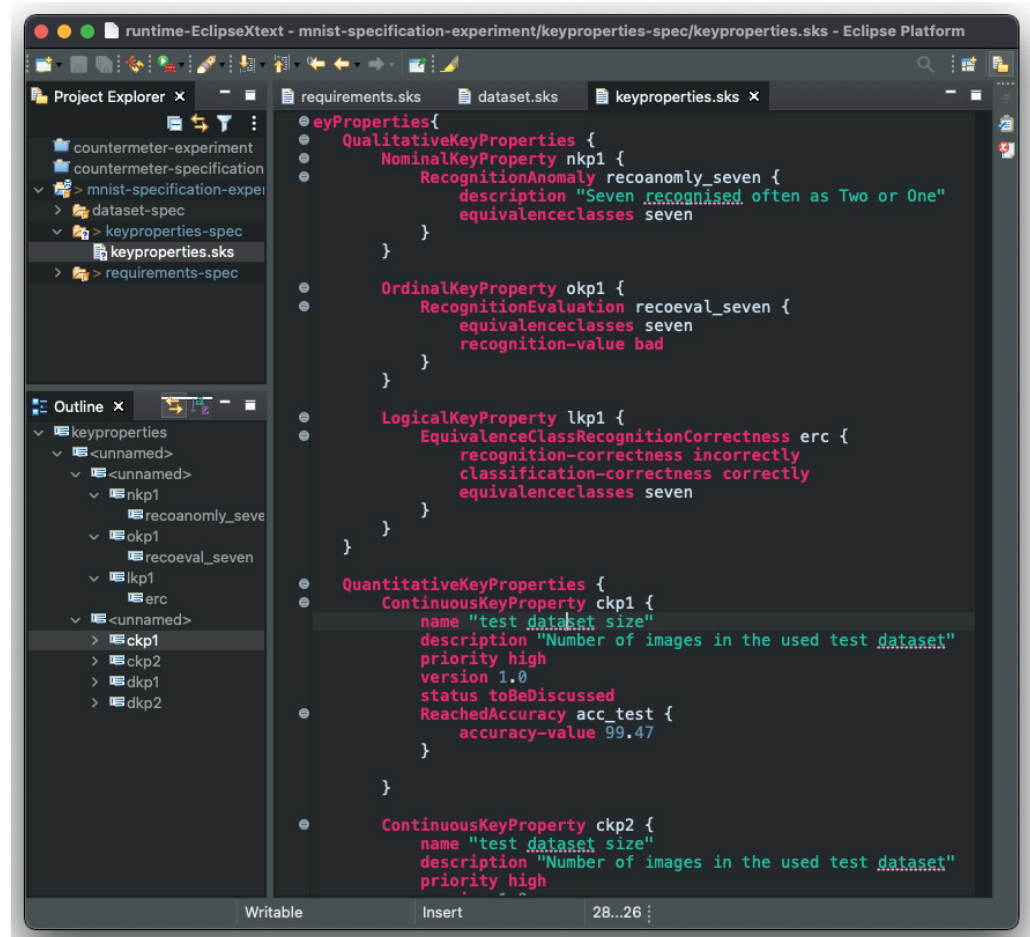
3.3.5. Tool Support

The SEMKIS-DSL has been developed using the Xtext framework [11] within the open source software engineering environment Eclipse IDE. We extended Eclipse using the Xtext plugin to develop the SEMKIS-DSL grammar in an .xtext file. The grammar comprises many syntactical rules that must be followed by the specifications in the SEMKIS-DSL. The syntactical rules define fixed keywords, optional words, relation types, user-selectable

keywords, and cross-references. These rules were determined from our conceptual models representing the NN's requirements and key properties.

In addition to these syntactical rules, the textual editor offers many features that can help software engineers efficiently write complete and structured specifications. These features can also help engineers avoid specification errors, speed up the specification process, structure their specifications, and improve the readability and traceability of the specifications by introducing a folder/file structure and supporting multiple specification views. Figure 11 shows a screenshot of the Eclipse IDE in which we specified our requirements and key properties. Other useful features include syntax highlighting, quick fixes, scoping, auto-completion, specification outlines, comparison views, and templates.

The aim of the SEMKIS-DSL and its textual editor is to guide engineers in analyzing the training monitoring data by introducing precisely defined concepts that need to be specified. Moreover, engineers can benefit from the SEMKIS-DSL to improve the readability, traceability, maintainability, and validation of their requirements and key-properties specifications.



```

eyProperties {
  QualitativeKeyProperties {
    NominalKeyProperty nkp1 {
      RecognitionAnomaly recoanomly_seven {
        description "Seven recognised often as Two or One"
        equivalenceclasses seven
      }
    }
  }
  OrdinalKeyProperty okp1 {
    RecognitionEvaluation recoeval_seven {
      equivalenceclasses seven
      recognition-value bad
    }
  }
  LogicalKeyProperty lkp1 {
    EquivalenceClassRecognitionCorrectness erc {
      recognition-correctness incorrectly
      classification-correctness correctly
      equivalenceclasses seven
    }
  }
}
QuantitativeKeyProperties {
  ContinuousKeyProperty ckp1 {
    name "test dataset size"
    description "Number of images in the used test dataset"
    priority high
    version 1.0
    status toBeDiscussed
    ReachedAccuracy acc_test {
      accuracy-value 99.47
    }
  }
  ContinuousKeyProperty ckp2 {
    name "test dataset size"
    description "Number of images in the used test dataset"
    priority high
  }
}

```

Figure 11. SEMKIS-DSL tool support [5] with sample key-properties specification.

4. Related Works

4.1. Software Development Process for Dataset Engineering

Rahimi et al. [17] provided some techniques and tools for RE in machine learning (ML). The authors aimed to improve dataset construction and thus the ML model by analyzing, understanding, and specifying the concepts of the domain problem. The objective of their approach was to create different ontologies, which structured the information collected about the domain problem, datasets, and ML model using the following three steps:

1. Benchmarking the domain via a web search to identify and specify the properties that need to be recognized by an ML model.
2. Interpreting the dataset's domain to identify problems in the collected data (e.g., risks, data inconsistencies), as inappropriate datasets may generate training issues that can result in inaccurate ML models.
3. Interpreting the domain learned by the ML model by extracting its features and using AI techniques to transform the model into more comprehensible models (e.g., a set of logical rules).

Finally, the acquired knowledge and the ontologies were compared to identify the correlations between the properties and the concepts. Thus, different quality concerns, e.g., incomplete or inconsistent requirements specifications or incompletely specified concepts could be detected. In this paper, we use the customer's requirements and NN's key properties to define the precise data selection criteria for iteratively augmenting datasets. The resulting augmented datasets are used to train an NN until it meets the requirements.

Villamizar et al. [18] analyzed 35 studies on RE in ML that were published in conferences and journals between 2018 and 2021. These studies presented guidelines, checklists, quality models, and taxonomies for RE in ML software. The authors analyzed the studies to identify the challenges and characteristics of RE in ML. The authors identified the following challenges:

- A lack of quality metrics for ML software.
- A lack of methods and tools for specifying requirements.
- The complexity of understanding non-functional requirements in ML.
- Measuring the satisfiability of the requirements.
- Customers have a limited understanding of ML.

Other researchers [1,19,20] have confirmed that requirements are becoming increasingly important in the development of AI software and that more research must be carried out. In their papers, these authors identified challenges related to the definitions of ML model performance (e.g., quality metrics, quality attributes such as reliability and availability), non-functional requirements, and the requirements related to datasets, as well as the lack of methods/tools for specifying and maintaining the requirements. We agree with the authors that RE is becoming increasingly important in the development of artificial intelligence software systems and thus for engineering ML software or NNs.

4.2. Model-Driven Engineering for Deep Learning

Hu et al. [21] proposed an approach that focused on specifying robustness requirements to build trustworthy safety-critical ML software. In this context, they designed an approach for specifying requirements that express the input/output behavior of ML software. They defined the requirements as image transformations by considering the human perception of objects (e.g., pedestrian recognition in the automotive industry). These robustness requirements consisted of formally defined image transformations (e.g., light changes, weather conditions). In our work, we address more general requirements engineering of ML-based systems. We develop a DSL intended for the SE and AI communities for the specification of requirements to further engineer improved datasets and NNs.

Several papers have presented DSLs for specifying NNs or ML software as mathematical operations or high-level textual descriptions of directed acyclic graphs. Zhao and Huang [22] presented DeepDSL, a DSL for specifying an NN as a composition of mathematical functions. Elango et al. [23] presented Diesel, a DSL for specifying the architecture of NNs using linear algebra mathematical operations. The primary objective of these DSLs is to optimize memory usage, reduce computational time, and reduce execution time when training, testing, and using an NN. StreamBrainDSL [24] was developed for specifying the architecture of NNs as layers of neurons (e.g., convolutional, fully connected layers). This DSL is embedded inside a Python framework for engineering NNs that are deployable on high-performance computing platforms. The aim of this DSL is to engi-

neer NNs that can be trained, tested, and executed on various backends to optimize their execution time. AiDSL [25] is a textual DSL for specifying the architecture of NNs. The authors developed an MDE approach for automatically generating an NN implementation in Java from a textual specification. Other DSLs have been designed to support engineers in developing various ML applications. Tensorflow Eager [26] and OptiML [27] are DSLs for designing and implementing an NN architecture. The aim of these DSLs is to facilitate the implementation of ML software using high-level descriptive languages without affecting the performance (e.g., accuracy, execution time) of the ML models. In this paper, we do not discuss DSLs that support the specification of the architecture of NNs. Neither do we consider the specification of mathematical operations or directed acyclic graphs for defining the architecture. In our work, we develop a DSL for precisely specifying the requirements and acquired recognition skills of an NN. We aim to support software engineers in precisely defining data selection criteria for improving NNs.

4.3. Model-Driven Engineering for Deep Learning Using Neural Networks for Dataset Augmentation

Ries et al. [28] introduced an MDE method to help DL scientists engineer requirements and build datasets for their DL. The method was developed as an iterative elicitation process for identifying dataset requirements. The requirements are described as formal dataset requirements concept models, which can be interpreted using MDE techniques to generate data specification instances. These instances are then analyzed to iteratively improve and validate the DRCM model. In this paper, we develop a DSL that, on the one hand, focuses on specifying the dataset requirements of an NN-based system without formal models and, on the other hand, focuses on specifying its key properties. The specifications written using our DSL can be used as input to a model transformation generating a data synthesizer.

Other papers have developed approaches or domain-specific languages for generating synthetic data to improve NNs without precise data selection criteria. Some papers [3,29,30] follow approaches for generating a large amount of data to construct large datasets to improve the training or testing of NNs without precise data selection criteria. Fremont et al. [31] introduced Scenic, a DSL that can specify digital objects in a virtual 3D space to produce images and videos. It is used to specify and generate car-driving scenarios in the domain of autonomous driving. The aim of our DSL is to specify the dataset requirements and NN-based systems' key properties for the generation of a data synthesizer. Then, according to the methodology defined in this paper, the NN dataset is augmented using synthetic data generated from the data synthesizer until it meets the customer's requirements and has been validated by the customer.

5. Discussion

In this section, we discuss some limitations, challenges to the validity of our approach, and potential research directions.

5.1. Limitations

We believe that the main limitations of the work presented in our paper concern the metamodel coverage and the automation support.

Metamodel coverage: The proposed metamodel offers only a reduced set of concepts and, more specifically, lacks the concepts needed to specify the refined accuracy and loss metrics of the NN or any other required quality metrics.

Automation: There are currently no translational semantics available for performing the proposed model-to-text transformations, which involve converting the requirements and key-properties model into a data synthesizer implementation. Therefore, this step must be carried out manually. To ensure requirements satisfiability, the NN's key properties are manually compared against the specified requirements. However, an automated module for satisfaction evaluation is currently lacking, which poses a challenge for verification and validation.

5.2. Threats to Validity

The main threat to the validity of our approach is its scalability. The utilized research methodology is susceptible to the well-known scalability problem that plagues the field of software engineering methods. In this research domain, our aim is to provide solutions that, on the one hand, have all the characteristics required by engineering for real-world application development and are supported by software engineering environments, and, on the other hand, are scientifically sound, complete, and accurate.

Given that our scope covered a broad range of topics, including the development process and domain-specific language for model-driven engineering, we focused our solution on a very limited case study that was primarily intended to illustrate and validate our approach.

The current version of our solution is far from scalable in its current form, making its validity for “real system development” questionable. This must be addressed by a serious and measurable experimental phase, which is discussed below. Despite its scalability limitations, the SEMKIS approach is still the most advanced from a software engineering perspective.

5.3. Perspectives

The main issues that we plan to address in the future concern experimentation and tool support.

- Experimentation:
 - In order to overcome the threat to the validity of our approach discussed above, the first phase of development should be to assess the SEMKIS methodology, including the SEMKIS-DSL, in an industrial context with software engineers and deep learning experts.
 - This case study should be conducted using a selection of companies and with a traditional dataset and NN engineering methods. For each of them, we propose to follow a simple yet reasonable process consisting of the following steps:
 1. Initial interviews with the software engineers to understand and discuss the current methodologies used by the company.
 2. Presentation of the SEMKIS methodology and its application to a concrete project that includes the SEMKIS-DSL.
 3. Delimitation of application modules that currently use neural networks in their products.
 4. Utilization of the SEMKIS approach for the maintenance of these modules in cooperation with SEMKIS experts.
 5. Follow-up interviews with software engineers to discuss their experience with the SEMKIS and determine its costs, complexity, usability, and maintainability.
 6. Evaluation of the data collected through experimentation and interviews.
 - The objective of this case study is to obtain enough data to assess the following aspects of the SEMKIS and its DSL:
 - * the cost/time reduction for NN and dataset engineering;
 - * the learning curve of using the SEMKIS-DSL;
 - * the maintainability and readability of the specifications in the SEMKIS-DSL;
 - * the traceability of NN’s recognition issues to the requirements specifications;
 - * the completeness, accuracy, and adequacy of the metamodel concepts.
- Tool support:
 - Another issue is the use of adequate tools, e.g., Xtend, to define and implement the model-to-text transformations to provide full automation of our MDE method for the generation of synthetic data based on requirements and key

properties. Moreover, we could extend our SEMKIS toolkit with a requirements and key-properties specification interpreter to generate a data synthesizer implementation.

6. Conclusions

In this paper, we introduce the SEMKIS domain-specific language to specify a customer's requirements and an NN's key properties. We utilize an MDE approach for automatizing data generation from the requirements and key properties specifications. In this context, our main contributions are (1) a SEMKIS-DSL conceptual model, which is used to engineer the SEMKIS-DSL grammar, for business stakeholders that contains the main concepts needed to specify the NN's requirements and key properties; (2) an approach for applying model transformations to translate these specifications into a data synthesizer or an NN architecture; and (3) an illustration of the SEMKIS-DSL using a running example, where we specify some requirements and key properties for the MNIST [10] problem.

We illustrate the SEMKIS-DSL concepts in the context of the recognition of handwritten digits. Our running example shows promising results and demonstrates the potential of our approach to support software engineers during the RE, testing, and validation phases for releasing an NN into production, as well as engineer improved NNs with appropriate datasets.

In future work, we intend to evaluate the SEMKIS methodology in an industrial environment. Our objective is to assess and validate the SEMKIS methodology, including the SEMKIS-DSL and the model transformations, with software engineers on larger projects and in an industrial context. We also intend to assess specific aspects such as the reductions in costs and time for NN and dataset engineering, the complexity of the learning curve of the SEMKIS-DSL for software engineers, and the readability and traceability of the specifications on large projects. Another idea for future work could be to extend the SEMKIS-DSL to allow for the specification of dependable NNs using the DREF framework [32]. We intend to extend our language with additional concepts to specify metrics for the resilience of NNs, as well as facilitate the specification of the satisfiability of the customer's requirements, including some tolerance margins. These metrics will allow us to measure the satisfiability evolution of the customer's requirements after each updated version of an NN. The software engineer would then be able to analyze the metrics and define new data selection criteria to improve the datasets for re-training the NN. The resulting NNs would then meet the customer's requirements within the specified tolerance.

Author Contributions: All authors contributed to the study's conception and design. Material preparation, data collection, and analysis were performed by B.J. The first draft of the manuscript was written by B.J. B.R. made some revisions to the first draft of the manuscript and all authors commented on the previous versions of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: The authors did not receive funds, grants, or other support from any external funding agency in the public, commercial, or not-for-profit sectors for the submitted work.

Data Availability Statement: The data generated and/or analyzed during the current study are available in the following public Github repository; <https://github.com/Benji91/lu.uni.lassy.phdthesis.semkis.toolkit.experimentations>, (accessed on 28 March 2023).

Conflicts of Interest: The authors have no competing, financial, non-financial, or other interests to disclose.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
DL	Deep Learning
DSL	Domain-Specific Language
MDE	Model-Driven Engineering
ML	Machine Learning
M2T	Model-to-Text Transformation
NN	Neural Network
RE	Requirements Engineering
SE	Software Engineering
SEMKIS	Software Engineering Methodology for the Knowledge Management of Intelligent Systems
UML	Unified Modeling Language

References

- Heyn, H.M.; Knauss, E.; Muhammad, A.P.; Eriksson, O.; Linder, J.; Subbiah, P.; Pradhan, S.K.; Tungal, S. Requirement engineering challenges for ai-intense systems development. In Proceedings of the 2021 IEEE/ACM 1st Workshop on AI Engineering-Software Engineering for AI (WAIN), Online, 30–31 May 2021; pp. 89–96.
- Xiao, T.; Xia, T.; Yang, Y.; Huang, C.; Wang, X. Learning from massive noisy labeled data for image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA, USA, 7–12 June 2015; pp. 2691–2699.
- Ros, G.; Sellart, L.; Materzynska, J.; Vazquez, D.; Lopez, A.M. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 3234–3243.
- You, Q.; Luo, J.; Jin, H.; Yang, J. Building a large scale dataset for image emotion recognition: The fine print and the benchmark. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.
- Jahic, B. SEMKIS: A Contribution to Software Engineering Methodologies for Neural Network Development. Ph.D. Thesis, University of Luxembourg, Esch/Alzette, Luxembourg, 2022.
- Jahić, B.; Guelfi, N.; Ries, B. Software engineering for dataset augmentation using generative adversarial networks. In Proceedings of the 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 18–20 October 2019; pp. 59–66.
- Jahić, B.; Guelfi, N.; Ries, B. Specifying key-properties to improve the recognition skills of neural networks. In Proceedings of the 2020 European Symposium on Software Engineering, Online, 6–8 November 2020; pp. 60–71.
- Fondement, F.; Silaghi, R. Defining model driven engineering processes. In Proceedings of the Third International Workshop in Software Model Engineering (WiSME), Lisbon, Portugal, 11–15 October 2004.
- Kosar, T.; Marti, P.E.; Barrientos, P.A.; Mernik, M. A preliminary study on various implementation approaches of domain-specific language. *Inf. Softw. Technol.* **2008**, *50*, 390–405. [[CrossRef](#)]
- LeCun, Y.; Cortes, C.; Burges, C.J.C. The MNIST Database of Handwritten Digits. 2022. Available online: <http://yann.lecun.com/exdb/mnist/> (accessed on 28 March 2023).
- Bettini, L. *Implementing Domain-Specific Languages with Xtext and Xtend*; Packt Publishing Ltd.: Birmingham, UK, 2016.
- Jahič, B. SEMKIS-DSL Complete Grammar (v2.0). 2022. Available online : <https://github.com/Benji91/lu.uni.lassy.phdthesis.semkis.toolkit.experimentations/blob/main/eclipse-workspace-semkis-dsl/lu.uni.lassy.phd.dsl.semkis.parent/lu.uni.lassy.phd.dsl.semkis/src/lu/uni/lassy/phd/dsl/semkis/Semkis.xtext> (accessed on 28 March 2023).
- Jahič, B. SEMKIS-DSL (v2.0). 2022. Available online: <https://github.com/Benji91/lu.uni.lassy.phdthesis.semkis.toolkit.experimentations> (accessed on 28 March 2023).
- Sommerville, I. *Engineering Software Products*; Pearson: London, UK, 2020; Volume 355,
- Ruparelia, N.B. Software development lifecycle models. *ACM SIGSOFT Softw. Eng. Notes* **2010**, *35*, 8–13. [[CrossRef](#)]
- Object Management Group B. *Unified Modeling Language 2.5.1*; Object Management Group: Milford, MA, USA, 2017.
- Rahimi, M.; Guo, J.L.; Kokaly, S.; Chechik, M. Toward requirements specification for machine-learned components. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), Jeju Island, Republic of Korea, 23–27 September 2019; pp. 241–244.
- Villamizar, H.; Escovedo, T.; Kalinowski, M. Requirements engineering for machine learning: A systematic mapping study. In Proceedings of the 2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Online, 1–3 September 2021; pp. 29–36.
- Habibullah, K.M.; Horkoff, J. Non-functional requirements for machine learning: Understanding current use and challenges in industry. In Proceedings of the 2021 IEEE 29th International Requirements Engineering Conference (RE), Notre Dame, IN, USA, 20–24 September 2021; pp. 13–23.

20. Vogelsang, A.; Borg, M. Requirements engineering for machine learning: Perspectives from data scientists. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference Workshops (REW), Jeju Island, Republic of Korea, 23–27 September 2019; pp. 245–251.
21. Hu, B.C.; Salay, R.; Czarnecki, K.; Rahimi, M.; Selim, G.; Chechik, M. Towards requirements specification for machine-learned perception based on human performance. In Proceedings of the 2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE), Zurich, Switzerland, 1 September 2020; pp. 48–51.
22. Zhao, T.; Huang, X. Design and implementation of DeepDSL: A DSL for deep learning. *Comput. Lang. Syst. Struct.* **2018**, *54*, 39–70. [[CrossRef](#)]
23. Elango, V.; Rubin, N.; Ravishankar, M.; Sandanagobalane, H.; Grover, V. Diesel: DSL for linear algebra and neural net computations on GPUs. In Proceedings of the 2nd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, Philadelphia, PA, USA, 18 June 2018; pp. 42–51.
24. Podobas, A.; Svedin, M.; Chien, S.W.; Peng, I.B.; Ravichandran, N.B.; Herman, P.; Lansner, A.; Markidis, S. StreamBrain: An HPC DSL for Brain-like Neural Networks on Heterogeneous Systems. In Proceedings of the 11th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies, Online, 21–23 June 2021.
25. Cueva-Lovelace, J.M.; García-Díaz, V.; Pelayo, G.; Bustelo, C.; Pascual-Espada, J. Towards a standard-based domain-specific platform to solve machine learning-based problems. *Int. J. Interact. Multimed. Artif. Intell.* **2015**, *3*, 6–12.
26. Agrawal, A.; Modi, A.; Passos, A.; Lavoie, A.; Agarwal, A.; Shankar, A.; Ganichev, I.; Levenberg, J.; Hong, M.; Monga, R.; et al. TensorFlow Eager: A multi-stage, Python-embedded DSL for machine learning. *Proc. Mach. Learn. Syst.* **2019**, *1*, 178–189.
27. Sujeeth, A.; Lee, H.; Brown, K.; Rompf, T.; Chafi, H.; Wu, M.; Atreya, A.; Odersky, M.; Olukotun, K. OptiML: An implicitly parallel domain-specific language for machine learning. In Proceedings of the 28th International Conference on Machine Learning (ICML-11), Bellevue, WA, USA, 28 June–2 July 2011; pp. 609–616.
28. Ries, B.; Guelfi, N.; Jahic, B. An mde method for improving deep learning dataset requirements engineering using alloy and uml. In Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development, SCITEPRESS, Online, 8–10 February 2021; pp. 41–52.
29. Dosovitskiy, A.; Ros, G.; Codevilla, F.; Lopez, A.; Koltun, V. CARLA: An open urban driving simulator. In Proceedings of the Conference on Robot Learning, Mountain View, California, USA, 13–15 November 2017; pp. 1–16.
30. Pei, K.; Cao, Y.; Yang, J.; Jana, S. Deepxplore: Automated whitebox testing of deep learning systems. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, 28–31 October 2017; pp. 1–18.
31. Fremont, D.J.; Dreossi, T.; Ghosh, S.; Yue, X.; Sangiovanni-Vincentelli, A.L.; Seshia, S.A. Scenic: A language for scenario specification and scene generation. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Phoenix, AZ, USA, 22–26 June 2019; pp. 63–78.
32. Guelfi, N. A formal framework for dependability and resilience from a software engineering perspective. *Open Comput. Sci.* **2011**, *1*, 294–328. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.