Abstract. The field of machine learning focuses on computationally efficient, yet approximate algorithms. On the contrary, the field of formal methods focuses on mathematical rigor and provable correctness. Despite their superficial differences, both fields offer mutual benefit. Formal methods offer methods to verify and explain machine learning systems, aiding their adoption in safety critical domains. Machine learning offers approximate, computationally efficient approaches that let formal methods scale to larger problems. This paper gives an introduction to the track “Formal Methods Meets Machine Learning” (F3ML) and shortly presents its scientific contributions, structured into two thematic subthemes: One, concerning formal methods based approaches for the explanation and verification of machine learning systems, and one concerning the employment of machine learning approaches to scale formal methods.

1 Preface

During recent decades, machine learning has risen to an immense level of prominence in the realm of computer science. From language processing [6] and computer vision [12, 40] to playing complex games [39] and self driving cars [36], machine learning has found success in domains where it is almost impossible to succeed with traditional, handwritten programs. At its core, the promise of machine learning is an attractive one: Given enough data and enough computational resources, complex problems can be trivially solved by an autonomously learning machine without requiring human work.

On the flipside of this utopian vision of machine learning stands the reality of its opaqueness and its common lack of reliability. The solutions that machine learning offers, while powerful, are sometimes heavily flawed, often unreliable and almost always incomprehensible. As machine learning solutions are trained from necessarily incomplete data, their behavior is largely uncontrolled in situations that are unlike those they encountered during training.
Fig. 1. Popular adversarial example of Goodfellow et al. [14]. Adding a small fine-tuned noise to a correctly classified image can change the classification at-will of the attacker.

Fig. 2. Popular example of an adversarial object from Athalye et al. [2]. The 3D printed turtle is incorrectly classified as a riffle throughout multiple images with different backgrounds and object rotations.

This situation is further complicated by the fact that machine learning solutions do not usually adhere to human intuition. Unlike code written by a human, that usually obeys reason and logic, machine learning systems are entirely unrestrained. Phenomena such as adversarial attacks [42], where machine learning systems can be fooled by an almost imperceptible change in their input, showcase just how big the gap between human reasoning and machine learning systems can be. Popular examples include the panda-gibbon adversarial example by Goodfellow et al. [14] (cf. Fig. 1) and the turtle-rifle adversarial object by Athalye et al. [2] (cf. Fig. 2). This erratic and unintuitive nature of machine learning decision making makes it especially hard to achieve machine learning systems that are trustworthy enough to be deployed in real world scenarios.

Disturbing issues with regards to both social fairness [31] and safety critical applications [26] are direct consequences of these properties and have repeatedly halted the adoption of machine learning systems in practical applications beyond the minimal-risk domains where they can currently be employed.
As machine learning transforms the world, formal methods play a key role in ensuring that learned systems are well-understood, fair and safe.

This landscape, while challenging, represents a fruitful soil for the application of formal methods. From model checking [8, 32] of large reactive systems to verifying large code bases [44], the formal methods community has in the past routinely provided reliable, mathematically sound approaches to analyze and verify large, complex systems. Conceptually, this is precisely what is needed to enable the widespread adoption of machine learning systems in practical domains where safety and security become imperative.

However, there is still a noticeable gap between machine learning and formal methods. The challenging nature of machine learning systems requires approaches that are specifically tailored to the system at hand to achieve scalability, comprehensibility and reliability. Symmetrically, the field of formal methods also stands to benefit from novel approaches in machine learning.

Formal methods, at present, offer mathematically precise and rigorous analyses of complex systems. As a consequence, formal methods play a significant role in the adoption of computer systems into safety critical industries, from healthcare to aerospace applications and finance [45]. However, when applying formal methods to large scale systems in practice, issues with scalability arise frequently. Even relatively fundamental techniques of formal methods such as SMT solving [4] and model checking [32] are challenging to scale to large problems.

In this respect, the profiles of formal methods and machine learning are diametrically opposed: Formal methods are mathematically rigorous and well founded while often incurring scaling issues. Machine learning approaches lack mathematical rigor in favor of approximate, statistical solutions that, in turn, are easy to scale to large problems. This motivates the introduction of machine learning approaches to the field of formal methods. By leveraging probabilistic methods from the field of machine learning, the scalability of formal methods can be improved.

The challenge is to integrate machine learning approaches which are inherently imprecise into the field of formal methods without losing their mathematical rigor.

The track “Formal Methods Meet Machine Learning” (F3ML) aims to bridge the gap between the communities of formal methods and machine learning. By nature of its theme, F3ML can be split into two distinct subthemes, one regarding the application of formal methods in the context of machine learning and one, conversely, regarding the application of machine learning techniques in the realm of formal methods.

The first subtheme contains multiple interesting approaches covering a wide spectrum of machine learning systems that, broadly speaking, make machine learning systems trustworthy using formal methods. This includes:
• Approaches that leverage formal methods to verify, explain and/or analyze machine learning systems in a mathematically rigorous manner.
• Approaches that construct machine learning systems in a manner that is by construction more trustworthy, either by enforcing safety standards during construction or constructing systems that are more easily explained or verified.
• Approaches that leverage statistical methods or methods from traditional test suite design to yield test suites that can either be used to give probabilistic bounds on error probabilities of a given system or which ensure that a certain degree of the systems behavior is properly covered.

The approaches presented here concern a wide variety of machine learning systems that are relevant in practical use.

The second subtheme addresses the application of machine learning techniques in formal methods. This includes work that, in any way, incorporates statistical, probabilistic or approximate methods from machine learning to formal methods to achieve scalability under the constraint that the resulting method is mathematically rigorous or at least such that its probabilistic error can be tightly controlled.

In the following, we will briefly introduce each paper belonging to either subtheme and sketch their respective contributions.

2 Formal Methods for Machine Learning

The presentation of the nine contributions of the track can be structured in the following three sections that roughly focus on explanation, verification, and testing, respectively:

2.1 Explanations of Machine Learning Systems

Explaining a machine learning system entails giving conceptual reasoning as to how the system works and how it makes its decisions in a way that is comprehensible to humans [10]. The following three contributions represent different flavors of explainability: The first paper presents an “explainable-by-construction” approach, aiming to construct a machine learning system that is by virtue of its structure comprehensible to humans. The second paper presents an approach that changes the representation of a machine learning system, transforming a previously black-box system into a comprehensible, white-box system that serves as an explanation. The third contribution combines formal methods and machine learning to increase the dependability of AI and increase their robustness. Central to their approach is the modeling of uncertainty—that is inherent to almost all machine learning systems—using formal methods.

Learning Explainable Controllers. In their paper “Algebraically Explainable Controllers: Decision Trees and Support Vector Machines Join Forces” [22]
Jüngermann, Krčtínský, and Weininger use two machine learning models—Support Vector Machines (SVMs) [35] and Decision Trees [33]—to encode a previously constructed controller in a new model that is easier to comprehend, analyse, and verify, and that also has a smaller memory footprint for usage in e.g. embedded systems. Their approach highlights the benefits of Decision Trees:

- They are human comprehensible and are therefore widely regarded as explainable structures in XAI research [17, 46].
- They encode shared predicates efficiently reducing memory footprint.

Generally, a drawback of using Decision Trees as a machine learning (ML) model is that they are either greatly limited in their expressiveness in comparison to other ML models or that their training is very inefficient (several aspects of optimal decision tree construction are NP complete [33, Chapter 6.1]). The inefficiency is a result of the learning algorithm of Decision Trees: At each node all possible splits are ranked using one of many established measures. Thus, the training time is mostly influenced by the class of predicates used. While the space of simpler (axis-aligned) predicates is finite and can easily be explored completely, more expressive predicates (like linear, polynomial, and algebraic predicates) cannot be ranked efficiently.

The authors instead propose the usage of SVMs to find proper predicates. An SVM works by finding the best separating hyperplane between two classes of points and is therefore a natural candidate for finding linear predicates. Additionally, the so-called kernel trick is almost always applied to SVMs. A kernel is a non-linear projection of the original data into a higher dimensional space that can be efficiently computed. As a result, linear predicates found by the SVM in the higher dimensional space correspond to algebraic predicates when observed in the original space. Combining this trick with domain knowledge, the authors derive a problem specific higher-dimensional space that yields polynomial predicates of second degree (quadratic equations) as found frequently in physical applications.

Further, the authors present a series of post-processing steps that can be applied to predicates to improve their comprehensibility making the model an even better explanation. With their running example “cruise control” the authors show how their approach can capture the behavior of a controller using a Decision Tree with just 13 predicates (the same number of predicates another team reached with a handcrafted solution).

Explaining Neural Networks via Decision Structures. It is a well-known fact that, despite their widely renowned success in practice, neural networks are regarded as opaque models (so-called black-box models), whose behavior still evades human intuition [11]. The paper “Towards Rigorous Understanding of Neural Networks via Semantics-preserving Transformation” [38] by Schlüter, Nolte, and Steffen presents a conceptual approach that opens the metaphorical black-box by transforming a (piece-wise linear) neural network into a fully semantically equivalent white-box model in the form of a **Typed Affine Decision Structure** (TADS). TADS are also structurally similar to Decision Trees, utilizing
again the explainable nature of Decision Trees [17, 46] as in the previous paper. Therefore TADS constitute a suitable model for correct, sound, and complete explanations of neural network behavior.

Conceptually, TADS are obtained from a given piece-wise linear neural network using symbolic execution [5, 25] yielding complete explanations for neural network behavior. The authors transform the resulting structure into a decision tree specialized to the profile of neural networks: For example, the predicates derived from neural networks are usually unique, rendering aggregation and variable ordering unnecessary (techniques used extensively in Decision Diagrams and Random Forests). On the other hand, new techniques are applied like infeasible path elimination [15, 34] that greatly reduce the size of TADS while preserving semantics. Moreover, much like in the case of Algebraic Decision Diagrams [3], the authors show that TADS also inherit the algebraic properties of the underlying algebra of the terminal nodes (which is for TADS a real-valued vector space). This can be used to elegantly decide a wide array of questions regarding neural networks, most notably whether two neural networks represent the same function or whether they represent functions that differ by only a small amount. The paper contains a running example that illustrates how TADS can be used to precisely explain, verify and compare smaller-sized neural networks.

Robust and Dependable Artificial Intelligence. In his talk “Robust and Dependable Artificial Intelligence” Nils Jansen presents his vision of foundational and application-driven research in artificial intelligence (AI). He takes a broad stance on AI that brings together machine learning, control theory, and formal methods, in particular formal verification. As part of his research line, he studies problems inspired by autonomous systems, planning in robotics, and direct industrial applications. A shared key aspect in these problems is a thorough understanding of the uncertainty that may occur when machine learning agents operate in the real world. He details the following goals and the inherent real-world challenges that are central to his efforts:

- Increase the dependability of AI in safety-critical environments.
- Render AI models robust against uncertain knowledge about their environment.
- Enhance the capabilities of verification to handle real-world problems using learning techniques.

As a concrete research highlight, Nils Jansen presents a method that directly integrates techniques from machine learning with formal verification. He uses partially observable Markov decision processes (POMDPs) as formal model for planning under uncertainty, and recurrent neural networks (RNNs) as policy representations for these POMDPs. He trains RNN-based policies and then automatically extracts a so-called finite-state controller (FSC) from the RNN. Such FSCs offer a convenient way to verify temporal logic constraints. His method exploits so-called counterexamples as diagnostic information to either adjust the complexity of the extracted FSC, or to improve the policy by performing focused retraining of the RNN. The method synthesizes policies that satisfy temporal logic specifications for POMDPs with up to millions of states.
2.2 Verification of Machine Learning Systems

The following three contributions are each concerned with different flavors of machine learning verification. Broadly speaking, the considered verification approaches entail any approach that can either prove or disprove properties of machine learning systems in a mathematically rigorous manner, ensuring their safe deployment in safety critical domains.

Formal Verification for Neural Networks. The current renaissance in artificial intelligence (AI) has led to the advent of data-driven machine learning (ML) methods deployed within components for sensing, actuation, and control in safety-critical cyber-physical systems (CPS). While such learning-enabled components (LECs) are enabling autonomy in systems like autonomous vehicles and robots, ensuring that such components operate reliably in all scenarios is extraordinarily challenging, as demonstrated in part through recent accidents in semi-autonomous/autonomous CPS and by adversarial ML attacks.

In his talk “Formal Verification for Neural Networks in Autonomous Cyber-Physical Systems” Taylor Johnson discusses formal methods for assuring specifications—mostly robustness and safety—in autonomous CPS and subcomponents thereof using the software tools NNV and Veritex, developed as part of an ongoing DARPA Assured Autonomy project. These tools have been evaluated in CPS development with multiple industry partners in automotive, aerospace, and robotics domains, and allow for analyzing neural networks and their usage in closed-loop systems. Further, Taylor Johnson discusses his ongoing community activities that are relevant in this context, such as the Verification of Neural Networks Competition (VNN-COMP) held with the International Conference on Computer-Aided Verification (CAV) the past few years, as well as the AI and Neural Network Control Systems (AINNCS) category of the hybrid systems verification competition (ARCH-COMP) also held the past few years. The talk concludes with a discussion of future directions in the broader safe and trustworthy AI domain, such as in new projects verifying neural networks used in medical imaging analysis.

Property Directed Verification of Recurrent Neural Networks. In their paper “Analysis of Recurrent Neural Networks via Property-Directed Verification of Surrogate Models” [24], Leucker et al. present a novel approach to the property directed verification of Recurrent Neural Network (RNN) classifiers such as Gated Recurrent Units (GRUs) and Long Short-Term Memory networks (LSTMs) [7,19]. RNNs operate on input sequences of variable length making them a suitable model for e.g. speech recognition tasks. Assigning a label to any given input allows one to model the behaviour of a binary classifier in terms of a deterministic automaton. As end-to-end verification of RNNs would require unrolling the network multiple times, standard neural network verification techniques would be overly expensive. This motivates the need for a different approach tailored specifically to their profile by building on active automata learning.
The authors use L*, a well understood and rigorous algorithm that can learn an unknown automaton by actively probing it [1], to iteratively learn an automaton that acts as a surrogate model of the RNN at hand. This yields an elegant and effective verification loop that repeatedly refines the surrogate model until it mirrors the neural network closely to reveal potential erroneous behavior. An especially important facet of this approach is the targeted refinement of the surrogate model, actively inspecting it for property violations and checking whether these violations transfer from the surrogate model to the actual system. The paper also presents experimental results underlining the scalability and utility of this approach, showing that it compares favorably to existing approaches at the task of verifying LSTMs.

Connecting Reinforcement Learning and Model Checking. In the setting of reinforcement learning, a machine learning agent is tasked with learning how to operate in some environment, formalized as a Markov Decision Process, to maximize some form of reward [23,41]. This setting is of huge practical interest as it models the learning process of humans that learn through experience how to operate in the real world. As a consequence, there exist many different approaches to solving these problems, some from the realm of machine learning and some being based upon formal methods and mathematical rigor.

The differences, similarities and synergies between these approaches are highlighted in the paper “The Modest State of Learning, Sampling, and Verifying Strategies” [18] by Hartmanns, and Klauck. The paper connects probabilistic model checking and statistical model checking with scheduler sampling, as well as traditional Q-learning with value iteration methods and deep Q-learning. Further, the authors present theoretical and empirical comparisons between these methods. An especially interesting observation is made regarding symmetry between the formal methods approaches and the reinforcement learning approaches. Q-learning and probabilistic model checking work relatively similar, both converging to the optimal strategy via a fixed-point based iteration scheme. Both methods incur huge memory costs as the problem becomes more complex. Deep Q-learning and statistical model checking with scheduler sampling are both used to remedy this issue, providing easier scalability at the cost of precision. The authors also discuss corresponding tool support for statistical model checking [29] and how it can be applied to machine learning in the form of deep statistical model checking [16].

2.3 Test-Based Validation of Machine Learning Systems

In many cases, verification of neural networks is computationally too expensive to be feasible. In such cases, it is natural to test the neural network at hand. While not as rigorous as formal verification, software testing is a cornerstone of ensuring trustworthiness of software [13], but its extension to machine learning systems is challenging. The following paper is concerned with the transfer of traditional testing strategies to neural networks.
Neural Network Testing. As neural networks do not generally adhere to human intuition, it is much more difficult to evaluate whether a given test suite adequately covers enough of the neural networks behavior to give confidence in its reliability. In their paper “An Overview of Structural Coverage Metrics for Testing Neural Networks” [43] Pasareanu et al. discuss various coverage metrics for testing neural networks.

The authors present multiple different coverage metrics from existing literature, some adapted from traditional testing, some specifically designed for neural networks. Especially for the coverage metrics that are adapted from traditional testing, it is not obvious whether they adapt well to neural networks. Therefore, the authors conduct an experimental study where they examine the link between coverage metrics and the quality of the corresponding test suite. In particular, they show that some existing coverage metrics are inadequate as an indicator for a good coverage of the neural networks behavior, revealing a potential gap in existing research.

The authors also present DNNCov, a tool that automatically generates coverage reports for a given test suite and a given neural network, indicating the coverage achieved by said test suite on said network. The same tool can also be used for other purposes such as the reduction of test suites and the pruning of unused neurons.

3 Machine Learning for Formal Methods

In the previous section, we already discussed the paper “The Modest State of Learning, Sampling, and Verifying Strategies” [18] which discusses, among other topics, the applicability of statistical model checking to the task of verifying neural network based reinforcement learning systems. Thus, it interprets statistical model checking as a formal method that is applied to machine learning systems.

However, it is important to note that statistical model checking plays a twofold role in this context. While statistical model checking has its roots in formal methods, it relies on probabilistic methods and is thus Technologically closely aligned with the field of machine learning. In that way, statistical machine learning can be considered as an approach that leverages tools from machine learning to achieve scalability of formal methods.

The following papers present improvements in the realm of statistical model checking, incorporating tools from probability theory and statistics to improve reliability and accuracy of statistical model checking.

Importance Splitting in Statistical Model Checking. Statistical model checking consists of learning the probability that the execution of a system will satisfy a given property [29,30]. The approach elegantly combines (1) a simulation-based algorithm for learning the probability distribution of satisfying the property by observing a fixed number of its executions with (2) runtime verification algorithms applied on these executions.

The efficiency of the SMC depends on the number of executions needed to obtain an estimate while minimizing the error rate. The most common SMC
learning algorithm is that of Monte Carlo. When it comes to validating a property that has a high chance of being satisfied, Monte Carlo is considered efficient. In this case, the algorithm minimizes the number of simulations and guarantees a low error rate. The situation changes when one must estimate probability distributions of rarer events such as the probability that an execution contains a bug. This situation comes from the uniform character of the Monte Carlo simulation which does not aim to find the bug. It therefore takes too many simulations to influence the variance of the distribution. To overcome this problem, several authors have proposed learning algorithms that guide the simulations. These techniques, called “importance splitting” [21], orient the execution simulation according to the intermediate results of the runtime verification algorithm. This helps to isolate simulations and identify the bug. These techniques have been deployed in many contexts ranging from automotive to computational biology. Most of the existing work is limited to prototypes and pure probabilistic systems. Except for very restricted situations [20], real-time is not considered. This limits the applicability of the approaches to concrete problems. The paper “Importance Splitting in UPPAAL” [27] proposes a new importance splitting approach for systems that combine both probabilistic and timed aspects. This work extends existing work on the topic by (1) adding real-time aspects into the sampling process, and (2) providing a professional implementation within the UPPAAL toolset. The efficiency of the approach is illustrated on two concrete problems, one of them being an estimate of the spread of the contagion of the COVID-19 epidemic.

Statistical Model Checking for Variability-Intensive Stochastic Systems. Software product lines are sets of computer systems that share many common behaviors and that differ in identified functionalities. For a set of \( n \) functionalities one can generally create \( 2^n \) different systems. Checking each system individually would introduce an explosion of time. To overcome these problems, researchers have proposed compact product line representations [9]. These representations make it possible to check all the products in one pass. For nearly 10 years, these approaches were limited to purely Boolean systems. Recently, this approach has been extended to stochastic systems. In this case, one must calculate the probability that a product satisfies the property. This calculation is generally done by extending classical exhaustive algorithms such as those implemented in PRISM (see [37] for an example). The contribution of the article “Verification of Variability-Intensive Stochastic Systems with Statistical Model Checking” [28] is to extend SMC to learn the probability distribution of each product by simulating the structure which gathers the behaviors of the set of products. This approach has a double advantage:

1. The SMC simulation is more efficient than the exhaustive model checking algorithm.
2. The simulations are done at once on all the products by exploiting the efficiency.
The authors implemented their approach in a prototype and proved its effectiveness on concrete case studies.

Acknowledgements. As organisers of the track, we would like to thank all authors for their contributions. We would also like to thank all reviewers for their insights and helpful comments and all participants of the track for asking interesting questions, giving constructive comments and partaking in lively discussions.

References

5. Boyer, R.S., Elspas, B., Levitt, K.N.: Select-a formal system for testing and debugging programs by symbolic execution. ACM SigPlan Not. 10(6), 234–245 (1975)


