A framework for Requirements specification of machine-learning systems

Xi Wang[*][†] and Weikai Miao[‡][✉]
[*]School of Computer Engineering and Science, Shanghai University, China
[†]Shanghai Key Laboratory of Computer Software Testing and Evaluating, Shanghai 201114, China
[‡]Software Engineering Institute, East China Normal University, China

***Abstract*—The rapid development of machine learning (ML) systems has raised many concerns over their quality. Due to the inherent complexity and uncertainty, most of the traditional quality assurance techniques have been challenged, including requirements specification. Current strategies mainly focus on model extraction from existing neural networks to improve interpretability and facilitate system analysis, but failing to include user expectations on the system. To handle the problem, this paper proposes a specification framework for ML requirements where each ML system is regarded as a set of snapshot systems along the evolvement process. There are 3 layers in the framework and the hierarchy indicates that higher-level models need to be built based on lower-level ones. The bottom layer consists of meta snapshot model and meta data model serving as the meta models for snapshot systems and data requirements respectively. The middle layer is for snapshot models each describing a snapshot system through relations between its outputs produced with different inputs. The top layer is a learning model capturing the evolvement process by transitions among snapshot models. These transitions are activated by data models instantiated from meta data model. We adopt the specification of a self-driving system to illustrate the framework.**

I. Introduction

Machine-learning (ML) system has gained much attention recent years not only for its thrilling achievement on training intelligent machines but also for the challenging difficulties on quality assurance [1]. These difficulties mainly come from the complexity and uncertainty of ML systems since these systems would adjust their functions themselves through mechanisms difficult to be understood by human. Challenges are proposed to all kinds of traditional software quality assurance methods including requirements modeling [2].

As the key activity in the early stage of software development, requirements modeling intends to clarify and describe expected behaviors in a requirements specification which serves as guidance for implementation and basis for verification. Traditional methods are suitable for deterministic functionalities but can hardly tackle ML systems that constantly evolve and produce unpredictable results [3]. Some solutions have been proposed and most of them focus on extraction algorithms for constructing system model from ML models [4]. Since their major goal is to facilitate human understanding and semantic analysis of existing ML systems by interpreting their learning behaviors with easy-to-digest notations, user expectation on the ML systems is not involved in the derived model and the system requirements remains unclear.

Several attempts were made to deal with user requirements for ML systems. Their most concerned issue is the performance of the trained ML models and how it can be defined and measured [5]. Once these non-functional requirements are specified, adversarial examples can be accordingly designed and used as training data to re-train the ML models and improve their performance. Data set has also gained much attention as it serves as the key element for distinguishing ML systems from traditional ones. Researchers are pursuing standards for evaluating data set quality and guidelines for deriving high-qualified data set. Besides, there are a few works on supporting decision making during the requirement stage, such as the selection of ML models and activation functions. Necessary requirement information is captured through domain analysis and suggested decisions will be accordingly provided. All these attempts made significant progress in requirements modeling for ML systems from different aspects, but there is still lack of systematic method for specifying system behaviors and their relation with data sets.

To this end, this paper proposes a systematic framework for requirements specification of ML systems. It adopts feature-oriented analysis method to specify the included models based on features and explains the evolvement process of the ML system as a set of snapshot systems each representing a state of the ML system during the process. Requirements on data sets are also included as triggers for snapshot system behaviors and the transitions among them.

Specifically, the framework is composed of 3 layers and lower-level models are the basis for higher-level models. The bottom layer includes 2 meta models. One is meta snapshot model with environment and system elements for describing the environment and internal states of the ML system respectively. The other is meta data model with built-in and learning-relative elements for describing data attributes independent from and dependent on the ML system respectively. These elements are organized in feature models that illustrate their hierarchical relations.

The middle layer consists of the snapshot systems necessary to be specified. Due to the difficulty in obtaining expected outputs from a snapshot system, we turn to the output relation strategy inspired by metamorphic testing [6]. Instead of modeling system behaviors through relations between input and output, this strategy pays attention to the change of outputs caused by the change of inputs. It describes a snapshot system

DOI reference number: 10.18293/SEKE2022-0143

as a set of relations based on meta models where each relation connects two outputs produced with a input data model and its variant respectively.

The top layer is a learning model describing the learning behavior by transitions among the snapshot systems. Each transition is labeled with a data model instantiated from its meta model, indicating that the original system will be evolved into the destination system by learning from the data model. Each snapshot system can either be modeled using output relation strategy or via its relation with the original system it evolves from.

Our framework covers the important aspects of ML requirements and deals with their complicated relations with hierarchy. An example self-driving system is introduced to illustrate the technical details within each layer.

The remainder of this article is organized as follows. Section II summarizes the related work. Section III explicitly describes the requirements modeling framework for ML systems with an example self-driving system. Section IV concludes the paper and discusses some future works.

II. Related Work

There are mainly 2 kinds of modeling strategies for ML systems. The first kind extracts simpler models from ML models to enable human understanding and the application of existing analysis methods, since ML models are usually complex and difficult to be understood and analyzed. In [7], guidance from RNN's step-wise predictive decisions and context information is provided for extracting weighted automata from neural network. In [8], a method for learning an FSA from a trained RNN model is given where the RNN is first trained and an FSA is learned based on the clustering result of all hidden states. In [4], probabilistic automata is extracted from RNN in a request-specific way and the experiment shows the accuracy and scalability of the method. In [9], the knowledge hidden in pre-trained CNN is interpreted by explanatory graph where different part patterns are disentangled from each filter of CNN in an unsupervised manner. Since the above works are conducted on already implemented ML systems for revealing their behaviors from black-box, they paid little attention to user requirements on the system and lack effective mechanisms for requirements specification. By contrast, our approach focuses on requirements modeling for ML systems and provides description framework from various aspects.

The other kind aims at modeling ML requirements from certain perspectives. In [10], goal-oriented requirements analysis method is extended as evidence-driven method where many aspects of decision making remain as hypotheses until being validated or invalidated by experiments, field tests and operation. It is conducted at goal level without touching concrete functions of ML systems. In [11], a logical approach is given for specifying statistical properties of ML systems based on a Kripke model. It includes formal notations for robustness and fairness of classifiers, as well as relations among properties of classifiers. In [12], definition of fairness is provided for an effective and scalable automatic testing method of ML systems in the domain of text classification. In [13], conventional quality characteristics is extended for ML systems with its measuring method and a method for requirements identification is proposed to derive quality characteristics and measurement method. In [14], a data-driven engineering process is proposed to link the operational design domain with the requirements on data sets at different levels. These researches mainly concentrate on non-functional and data requirements of ML systems, but are still incapable of specifying their overall expected behaviors. Our framework intends to bridge the gap by specifying ML systems at 3 different layers in a systematic way to cover both snapshot and learning behaviors.

III. Framework for modeling ML systems

Comparing to traditional software which determines its behavior with programs, ML systems learn from training data set and evolve themselves towards the target goals. This fact makes it impossible to adopt traditional modeling method in specifying ML systems as it is only suitable for predictable behaviors. Customized modeling method can only be established when differences between two kinds of software systems are fully identified. If we take snapshots for a ML system along its evolving process, a set of simpler systems can be captured each representing one state of the ML system reached by learning. The transitions among these states are triggered by training data, indicating the learning path of the ML system. According to the above analysis, there are 3 aspects to be included in the requirements of ML systems: snapshot systems for solving the target problem at different time points, learning behaviors for transiting among different snapshot systems and training data for activating the learning behaviors. To enable the specification of all these 3 aspects for ML systems, a requirements modeling framework with 3 layers is given as shown in Fig. 1.

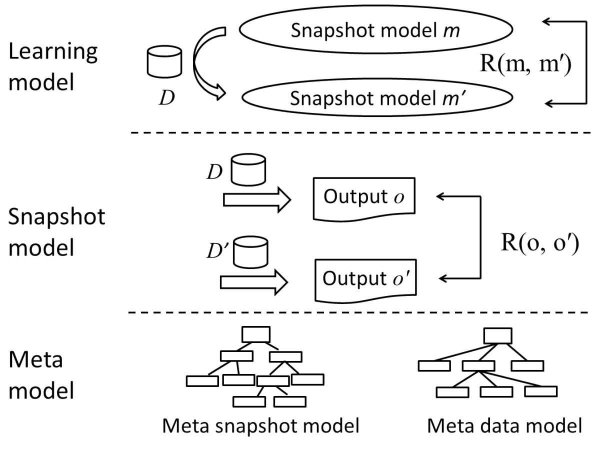

Fig. 1. The requirements modeling framework for ML systems

The hierarchy indicates that upper-level models need to be built based on lower-level ones. The bottom level consists of two meta models: meta snapshot model as meta model for

DOI reference number: 10.18293/SEKE2022-0143

describing snapshot systems and meta data model as meta model for describing training or testing data set. They provide general patterns to build various concrete snapshot and data models for the higher layers.

The middle level adopts the idea of metamorphic relation to describe concrete snapshot systems since the expected output is almost impossible to determine. In stead of modeling the relation between input and output data as in tradition method, we turn to the relation $R(o, o')$ between expected output o and o' produced by the snapshot system with input data D and D' respectively. For each pair of different input data D and D', either corresponding concrete data models or their relations need to be established based on meta data model. With the common elements from meta snapshot model, relation $R(o, o')$ can be described as properties of relevant elements.

The top level deals with learning behaviors by describing the evolvement of the ML system based on training data sets. Although learning process is continuous, expected requirements can be reflected by important transitions each representing a measurable change of the ML system made by learning from certain kind of data set. A transition connects two snapshot models m and m' before and after the corresponding change and is labeled with a data model D instantiated from its meta model. It indicates the learning process where the ML system will be transited from snapshot system m to m' if being trained with data set D. The change made by D can also be described as a relation $R(m, m')$ between m and m' if necessary.

We will present the details of each layer with an example of self-driving system which fully controls the movement of vehicles and evolves by learning from animation on labeled route and driving scenarios.

Meta model

The bottom-level meta snapshot model and meta data model aim at capturing the basic elements for composing concrete snapshot model and data model. We adopt Feature-oriented Analysis Method to specify these elements with feature model illustrated as tree structure [15]. Each node of the tree represents an element and its children nodes decompose the element into lower-level elements. There are 4 kinds of children nodes: *mandatory* element, *optional* element, *alternative* elements where only one element can be included and *or* elements where at least one element must be included. There are 2 kinds of dependency relations among elements: *requires* indicating the inclusion of certain element requires for the inclusion of another one and *excludes* indicating that only one of the two elements can be included.

Meta snapshot model consists of environment and system elements. The former indicates the factors that would affect system behaviors from outside of the system and the latter indicates system variables determining the state of the system. Fig.2 gives the partial feature model of the environment elements for the example self-driving system. It shows 4 of the mandatory environment elements with some of their children elements: the weather condition, the local policy such as driving on left side and other governmental policies, the brightness, the road scenes such as the traffic signs erected above roads to give instructions, jam or intersection scenarios and various obstacles.

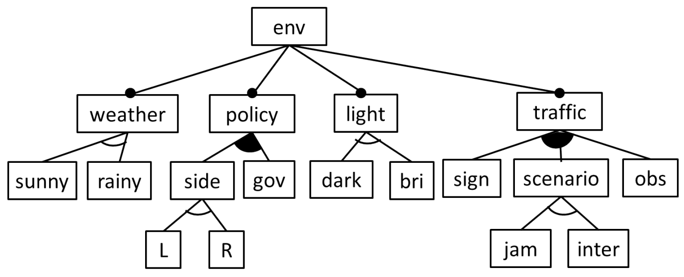

Fig. 2. The feature model for environment elements of the self-driving system

Fig.3 shows the partial feature model of the system elements for the self-driving system. To specify the system behavior, its steering angle, velocity and route are 3 of the elements that must be clarified. Two of the optional elements performance and windshield-wiper will be specified as various metrics and wiping intervals respectively. Two example metrics are given: latency of the behavior and the accuracy of object detection and trajectory prediction. The element *detect* is defined as a mapping $obj \rightarrow a$ denoting the accuracy a of detecting object obj.

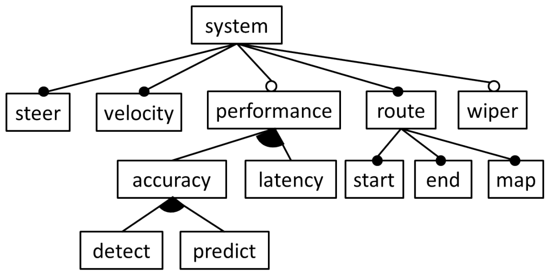

Fig. 3. The feature model for system elements of the self-driving system

Dependency relations can also be established among the above elements. For example, elements *rainy* requires for *wiper* since the windshield-wiper needs to be turned on for rainy days.

Meta data model includes built-in and learning-relative elements which represent data attributes independent from or depending on the specific learning system respectively. A concrete data model can be obtained by specifying the values of or properties on these elements. For built-in elements, one unified feature model is sufficient since they are shared by concrete data models fed to different ML systems. But for learning-relative elements, different feature models need to be built for different learning systems since their definitions are given based on the specific ML systems.

The partial feature model of built-in elements is given in Fig.4 with one of the optional elements and three of the mandatory elements. For each data set, we can decide whether to specify its collection and expire date but must at least clarify the representation of the involved data, the source of

DOI reference number: 10.18293/SEKE2022-0143

the data and its labeling method. Four example representations are given and each representation is attached with its own attributes, such as the size and resolution of image data. With the rapid development of Multi Modal Machine Learning, more and more data sets are provided in multiple representations. Our feature model allows for multiple representations through *or* children elements of *representation* where *relation* denotes the correspondence between data in different representations. For example, multi-modal data set *nuScenes* includes images from camera, pointclouds from Lidar, the returns from Radar sensors and human annotated semantic map for the same obstacles to train self-driving systems with complementary data [16].

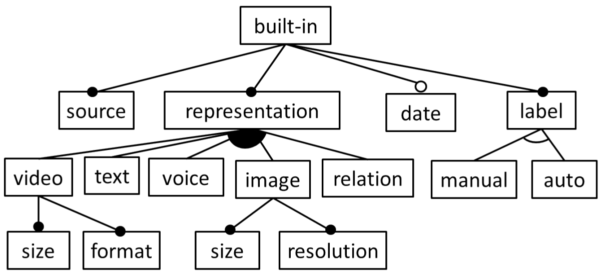

Fig. 4. The feature model of meta data model built-in elements

Learning-relative elements are created for each specific ML system since learning-relative requirements on data sets depends on what to be learned from them. Involving essential environment and system elements of functions to be learned for solving the target problem, meta snapshot model serves as the basis for achieving learning-relative elements. Fig. 5 provides a feature model template for learning-relative elements of specific ML systems. The semantic of a data set must be specified by a set of pairs $es-pair_1, ..., es-pair_n$ each $es-pair_i = (Prop_{env}, Prop_{system})$ representing an environment-system pair where $Prop_{env}$ indicates an instantiated environment model and $Prop_{system}$ indicates a concrete system model. With well-defined environment and system state, each pair corresponds to a kind of scenarios and the universal set of pairs is able to capture all the behaviors to be learned from the data set by the ML system. Optional element *metrics* represents measurements on the data set including one or more of its attributes.

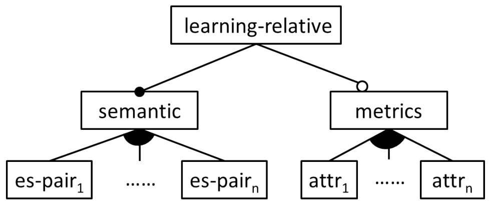

Fig. 5. A template for the feature model of meta data model learning-relative elements

For the example self-driving system, the semantic of its data sets is interpreted through property pairs on its own environment and system elements. Domain-specific attributes are attached to element *metrics*: *diversity* measuring dissimilarity among closest and furthest samples, *distribution* measuring the distribution of data samples in the context of certain features, *complexity* measuring the complexity of objects within the traffic scenes.

It should be noted that the presented example meta models are only partially given and new domain-dependent elements may need to be introduced as the relevant domain develops. These meta models should be maintained by analyst and domain experts to provide advanced information for concrete model construction.

Snapshot model

A snapshot system can be regarded as one of the states of the ML system. It takes a data set as input and transfers itself to a new system state. Since the input data provides specific settings on environment and system initialization, its data model contains only one environment-system pair.

Theoretically, the behavior of a snapshot system is deterministic because of the pause of the evolvement at that time point. However, instead of executing under man-made instructions, the snapshot system leads its own complicated implementation process with the given training data, making itself a black-box difficult to be understood and analyzed. Even at requirements stage, expected functions remain unclear because they are hidden in the training data. In most cases, we found ourselves trapped in a dilemma where solutions achieved by ML systems cannot be interpreted but need to be modeled and verified. Although we could simply define the relation between inputs and outputs in some cases, such as correctly recognizing or never crashing on a pedestrian, but such requirement contains little effective information and can hardly contribute to system verification.

Inspired by metamorphic testing, we revealed users' expectation on system behaviors when making certain change to input data and thus define snapshot model as follows.

Definition 1. Given a snapshot system S with its input domain DM_S consisting of data models, the snapshot model of S is a set $R_1^{T_1}, ..., R_n^{T_n}$ and each $R_i^{T_i}$ represents a relation formulation $\forall_{D \in DM_S} \cdot S(D^{T_i}) = [S(D)]^{T_i'}$ where T_i denotes a kind of transformational relation between data models, D^T denotes the data model obtained by performing transformation T to data model D, $S(D)$ denotes the system state of S after receiving data model D, and $[s]^T$ denotes the state achieved by conducting transformation T on the original state s.

According to the definition, each snapshot system is modeled as a set of behavior relations between different system states caused by transformations on the original input data models. For each behavior relation, the change of any input data model by the given transformation should result in system state reached from its original state with the same transformation. Transformation on data model is described based on the built-in and learning-relative elements from meta data model while transformation on system state is described based on system elements from meta snapshot model.

DOI reference number: 10.18293/SEKE2022-0143

We will take the self-driving system as an example to illustrate the definition. Assume the system has evolved to certain snapshot version S, the following are some example relations established for the corresponding snapshot model. Note that we use $\tau(m, <e>)$ to denote the value of element e within model m and $\sigma(m, <e>)$ to denote a model different from m in the way they specify element e.

Exchange of start and end point: As one of the key functions in self-driving system, navigation component guides the vehicle from start to the end point with the generated route. It is difficult to specify our requirement on the resultant route, but we expect the fact that exchanging the start and end point should result in a new route similar to the original one under the same traffic condition. The corresponding relation formulation is given as follows.

$$\begin{aligned}&\forall_{D,D^T \in DM_S} \cdot \tau(D, <start>) = \tau(D^T, <end>)\\ &\quad \wedge \tau(D, <start>) = \tau(D^T, <end>)\\ &\Rightarrow dist(\tau(S(D), <map>), \tau(S(D^T), <map>)) < \epsilon\end{aligned}$$

The difference between route map and map' is measured by function $dist(map, map')$ and the threshold ϵ needs to be given by domain expert.

Change of weather : This relation can be established for driving scenarios under different weather conditions. Although some system elements maybe specified in different ways under different weather conditions such as *wiper*, but there should not exist long distance between steering angles since steering angle largely depends on road scenes rather than weather conditions. The relation formulation is given as follows.

$$\begin{aligned}&\forall_{D \in DM_S} \cdot | \, \tau(S(D), <steer>) -\\ &\quad \tau(S(\sigma(D, <weather>), <steer>) \, | < \epsilon\end{aligned}$$

The threshold ϵ measures the distance between steering angles in the systems states led by different data models and needs to be given by domain expert.

Perturbations to traffic signs: Correct recognition of traffic signs is crucial to the safety of self-driving vehicles and perturbations to them should not affect the recognition result. This is also an important and verifiable property for the detection of pedestrians and other obstacles, we will take traffic signs as an example for illustration. The corresponding formulation is given as follows.

$$\begin{aligned}&\forall_{D,D^T \in DM_S} \cdot dist(\tau(D, <sign>), \tau(D^T, <sign>)) < \epsilon\\ &\Rightarrow S(D) = S(D^T)\end{aligned}$$

If the distance between the original sign and the perturbed one is less than the given threshold ϵ, the behavior of the snapshot system should be consistent.

Learning model

Based on the definition of snapshot models, the top-level learning model can be regarded as a state transition diagram where each state represents a snapshot system and each transition represents the evolvement of the ML system activated by training data. The formal definition is given as follows.

Definition 2. The learning model of a ML system is a 4-tuple (M, s_0, DM, δ) where M is a non-empty set of snapshot models, $s_0 \in M$ is the model of the initialized ML system, DM is the universal set of involved data models and $\delta : M \times DM \rightarrow M$ is the transition function relating two snapshot models by a data model.

Connecting the identified snapshot models by transitions labeled with data models, the learning model traces the expected learning process of the corresponding ML system. Each snapshot model can either be defined as a set of relations between outputs produced by different inputs or a relation to its original model. Each data model is specified as a set of properties on its meta data model.

Fig. 6 gives a partial learning model for the self-driving system. It includes 4 snapshot models as critical learning points and 3 data models as learning materials contributing to the evolvement. The self-driving system is initially a snapshot system m_0 with all parameters of the neural network set as random values. After we train m_0 with data sets as described in D_1 or D_2, the ML system will be evolved into snapshot system m_1 or m_2. These two different transitions come from user expectation on customizing the self-driving system for specific customers and markets. Snapshot system m_1 and m_2 intend to serve the countries or areas driving on left and right respectively and their corresponding training data D_1 and D_2 should provide videos and images under different policies.

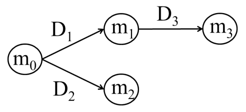
$$\begin{tikzcd} & m_1 \arrow[r, "D_3"] & m_3 \\ m_0 \arrow[ur, "D_1"] \arrow[dr, "D_2"'] & & \\ & m_2 & \end{tikzcd}$$

Fig. 6. An example learning model for the self-driving system

Specifically, data models D_1 and D_2 are described as a set of properties based on the pre-defined meta data model. Table I lists some example properties where $parent.child$ denotes the value of the children element *child* of element *parent*.

TABLE I
THE PROPERTIES FOR DATA MODEL D_1 AND D_2

D_1	$\forall_{es-pair_i \in \tau(D_1, <semantic>)} \cdot$ $es-pair_i.env.policy.side = L$
	$\tau(D_1, <representation>) = \{video\} \wedge$ $\tau(D_1, <video.size>) > 10000h$
	$\tau(D_1, <distribution>) =$ $P(\tau(D_1, <scenario>) = jam) > 0.7$
D_2	$\forall_{es-pair_i \in \tau(D_2, <semantic>)} \cdot$ $es-pair_i.env.policy.side = R$
	$\tau(D_2, <representation>) = \{image\} \wedge$ $\tau(D_2, <image.resolution>) > 600ppi$
	$\tau(D_2, <distribution>) =$ $P(\tau(D_2, <light>) = dark) > 0.9$

Training data in $D1$ satisfies at least 3 properties: all traffic scenarios are under the policy of driving on the left, video is the only format and the size should be more than 10000 hours, more than 70% of environment settings involve traffic jam.

DOI reference number: 10.18293/SEKE2022-0143

Large portion of the videos are required to capture scenarios of traffic jam and the resultant system m_1 is supposed to be applied for metropolitan areas. There are also 3 example properties for D_2: all traffic scenarios are under the policy of driving on the right, image is the only format and the resolution should be more than 600 *ppi*, more than 90% of environment settings are in dark. Most of the training images are in dark environment and system m_2 is expected to support truck drivers.

The more properties we specify, the more effective training data we can collect for achieving expected snapshot systems. As the basis for property description, element structures in meta models need to be enriched through the cooperation of software and domain experts.

Snapshot model m_1 and m_2 can be built by establishing relations between output from different input, as we have mentioned in the middle layer. They are not explicitly described due to the sake of space.

Learning from data model D_3, m_1 will be further evolved into m_3. Training data in D_3 should be collected with signs satisfying the following property.

$$\forall_{es-pair_i \in \tau(D_3,<semantic>)} \cdot \forall_{sign \in es-pair_i.env.traffic.sign} \cdot \\ \exists_{obj \to a \in \tau(m_1,<detect>)} \cdot a > 0.9 \wedge dis(sign, obj) < \varepsilon$$

According to the property, all the driving scenarios in D_3 involve adversarial samples for sign recognition. After conducting the transition from m_1 to m_3, the robustness of the system will be improved in terms of the following accuracy relation between m_1 to m_3.

$$\forall_{obj \to a \in \tau(m_3,<detect>)} \cdot a > [\tau(m_1, < detect >)](obj)$$

It formally states that the accuracy in object detection will be increased after the evolvement from snapshot system m_1 to m_3.

At requirement stage, learning model serves as a study plan for guiding the implementation of learning strategy. Meanwhile, it's detailed description on transitions among snapshot systems facilitates system traceability and maintenance.

IV. Conclusions

This paper intends to take our first step towards systematic requirements modeling method for ML systems. A framework with 3 layers is proposed where lower-level models serve as basis for high-level models. The bottom-level provides meta models for describing environment, system state and data. The middle level consists of snapshot models for describing the behaviors of the ML system at certain learning point and the top level is a learning model that describes learning behavior by transitions among snapshot models. This framework enables us to understand and analyze ML systems at requirements stage and bridges the gap between system modeling and validation.

Our case study demonstrates the framework with a partial model for simplicity. Its performance on large-scale systems needs to be evaluated by applying it in real settings. As the number of elements in meta models increases, the establishment of relevant properties will be much more difficult. How to facilitate model construction for each layer is one of our future works. Furthermore, a supporting tool also needs to be developed in the future to alleviate the burden of model construction, analysis and management.

Acknowledgment

This work is supported by the NSFCs of China (No. 61872144, No. 61902234 and No. 61872146) and National Social Science Foundation (No. 17AZX003).

References

[1] M. Chechik, "Uncertain requirements, assurance and machine learning," *Proceedings of the IEEE International Conference on Requirements Engineering*, vol. 2019-Septe, pp. 2–3, 2019.

[2] Z. Wan, X. Xia, D. Lo, and G. C. Murphy, "How does machine learning change software development practices?" *IEEE Transactions on Software Engineering*, vol. 47, no. 9, pp. 1857–1871, 2021.

[3] K. Ahmad, M. Bano, M. Abdelrazek, C. Arora, and J. Grundy, "What's up with Requirements Engineering for Artificial Intelligence Systems?" *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 1–12, 2021.

[4] G. Dong, J. Wang, J. Sun, Y. Zhang, X. Wang, T. Dai, J. S. Dong, and X. Wang, "Towards Interpreting Recurrent Neural Networks through Probabilistic Abstraction," *Proceedings - 2020 35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020*, pp. 499–510, 2020.

[5] K. M. Habibullah and J. Horkoff, "Non-functional Requirements for Machine Learning: Understanding Current Use and Challenges in Industry," *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 13–23, 2021.

[6] Z. Q. Zhou, L. Sun, T. Y. Chen, and D. Towey, "Metamorphic relations for enhancing system understanding and use," *IEEE Transactions on Software Engineering*, vol. 46, no. 10, pp. 1120–1154, 2020.

[7] X. Zhang, X. Du, X. Xie, L. Ma, Y. Liu, and M. Sun, "Decision-Guided Weighted Automata Extraction from Recurrent Neural Networks," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 13, pp. 11 699–11 707, 2021. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/17391

[8] B. J. Hou and Z. H. Zhou, "Learning with Interpretable Structure from Gated RNN," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 7, pp. 2267–2279, 2020.

[9] Q. Zhang, R. Cao, F. Shi, Y. N. Wu, and S. C. Zhu, "Interpreting CNN knowledge via an explanatory graph," *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, pp. 4454–4463, 2018.

[10] F. Ishikawa and Y. Matsuno, "Evidence-driven Requirements Engineering for Uncertainty of Machine Learning-based Systems," *Proceedings of the IEEE International Conference on Requirements Engineering*, vol. 2020-Augus, pp. 346–351, 2020.

[11] Y. Kawamoto, *Towards Logical Specification of Statistical Machine Learning*. Springer International Publishing, 2019, vol. 11724 LNCS. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-30446-1_16

[12] P. Zhang, J. Wang, J. Sun, X. Wang, G. Dong, X. Wang, T. Dai, and J. S. Dong, "Automatic Fairness Testing of Neural Classifiers through Adversarial Sampling," *IEEE Transactions on Software Engineering*, vol. 5589, no. c, pp. 1–20, 2021.

[13] K. Nakamichi, K. Ohashi, I. Namba, R. Yamamoto, M. Aoyama, L. Joeckel, J. Siebert, and J. Heidrich, "Requirements-driven method to determine quality characteristics and measurements for machine learning software and its evaluation," *Proceedings of the IEEE International Conference on Requirements Engineering*, vol. 2020-Augus, pp. 260–270, 2020.

[14] R. Zhang, A. Albrecht, J. Kausch, H. J. Putzer, T. Geipel, and P. Halady, "DDE process: A requirements engineering approach for machine learning in automated driving," *Proceedings of the IEEE International Conference on Requirements Engineering*, pp. 269–279, 2021.

[15] P. Höfner, R. Khedri, and B. Möller, "An algebra of product families," *Software and Systems Modeling*, vol. 10, no. 2, pp. 161–182, 2011.

[16] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "Nuscenes: A multimodal dataset for autonomous driving," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, no. March, pp. 11 618–11 628, 2020.