Designing Robust Autonomous Systems

Designing Robust Autonomous Systems

Proefschrift ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college voor promoties
in het openbaar te verdedigen op

vrijdag 22 april 2022
om 10.30 uur precies

door

Alexandru Constantin Serban

geboren op 10 mei 1992
Macin, Roemenië

Promotor(en):

Dr. ir. E. Poll
Prof. dr. ir. J.M.W. Visser (Universiteit Leiden)

Manuscriptcommissie:

Prof. dr. L. Batina
Prof. dr. I. Crnkovic (Chalmers Tekniska Högskola, Zweden)
Prof. dr. A. Van Deursen (Technische Universiteit Delft)
Prof. dr. K. Kersting (Technische Universität Darmstadt, Duitsland)
Prof. dr. I. Valera (Universität des Saarlandes, Duitsland)

The work in the thesis has been carried out under the support of NWO through the i-CAVE program.

Good tests kill flawed theories.
We remain alive to guess again.

Karl Popper

Contents

Summary

Following the recent surge in adoption of machine learning, the negative impact that well-intended but ill-considered development of these technologies can have on organisations, users or society is now widely recognised. To address these issues, researchers, policy makers and other stakeholders have advanced methods and guidelines for the development of ethical, lawful and robust machine learning.

Robustness is the ability of a system to cope with errors and erroneous inputs during execution and is considered one of the most important challenges to large scale deployment of machine learning technologies for which technical solutions can be developed. In this thesis we tackle several challenges related to the design, development and deployment of *robust autonomous systems*. We regard autonomous systems as *software* systems capable to *perceive* the environment they operate it, reason about it and *plan* future actions – where both perception and planning are implemented using machine learning algorithms. While we focus on a particular class of machine learning algorithms, called deep learning, many of the practices and methods proposed generalise to the broader field of machine learning.

Since trying to achieve robustness has broad implications along each stage of the software development life cycle for autonomous systems, we propose a *holistic* approach to achieve robustness that incorporates (i) a system wide, macro perspective and (ii) an algorithmic, micro perspective.

In the first part of the thesis we discuss the (i) macro perspective through the lens of *software engineering for autonomous systems*. We tackle robustness challenges at all stages of the development life cycle, although we focus on *software architecture*, which is one of the first stages of the life cycle. We introduce and validate a catalogue of architectural tactics that can be used to satisfy quality attributes of systems with machine learning components and a catalogue of engineering best practices for their development. Moreover, we introduce a software architecture evaluation method for systems with deep learning components that focuses on their inherent uncertainty.

In the second part of the thesis we discuss the (ii) micro, algorithmic perspective. We focus on developing robust computer vision algorithms against intentional perturbations called *adversarial examples* and on developing *formally verifiable deep learning based planning algorithms*. We introduce two methods that decrease or completely remove the need to add adversarial examples to the training process, which can decrease training time for robust computer vision algorithms by a factor of seven. Moreover, we present a method to reduce the complexity of deep learning based planning algorithms, which makes formal verification of these algorithms possible.

Acknowledgements

I recall the train journey back from the first interview with Erik in Amsterdam. One of the questions he asked me was why would I like to do research and get the position. My answer was because I thought it was fun. On the way back, reflecting on the interview, I found my answer naive and thought would never get the position. Looking back now, no matter how many twists and turns the PhD journey had, it never stopped being fun.

First and foremost I would like to say thank you, Erik. You've been more than a mentor to me. I learned so many things from you, professionally and personally, that I would be a completely different person without them. Thank you for your patience, for your extensive support and for constantly pushing me out of my bubble. Your wittiness and humor made every trip to Nijmegen fun, no matter what NS had planned that day.

To Joost, thank you for your pragmatism and vision. You taught me that ideas need to be focused, embedded in context and practical. Thank you for also nurturing the philosophy part of the PhD. I always enjoyed a brief digression on Wittgenstein or on the profound differences between Popper and Kuhn.

Magiel, thank you for your support and for always being a rational voice. Our discussions helped me keep my feet on the ground and put things in perspective. Also thank you for hosting me at SIG, an experience from which I learnt the most.

Writing this thesis has been a collective effort and I am grateful to all the collaborators who made it possible and helped brainstorm ideas. Thank you Holger, Koen, Loek, Nils, Sangeeth and Yanja for being part of the team and the manuscript committee for reviewing the manuscript and for providing feedback.

To the amazing people that coloured this journey, words can not describe my gratitude for the unforgettable moments we spent together. A warm thank you to the SIG team, from which I learnt so much about software engineering over coffee or lunch. To Dan, Kiril and Laura for keeping the Rotterdam group together and for organising countless trips and parties. To Aykan, Dimitris, Gaby and Moustafa for making the weekends fly. To Bogdan for pub crawling and Sunday cappuccinos. To Enrique for being so kind. To Cris and Mircea for endless discussions about the universe. To Xef for the pandemic adventures. To Alin and Georgian for unconditional friendship and invaluable support. And especially to my family for always supporting my endeavours and making the world feel small.

Constanta, February 2022
Alex

1

Introduction

Following decades of large scale data collection from software systems, the world is increasingly relying on machine learning models to extract insights from data and predict future events. Together with the proliferation of software systems across all levels of our society, which enables the collection and integration of larger amounts of data, the influence of machine learning models on human lives and our society is expected to increase substantially. Machine learning is an application of artificial intelligence which builds technical models of data to help computers learn new tasks without instructions.

Daily, we already rely on autonomous decisions made by machine learning models to curate our e-mails and news, to guide our shopping or to choose which movies to watch. And, unfortunately, we can already observe (unintentional) harm caused by improper development of these technologies, such as the perpetuation of social bias in digital products or the creation of filter bubbles.

Yet, we can expect larger unintentional harm once decisions made by machine learning models become mission- and safety-critical, as is the case for self-driving cars or automatic diagnostic systems. To prevent autonomous decisions from causing harm to human lives, society, or the environment, machine learning models (and the systems they are part of) have to be *robust* against a wide set of technical and societal factors.

The challenges associated with the development of robust systems that incorporate machine learning models are well acknowledged in research and by policy makers or advisers. For example, the European Commission published a set of guidelines for the development of trustworthy artificial intelligence systems in Europe [106].

The guidelines make technical and societal robustness one of the three pillars for trustworthiness, along with ethical and lawful development. From these pillars, robustness is considered the main challenge for which technical solutions can be developed. Nonetheless, while the challenges associated with robustness are well acknowledged, the solutions fall short. For example, after more than a decade of research in adversarial examples – a known threat to robustness of machine learning models – all defences against adversarial examples have been breached and no technical solution exists to decisively overcome this threat.

In this thesis, we propose to approach the challenges stemming from the design, development and deployment of robust autonomous systems holistically, by tackling them both

1

at a (macro) *system* level and by zooming in on specific (micro) *algorithmic* challenges. We regard autonomous systems as *software* systems capable to *perceive* the environment they operate in, reason about it and *plan* future actions - where both perception and planning are based on machine learning models (as for instance in autonomous vehicles). While we focus on a class of machine learning algorithms, called deep learning, many of the practices and methods that we propose generalise to the broader field of machine learning.

We believe a holistic approach is needed to achieve robustness and tackle its broad implications along each stage of the development life cycle: from system, data and algorithm design, to operation and governance.

1.1 Small and large worlds

When describing statistical inference, Savage [231] used a metaphorical distinction between small and large worlds. Making a decision in the large world, such as predicting the outcomes of an action, entails that every relevant information describing the state of the world is known. This is rarely the case in real life, since the complexity of the world hardly ever reveals itself in complete detail. Instead, when faced with decisions we are bound to only use a fraction of the potentially relevant information, equivalent to a smaller world.

The large world is a complete and highly detailed description of the information relevant to a decision, while the small world is a less detailed or incomplete version of the large world. For example, when Christopher Columbus set sail west towards the East Indies, he only had access to a part of the information relevant for his trip: he knew that Earth is spherical. He believed that the planet is smaller than it is in reality, which led to his decision to set sail west, instead of east. Columbus made a decision based on a small world, which turned out to be wrong in the large world, but favorable for his voyage [167].

Machine learning models can be defined using the same approach: a model is built based on a small world with the aim of deploying it in the large world. The small world is always an incomplete representation of the large world, which often includes favorable assumptions. Since in the large world there may be events not modeled in the small world, we expect these models to make mistakes once deployed.

Together with the availability of large data sets we are witnessing a prevalence of machine learning models deployed in the large world. These models allow inductive inference in applications where deductive inference falls short. For example, the models are used to identify objects in streams of videos, classify them, or translate speech to text. More broadly, the models seek solutions to problems for which writing specifications or finding analytical solutions is currently beyond reach.

The increasing reliance on machine learning has been powered by innovations and popularisation of machine learning models that allow a better representation of the small world and avoid favourable assumptions. In particular, attention has focused on deep learning models, which are very good at representing small worlds from raw data with minimal assumptions about the structure of the worlds. Nonetheless, the popularity of deep learning does not entail that other machine learning models are not relevant. It rather acted as a catalyst and attracted general attention for all classes of machine learning models. While in this thesis we focus more on deep learning, many practices and methods proposed generalise to the broader field of machine learning.

1.2 Research challenges

The design, development and deployment of robust autonomous systems with deep learning components raises several technical and organisational challenges. In particular, regarding (i) how deep learning components can be integrated in a robust way in larger systems and (ii) how robust deep learning models can be developed.

To *integrate* deep learning components in larger systems, engineering principles that can ensure functional and non-functional requirements, such as robustness, reliability or availability must be applied or new principles tailored to deep learning must be developed. A complication here is that deep learning models are considered opaque and difficult to interpret. For example, the expressiveness of deep learning models to represent small worlds is given by their large number of parameters. For instance, a basic image classification algorithm has an order of 10^9 parameters [102]. This large number of parameters makes reasoning about the model's decisions and functional (or non-functional) requirements difficult (in most cases practically impossible). It also hinders the adoption of traditional methods for proving the correctness of algorithms, such as formal verification.

Moreover, because the performance of deep learning components is strongly connected with the data sets describing the world, the development of software with deep learning components requires organisations to adopt a *different development process* from traditional software. For example, the evolving nature of the large world and the data driven behaviour of deep learning components requires teams to adopt *faster, experimental, data collection and development iterations.* Also, the incapacity to represent the large world based on the data collected (the small world) requires teams to adopt a thorough strategy for monitoring deployed models and for managing incidents. Likewise, the inherent uncertainty of deep learning models together with their probabilistic behaviour requires teams to adopt *new strategies for software design* and new tactics to satisfy non functional requirements such as robustness, reliability, or redundancy. Therefore, the first challenge we identify is *How to develop robust systems with deep learning components?*

A challenge that follows naturally is *How to develop robust deep learning models?*, i.e., how to develop more robust *components of a system.* It is generally accepted that more data describing the world allows training better deep learning models. However, to model complex tasks reliable (such as image classification), larger data collections than those available are needed. In fact, theoretical results suggest the size of the data sets needed to reliable model complex tasks is currently beyond reach [256]. Therefore, deep learning models rely on the favorable assumption that the data from the large world are in close resemblance to the data from the small world.

Nevertheless, when faced with a complex and evolving large world, in which the distribution of test data shifts naturally over time, *deep learning models exhibit low robustness.* This lack of robustness manifests when the test data are slightly perturbed and severely threatens adoption of deep learning in contexts where decisions may have a negative impact on human lives, or on the environment [248]. For example, adding small intentional perturbations to input data – called adversarial examples – can easily induce undesired behaviours. This lack of robustness can be exploited by malicious actors in security sensitive contexts [242].

1

1.3 Research plan

The challenges mentioned above motivate a holistic approach to achieving robustness for autonomous systems. As previously mentioned, we regard autonomous systems as software systems capable to perceive the environment they operate in, reason about it and plan future actions. Both perception and planning are tasks where deep learning models excel over other machine learning models. Furthermore, recent initiatives show that deep learning models can be used to combine perception and planning in one model and outperform methods where the two tasks are solved separately [148]. This type of design decisions, where some functionality can be implemented as one or many components, further motivates the holistic approach to robustness because it involves both the system aspect, where the components and their integration are discussed, and the algorithmic aspect, where specific algorithms are chosen for development.

Autonomous systems also consist of multiple traditional software components, with which deep learning components have to be integrated. In most cases, these systems have stringent safety requirements. For example, in the context of i-CAVE[1] – the research project in which this PhD thesis was carried out – the goal was to evolve vehicles into autonomous systems by deploying computer vision and planning algorithms. We started our journey with explicit investigations into autonomous vehicles, but soon realised that the challenges faced in this use case generalise to the broader field of autonomous systems.

In light of the challenges mentioned above, we ask the question *How do we design, develop and deploy robust autonomous systems with deep learning components?* As mentioned above, we approach this questions at two levels: a system level that tackles robustness from a software engineering angle, and an algorithmic level that tackles robustness from an algorithmic angle. In the first part of the thesis we look at *software engineering* challenges for building *robust autonomous systems* and ways to overcome them. Our work spans all stages of the software development life cycle for autonomous systems, although we maintain a focus on the stage where the software architecture of a system is defined. In the second part of the thesis we look at methods to design robust deep learning models for computer vision and planning. In particular, we discuss robustness challenges for computer vision algorithms against intentional perturbations, also called *adversarial examples* and the challenge caused by the inability to *formally verify deep learning based planning algorithms*.

1.3.1 Part I – Designing robust systems

We begin by studying the challenges that arise when the software architecture of a system is being defined. Software architecture is a step in the software development life-cycle that consists of designing, documenting, evaluating and evolving software architectures [166]. At design time, decisions about the structure of the system, the communication between components and the functional and non-functional requirements are made [237]. Moreover, the software architecture includes decisions about running, maintaining and updating a system in a production environment [299]. Since software architecture is one of the first stages in the software development life-cycle, where trade-offs between quality attributes of a system are decided, it is a natural step to begin with.

[1] https://i-cave.nl

Recurring solutions to issues in software architecture are abstracted to general and reusable solutions called architectural patterns or styles. Lower level decisions with respect to quality attributes of a system are also called architectural tactics [17]. Tactics focus strictly on quality attributes and capture knowledge about the relationship between architectural decisions and their outcomes in terms of improving a quality attribute of a system. In this sense, tactics are focused on smaller issues than patterns and are more dependent to context. For example, a tactic would address issues such as performance by deciding the number of units of concurrency in a component, while a pattern would define the higher level decisions about components and connectors and decide on how different instances of the same component can run in parallel.

In Chapter 2, we address the question *How can software systems be (re-)architected to enable robust integration of deep learning components?* This is one of the first steps for integrating deep learning components in software systems and for deploying them in the large world. To answer the question, we conducted a mixed-methods empirical study consisting of (i) a systematic literature review to identify the challenges and their solutions in software architecture for deep learning, (ii) semi-structured interviews with practitioners to qualitatively complement the initial findings, and (iii) a global survey to quantitatively validate the set of challenges and their solutions. This resulted in twenty challenges and solutions for (re-)architecting systems with deep learning components. The results indicate, for example, that traditional software architecture challenges (e.g., component coupling) also play an important role when using deep learning components, but there are also important challenges specific only to deep learning (e.g., the need for continuously retraining). Moreover, the results indicate that architectural decision drivers which should be emphasised by deep learning, such as privacy, play a marginal role compared to traditional decision drivers, such as scalability or interoperability. Using the survey, we were able to establish a stronger link between the solutions and software quality attributes, which enabled us to provide twenty architectural tactics used to satisfy individual quality requirements of systems with deep learning components. Altogether, the results from this chapter can be interpreted as a first empirical framework that supports the process of (re-)architecting software systems with deep learning components.

In Chapter 3, we argue that deep learning components add a new dimension to traditional software architecture design. This dimension captures the inability to verify that deep learning components satisfy their intended functionality and are able to cope with stochastic events coming from the operational environment [244]. Thus, it translates to uncertainty regarding the functionality of deep learning components in the large world, which contrasts with traditional, deterministic, software.

Although researchers in software architecture have developed methods to tackle uncertainty at design time [71, 170] or at run-time [70], these methods focused primarily on tackling uncertainty related to the parameters used to model a software system, its context or to the instrumentation [70, 71, 170] (e.g., the uncertainty regarding hardware performance [72]). However, deep learning components add a new source of uncertainty, which was only briefly explored previously – uncertainty due to “automated learning" [81, 163, 290].

Motivated by this gap, we address the question *How to compare software architectures with deep learning components?* To answer this question, we introduce a method to evaluate

architecture alternatives for software using both traditional and deep learning components. The method supports reasoning over how architectural patterns can mitigate uncertainty and enables comparison of different architectures combining deep learning and traditional software components. While domain-agnostic and suitable for any system where uncertainty plays a central role, we validate our approach using as example a perception system for autonomous driving. We show that design patterns used in safety-critical systems [13] can be used to decrease the uncertainty of a system with deep learning components and lead to more robust autonomous systems.

In Chapter 4, we address the question *How do teams design, develop and deploy software with deep learning components?* To answer this question, we ran a multi-vocal literature review and compiled from the selected literature a catalogue of software engineering best practices for deep learning applications. The practices and their perceived effects were validated through a large survey with practitioners around the world. The results of the survey show that the practices can be used in any context, independent of the data type or of the algorithms used. Moreover, the results show that adoption of best practices increases with team size and with team experience. The list of practices and the analysis of responses provides a quantitative basis for quality assessment and improvement for teams developing software with deep learning components.

1.3.2 Part II – Designing robust components

In the second part of the thesis, we zoom in on algorithmic robustness for perception and planning. We focus on computer vision and planning because they play an important role in autonomy. In particular, we are concerned with robustness of computer vision algorithms against small, intentional, perturbations also called adversarial examples. Achieving robustness against these perturbations requires to complement the training data set with perturbed samples (adversarial training); a procedure which negatively impacts the training time and the number of samples. In this context, we address the question *How can we reduce the impact of adversarial training on robust computer vision algorithms?*

We begin in Chapter 5 with a brief introduction to adversarial examples and discuss their implications to robustness and security of deep learning models.

Following up, in Chapter 6, we introduce a method to partition the output space of classifiers into class prototypes with large separation and train deep learning models to preserve the separation. The optimisation procedure to obtain the prototypes increases the distance between their centres based on a metric defined in the attack model. Therefore, we call the prototypes *repulsive* prototypes. We show empirically that models trained with repulsive prototypes are almost as robust as adversarially trained models, without the need to generate perturbed data for training. Moreover, models trained with repulsive prototypes are more resilient to large perturbations than adversarially trained algorithms.

Next, in Chapter 7, we introduce a method to train robust classifiers with small training data sets and transfer the knowledge learned about robustness between different models. Towards this goal, we train a meta-optimiser which learns to robustly optimise a model using perturbed samples and use the meta optimiser to transfer the knowledge learned to new models. Thus, the method eliminates the need of adversarial training once the meta-optimiser is trained. We show empirically that the meta-optimiser improves robust-

ness consistently across different architectures and data sets, suggesting it is possible to automatically patch robustness vulnerabilities.

In Chapter 8, we investigate the robustness of deep learning based planning algorithms, where formal verification can not be applied directly because the planning algorithms are too complex. In this context, we answer the question *Can we reduce the complexity of deep learning based planning algorithms and allow formal verification?* In particular, we study the problem of strategy synthesis for partially observable Markov decision processes, where the strategy is synthesised by a deep learning model. To determine if the strategies adhere to (probabilistic) temporal logic constraints is computationally intractable and theoretically hard. In order to overcome this limitation, we introduce a method that combines techniques from deep learning and formal verification. The strategy is learned using a recurrent neural network and restricted to represent a finite memory strategy, which can be implemented on a specific partially observable Markov decision process. For the resulting finite Markov chain, formal verification techniques to be used in order to provide guarantees against temporal logic specifications.

The thesis ends with a discussion and conclusions in Chapter 9.

1.4 Research methodology

In the first part of the thesis we apply various methods from empirical software engineering. In particular, we used four methods from this field: systematic literature reviews, interviews, surveys and case studies [68].

Systematic literature reviews are widely used in empirical software engineering and provide a structured process to identify, evaluate, and interpret the information available regarding a research topic [68, 133, 134]. They consist of an analytical review of the available literature, from which common themes can be distilled. Since software engineering is a practitioner-driven process, we combined literature reviews of academic and non academic literature [82].

Interviews are widely used to collect qualitative data about a topic and to add depth to quantitative observations. We used interviews both in an exploratory manner, to uncover research challenges and solutions adopted by practitioners and in a confirmatory manner, as a way to validate research outcomes [110].

Surveys are used to identify characteristics of software engineering practitioners, such as the solutions they adopt for various challenges. It is generally recommended to compare survey results with other empirical methods, in order to avoid bias stemming from the survey respondents [55, 135]. In this thesis, we combined outcomes of surveys with systematic literature reviews and interviews, an approach that is characterised as mixed-methods research.

Case studies are used in an exploratory manner, as initial investigations of some phenomena in order to derive theories, or in a confirmatory manner, to test hypotheses or existing theories [68]. In this thesis, we used case studies in a confirmatory manner, to test the applicability of the proposed methods.

In the second part of the thesis we applied quantitative evaluation methods specific to machine learning, i.e., we evaluated our algorithms on widely adopted benchmarks for the problems studied [45].

Table 1.1: Research overview, where the chapters in parentheses indirectly support the research questions.

Research Challenge	Research Question	Part	Chapters
How to develop robust systems with deep learning components?	How can software systems be (re-)architected to enable robust integration of deep learning components	I	2, (4)
	How to compare software architectures with deep learning components?	I	3, (2)
	How do teams design, develop and deploy software with deep learning components?	I	4, (2, 8)
How to develop robust deep learning models?	How can we reduce the impact of adversarial training on robust computer vision algorithms?	II	6, 7, (5)
	Can we reduce the complexity of deep learning based planning algorithms and allow formal verification?	II	8

1.5 Research overview

A mapping between the three challenges introduced in Section 1.2, the research questions from Section 1.3 and the structure of the thesis is presented in Table 1.1. We note that the chapters in parentheses address the main research questions by either providing background information (as is the case with Chapter 5) or by providing different perspectives to the research question, but not fully answering it (as is the case with Chapter 4 for the first research question).

1.5.1 Publications supporting the chapters

All chapters have been published, or are under review, in peer reviewed journals and conferences. Therefore, each chapter is self-contained and can be read independently. Only Chapter 5 is a short summary of a previous publication, presenting only the details that are relevant to this thesis. Below, we present a mapping between the thesis chapters and the corresponding publications. Unless otherwise specified, the author of the thesis is also the main contributor to the publications.

- **Part I – Designing robust systems**
  - **Chapter 2** has been published as *Adapting Software Architectures to Machine Learning Challenges*, by Alex Serban and Joost Visser, at the IEEE International Conference on Software Analysis, Evolution and Reengineering, 2022 [239].
  - **Chapter 3** has been published as *Towards Using Probabilistic Models to Design Software Systems with Inherent Uncertainty*, by Alex Serban, Erik Poll, and Joost Visser, at the European Conference on Software Architecture (ECSA), 2020 [244].

  - **Chapter 4** has been published as *Adoption and Effects of Software Engineering Best Practices in Machine Learning*, by Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser, at the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2020 [245].

- **Part II – Designing robust components**

  - **Chapter 5** is a short summary of a paper published as *Adversarial Examples on Object Recognition: A Comprehensive Survey*, by Alex Serban, Erik Poll, and Joost Visser, at ACM Computing Surveys (CSUR), 2020 [242].
  - **Chapter 6** is under review as *Deep Repulsive Prototypes for Adversarial Robustness*, by Alex Serban, Erik Poll, and Joost Visser, 2020 [240].
  - **Chapter 7** has been published as *Learning to Learn from Mistakes: Robust optimisation for Adversarial Noise* at the International Conference on Artificial Neural Networks (ICANN), by Alex Serban, Erik Poll, and Joost Visser, 2020 [243].
  - **Chapter 8** has been published as *Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks*, by Steven Carr, Nils Jansen, Ralf Wimmer, Alex Serban, Bernd Becker, and Ufuk Topcu at the International Joint Conference on Artificial Intelligence (IJCAI), 2019 [46].

    For Chapter 8, the author of the thesis was responsible with the development of the recurrent neural network used to learn a strategy for planning.

1.5.2 Publications not used in the thesis

As previously mentioned, we started our journey with explicit investigations into autonomous vehicles. Therefore, some publications which were not included in this thesis tackle various topics related to robustness of autonomous vehicles (such as developing software architectures for robust autonomous vehicles [241, 251, 252], or investigating the security of the communication between connected vehicles [250]). In the list below we present these publications together with other publications [138, 247–249] to which the author of this thesis contributed:

- Automotive specific publications:

  - *A Standard Driven Software Architecture for Fully Autonomous Vehicles*, by Alex Serban, Erik Poll, and Joost Visser, in Journal of Automotive Software Engineering, 2020 [241].
  - *A Security Analysis of the ETSI ITS Vehicular Communications*, by Alex Serban, Erik Poll, and Joost Visser, at the Workshop on Safety, Security and Privacy in Automotive Systems from the International Conference on Computer Safety, Reliability, and Security (SafeComp), 2018 [250].
  - *Tactical Safety Reasoning. A Case for Autonomous Vehicles*, by Alex Serban, Erik Poll, and Joost Visser, at IEEE 87th Vehicular Technology Conference (VTC Spring), 2018 [252].

  - *A Standard Driven Software Architecture for Fully Autonomous Vehicles* (best paper award) at the IEEE Workshop of Automotive Software Architecture (WASA) from the International Conference on Software Architecture (ICSA), 2018 [251].

- Other publications:

  - *Practices for Engineering Trustworthy Machine Learning Applications*, by Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser, at IEEE Workshop on AI Engineering (WAIN'21) part of the International Conference on Software Engineering (ICSE), 2021 [247].
  - *Safe Reinforcement Learning Using Probabilistic Shields*, by Nils Jansen, Bettina Könighofer, Sebastian Junges, Alex Serban, and Roderick Bloem, at International Conference on Concurrency Theory (CONCUR), 2020 [138].
  - *Designing Safety Critical Software Systems to Manage Inherent Uncertainty*, by Alex Serban, at IEEE International Conference on Software Architecture (ICSA), 2019 [248].
  - *Adversarial Examples - A Complete Characterisation of the Phenomenon*, by Alex Serban, Erik Poll, and Joost Visser, available as preprint arXiv:1810.01185, 2018 [249].

1.6 Context in machine learning or artificial intelligence

We focus on deep learning models because, for the most part, the algorithms used for autonomous systems (and vehicles, relevant to the i-CAVE project) are based on deep learning. However, the practices and methods in the first part of the thesis generalise to the broader field of machine learning. When performing the studies in Part I of the thesis, we made no distinction between machine and deep learning. The only distinction made was with the broader field of artificial intelligence, which is less clearly delineated and involves many more concerns that do not have strictly technical solutions (such as ethics or law).

1.7 Data management

The data collected in the chapters and the code used to produce the results presented in this thesis are listed below.

- Part I

  - Chapter 2 – An Empirical Study of Software Architecture for Machine Learning - Supplementary Materials, https://doi.org/10.5281/zenodo.4564113 [2].
  - Chapter 3 – Towards Using Probabilistic Models to Design Software Systems with Inherent Uncertainty - Supplementary Materials, https://doi.org/10.5281/zenodo.4700095 [1].

  - Chapter 4 – Towards Using Probabilistic Models to Design Software Systems with Inherent Uncertainty - Supplementary Materials, https://doi.org/10.5281/zenodo.4700095 [246]

- Part II

  - Chapter 6 – Deep Repulsive Prototypes for Adversarial Robustness - Code, https://github.com/NullConvergence/repulsive-proto.
  - Chapter 7 – Learning to Learn Adversarial - Code, https://github.com/NullConvergence/Learning2LearnAdv.

The materials from the first part of the thesis are disseminated through the https://se-ml.github.io website, and through the Awesome Software Engineering for Machine Learning reading list https://github.com/SE-ML/awesome-seml.

Chapter 5 is also supported by a tutorial presented at WIFS 2019, and available at https://github.com/NullConvergence/tutorial_adversarialml.

Part I

Designing robust systems

2

Adapting Software Architectures to Machine Learning Challenges

Specific developmental and operational characteristics of machine learning (ML) components, as well as their inherent uncertainty, demand robust engineering principles are used to ensure their quality. In this chapter, we aim to determine how software systems can be (re-)architected to enable robust integration of ML components. Towards this goal, we conducted a mixed-methods empirical study consisting of (i) a systematic literature review to identify the challenges and their solutions in software architecture for ML, (ii) semi-structured interviews with practitioners to qualitatively complement the initial findings, and (iii) a survey to quantitatively validate the challenges and their solutions. In total, we compiled and validated twenty challenges and solutions for (re-)architecting systems with ML components. Our results indicate, for example, that traditional software architecture challenges (e.g., component coupling) also play an important role when using ML components; along new ML specific challenges (e.g., the need for continuous retraining). Moreover, the results indicate that ML heightened decision drivers, such as privacy, play a marginal role compared to traditional decision drivers, such as scalability or interoperability. Using the survey, we were able to establish a link between architectural solutions and software quality attributes; which enabled us to provide twenty architectural tactics used for satisfying individual quality requirements of systems with ML components. Altogether, the results can be interpreted as an empirical framework that supports the process of (re-)architecting software systems with ML components.

This chapter has been published as *A. Serban, J. Visser, Adapting Software Architectures to Machine Learning Challenges, IEEE International Conference on Software Analysis, Evolution and Reengineering, 2022* [239].

2.1 Introduction

Software architecture (SA) plays an important role in data intensive systems, such as big data and analytics platforms. However, until recently, the focus has been on the architectural decisions related to handling and storing large amounts of data, and on decisions that mitigate performance demands of analytics platforms [26, 237].

The interest to develop software with machine learning (ML) components shifts the focus to decisions regarding the operational requirements of serving, monitoring, retraining and redeploying models [236]. These decisions align with proposals to emphasise the operational aspect of SA [299]. Moreover, the inherent uncertainty of ML components demands a stronger emphasis on the uncertainty aspect of SA; where the focus is on assessing the impact of uncertainty, and on the decisions made for its mitigation [70, 244].

Although a significant body of literature studied the relevance of SA for big data and analytics platforms [16, 237], there is little empirical research on the role of SA in systems with ML components [160, 282]. In this chapter, we aim to determine how software systems can be (re-)architected to enable robust integration of ML components.

Towards this goal, we conducted a mixed-methods empirical study consisting of three stages. First, we performed a systematic literature review (SLR) to identify the challenges faced in (re-)architecting systems with ML components, and the solutions proposed to meet them. We analysed 42 relevant articles, from which we compiled an initial set of 18 challenges and solutions. Second, we performed 10 semi-structured interviews with practitioners from 10 organisations – ranging from start-ups to large companies. The interviews were used to complement the initial set of challenges (and solutions), and to assess the impact of each challenge on SA. In total, 2 new challenges were discovered in the interviews, as well as 46 new solutions. Third, we ran a survey with 47 software architects in order to quantitatively validate and complement the challenges and solutions. The survey also established a link between challenges, solutions, and software quality attributes, allowing the solutions to be restated as architectural tactics.

Overall, our main contributions are as follows. First, we summarised academic and grey literature on the topic of SA for ML in a catalogue of SA challenges and related solutions. This information can guide practitioners to (re-)architect software with ML components, or as a gateway to relevant literature. Second, we validated and complemented the initial findings by engaging with practitioners. We found out that, although the initial challenges had solutions in the literature, the solution were considered incomplete by practitioners. Third, we linked the architectural solutions to software quality attributes from the ISO/IEC 25010 standard [118], which allowed to restate them as architectural tactics. Last, we assessed the impact of each challenge on SA, which allowed us to contrast traditional SA concerns with emergent ML related concerns.

This chapter is organised as follows. First, we introduce background information and related work (Section 2.2). Next, we discuss the study design (Section 2.3), and the results (Section 2.4). We end with a discussion (Section 2.5) and conclusions (Section 2.6).

2.2 Background and related work

Software engineering (SE) for ML is receiving increasing attention [185]. The related

literature covers a broad range of topics; from SE challenges raised by the adoption of ML components [160], to practices [245], guidelines [312], or design patterns [287]. Moreover, we consider the related field of SA for big data and analytics platforms [237]. Therefore, we structure the presentation in three steps: first we introduce SE challenges for ML (with a focus on SA), followed by solutions that meet the challenges, and by a discussion on SA for big data and analytics, in the context of ML.

Arpteg et al. [14] introduced twelve SE challenges for ML, classified in three categories: development, deployment and organisational. From these, the challenges related to monitoring and logging ML components, and to effort estimation for development and maintenance, were also identified in our SLR. Since Arpteg et al. [14] do not introduce solutions, the second and third stages of our study can be used to complement theirs.

Similarly, Ishikawa and Yoshioka [117], as well as Wan et al. [282], studied how ML impacts the traditional software development life-cycle. Both studies are based on surveys, and have the bulk of responses from Asia. Notwithstanding this regional bias, they concluded that testing and evaluating the quality of ML components is particularly difficult. Distinct conclusions are drawn with respect to SA. While Wan et al. [282] acknowledged SA for ML as difficult, Ishikawa and Yoshioka [117] concluded that existing SA methods apply equally to software with ML components, although the tool support is immature. We analysed the SA challenges raised by ML with finer granularity, and found out that while some challenges apply equally to software with or without ML components, ML specific challenges (and solutions) also arise.

To classify the SE challenges for ML, Lwakatare et al. [160] introduced a taxonomy, from which the challenges related to scalability, and to serving requirements, were also identified in our study.

An early publication that outlined SA challenges and solutions for ML was the work of Sculley et al. [236]. The authors used the framework of technical debt to explore risk factors for ML components. Particularly, they argued that ML components are subject to all maintenance issues specific to software components, as well as to new issues specific only to ML. Moreover, they introduced a set of anti-patterns and practices used to avoid technical debt. Compared to Sculley et al. [236], the challenges (and solutions) introduced in this paper are broader, and consider more quality attributes. Nonetheless, there is an overlap between the studies.

Breck et al. [36] and Zhang et al. [311] studied the topic of testing for ML components, and introduced testing and monitoring practices for different stages of the ML development life-cycle. While these practices are relevant to SE for ML, we are interested in the architectural decisions for testing ML components. Therefore, we focus on higher-level decisions, such as using automated tests.

Amershi et al. [8] conducted an internal study at Microsoft, aimed at collecting SE challenges and practices for ML. They reported on a broad range of challenges and practices used at different stages of the ML development life cycle. In particular, modularity and component reuse in software with ML components are challenges closely related to SA, also tackled in this study.

Serban et al. [245] and Zhang et al. [312] introduced two sets of SE practices for ML and deep learning, respectively. While some practices are considered in SA – e.g., the adoption of continuous integration – the broad selection of practices does not allow a focus on SA

(as in our study). Therefore, the challenges (and solutions) introduced here can be used to complement theirs.

Nascimento et al. [185] introduced a SLR on the topic of SE for ML that analysed all articles up to 2019. The authors observed that SA is not yet a popular topic. However, their taxonomy classifies software quality and infrastructure concerns separately from architecture. We argue that such concerns are discussed extensively during SA, and should be considered together when evaluating the popularity.

Washizaki et al. [286] studied SA patterns and anti-patterns for ML, extracted from white and grey literature. Their proposal was followed by a larger study, where the initial set of patterns and anti-patterns was extended [287]. Their work is close to the first stage of our study (SLR), where we identified a set of challenges and solutions in SA for ML. We build upon it by enlarging the number of challenges and solutions, and by extensively validating our findings in the second and third stages of the study. Moreover, although the challenges presented by Washizaki et al. [286] are recurrent, we found out the solutions are not. Therefore, we are cautious in using the taxonomy of design patterns. Instead, we focus on smaller building blocks called tactics; which bridge architecture decisions with quality attributes, and form the basis of design patterns [22, 99].

The challenges raised by big data systems regarding continuous expansion of data volumes and the adoption of new technologies have been well studied, and several reference architectures have been proposed – e.g., [16, 237, 238]. However, the proposals emphasise the data aspect of SA, i.e., how to collect and manage various sources of data and satisfy performance demands of analytics platforms. Therefore, although data visualisation and ML components are present in the reference architectures, these do not record decisions taken for development, integration, serving and maintenance of ML components. In this chapter, we focus on the latter, where the data aspect plays an important role, but it is not the main decision driver.

2.3 Study design

Our study was organised in 3 stages, and consisted of a mixed-methods approach with a sequential exploratory strategy [68]. In the first stage, we ran a SLR to identify the challenges faced when (re-) architecting systems with ML components, and the solutions proposed to meet them. The second stage of the study consisted of semi-structured interviews, meant to complement and partially validate the data extracted in the first stage. In the third stage, we ran a survey to gather quantitative data, augment and generalise the findings from the first two stages. Data triangulation from multiple sources is known to increase the reliability of the results [68].

Systematic Literature Review. SLRs are widely used in empirical SE research, and provide a structured process to identify, evaluate, and interpret the information available regarding a research topic [68, 133, 134]. SLRs consist of three parts, namely defining a research protocol, conducting a review and reporting the results. We followed the guidelines from Kitchenham and Charters [133], and defined a research protocol as follows.

Research questions. We aimed to gather evidence about the challenges faced when (re-) architecting software systems with ML components. Moreover, we looked for solutions that meet the challenges and synthesise practices, tactics or patterns. Towards this goal, we

Table 2.1: Research questions for the SLR.

ID	Research Question	Motivation
RQ1	Which are the challenges reported in (re-)architecting software systems with ML components?	Understand the technical and organisational challenges, but also the requirements posed by adoption of ML components.
RQ2	What solutions, tactics or patterns have been reported to successfully meet these challenges?	Understand and identify solutions, tactics, or patterns for SA with ML components.

formulated a set of research questions, summarised with their motivation in Table 2.1. The questions facilitated the identification of challenges in the area of SA with ML components, and enabled the creation of an initial body of knowledge with solutions.

Search strategy. To get a broad set of studies, we used multiple information sources. First, we used automatic queries to retrieve studies from several digital libraries, namely IEEE Xplore, ACM Digital Library (ACM DL), Scopus, and ScienceDirect. Shahin et al. [255] observed that SpringerLink uses a different query mechanism than the others, and that Scopus indexes most articles from SpringerLink. Therefore, in order to avoid inconsistencies in data retrieval, we relied on Scopus. Second, motivated by the findings of Serban et al. [245] – which noticed that most literature on the topic of SE for ML consists of so called grey literature – we performed manual search in Google and Google Scholar, where the first 5 pages of results were inspected. Last, we complemented the data set through a snowball strategy [40], following references of relevant articles.

To define the search query, we followed the guidelines from [133], and composed a string with synonyms of the words "software architecture", "machine learning", "challenges", and "solutions". After piloting several queries to validate the inclusion of previously known articles, we decided to use two distinct queries. The first query retrieved challenges in SA for ML, and the second query retrieved solutions. The search string for the first query was: "(("software architecture" OR "software engineering" OR "systems engineering") AND ("machine learning" OR "deep learning" OR "artificial intelligence" OR "AI") *AND ("challenge" OR "problem" OR "issue")*)", where the emphasised string was replaced in the second query with: *AND ("solution" OR "practice" OR "guideline" OR "tactic" OR "pattern" OR "architecture pattern" OR "design pattern").* Using the word "software" next to architecture or engineering helped to avoid articles from the general field of engineering (e.g., electrical engineering) or architecture. Moreover, we observed empirically that including both "pattern" and "design pattern" makes the query more effective.

Exclusion and inclusion criteria. Since the initial queries returned over 10 000 results, we limited the answers to the first 500 articles, for each data source and query. This reduced the number of articles to 1 000 per source, which corresponds to recommendations and previous studies [165, 255]. Washizaki et al. [286] and Nascimento et al. [185] showed that the majority of articles on the topic of SE for ML were published after 2016. Therefore, we also restricted our search to articles published after 2016. Next, we automatically filtered for duplicates and for records that contained the words "proceedings" or "workshop" in the

Table 2.2: Document selection for each information source.

Source	Retrieved	Automatic Filtering	Manual Inspection	Used
ACM DL	1000	647	21	7
IEEE Xplore	1000	521	22	9
ScienceDirect	1000	513	2	0
Scopus	1000	732	2	0
Google Scholar	100	100	7	3
Google	100	100	12	10
Snowball	-	-	16	13
Total	4200	2613	82	42

title. Moreover, we excluded all opinionated articles; coming from companies or authors which could be traced back to companies that provide tools or services for SA/SE for ML. Thus, some bias regarding solutions driven by tools was avoided. Since ML for SE receives increasing interest, we carefully curated and removed all articles on this topic. Moreover, we removed tool demonstration articles, and those not written in English. In the final selection, we included all studies or grey literature articles that presented challenges or solutions based either on empirical studies or on experience (e.g., studies with empirical validation or organisation blogs describing their processes).

Study selection. After retrieving the initial set of 4 200 documents, we applied the selection criteria as follows. In the first phase we applied the automatic filters, which reduced the set to 2 613 articles. For these articles, we manually inspected the titles and the keywords, and selected 66 articles to be completely assessed. These were read completely and critically analysed, which reduced their number to 29 relevant articles. From their references, 16 new articles were read, from which 13 were used in the final selection. The distribution of articles and their sources for each stage of the review is presented in Table 2.2. We delegate the complete list of articles, their sources, and a demographic characterisation to the supplementary materials. We observe that although ACM DL and IEEE Xplore retrieved the bulk of articles for complete assessment, the grey literature search and snowballing strategies were more effective for the final selection. Moreover, we observe that the distribution of articles per date, (supplementary materials) resembles the one from [185, 286] – i.e., the number of articles is increasing year by year. By analysing the distribution of articles based on the venue type (supplementary materials), we observe that the majority of academic articles were published in conferences, and not in journals. We conjecture that: (i) SA for ML is a emerging field, and the publications did not reached the maturity needed for journal publication, and (ii) journals have longer review cycles, and publications may be in review.

Data extraction and synthesis. From all articles, we extracted the information based on various data items (see supplementary materials). We classified the information in: (i) demographics and context, (ii) SA for ML challenges (RQ1), (iii) SA for ML solutions and tactics (RQ2), (iv) data types used. To analyse the demographics data we used descriptive

Table 2.3: Interview participants profiles.

ID	Position	Experience	Research
P1	Solutions Architect	3-5 years	No
P2	System Architect	3-5 years	Yes
P3	Software Architect	6-9 years	Yes
P4	Tech. Lead	3-5 years	Yes
P5	Software Architect	6-9 years	Yes
P6	Software Architect	3-5 years	No
P7	Director of Engineering	6-9 years	No
P8	Senior Solutions Architect	3-5 years	No
P9	Head of Engineering	3-5 years	No
P10	CTO	0-2 years	Yes

Table 2.4: Interview organisation profiles, where the acronym "Trans." stands for Transportation.

ID	Org. Profile	Org. Size	Org. ML Experience	Team Size	Deployment Interval
P1	Tech. (Internet)	10 000+	6-9 years	6-9	0-1 week
P2	Non Tech. (Trans.)	10 000+	3-5 years	6-9	1-2 weeks
P3	Tech. (Automation)	10 000+	3-5 years	10-15	3-4 weeks
P4	Tech. (AI/ML)	0-50	0-2 years	10-15	0-1 week
P5	Non Tech. (Medical)	10 000+	3-5 years	6-9	3-4 weeks
P6	Tech. (Automation)	1000-5000	3-5 years	10-15	1-2 weeks
P7	Tech. (AI/ML)	51-200	3-5 years	10-15	1-2 weeks
P8	Tech. (AI/ML)	51-200	3-5 years	6-9	3-4 weeks
P9	Tech. (Space)	51-200	0-2 years	10-15	1-2 weeks
P10	Tech. (Robotics)	0-50	1-2 years	16-20	3-4 weeks

statistics. To extract the data for (ii) and (iii), we used qualitative analysis methods. In particular, we used thematic analysis [35], which defines a process based on the following 5 steps: (1) familiarity with data – the articles were examined to form initial ideas, (2) initial code generation – the initial list of challenges and solutions was extracted, (3) theme search – common elements between the challenges and the solutions, respectively, were identified, (4) theme review – challenges and solutions were compared, and common items were merged or dropped, (5) definition and naming – each challenge and solution was defined and named. The results from the demographics analysis were presented above, and the ones from the thematic analysis are introduced in Section 2.4.1.

Interviews. To complement the results from the SLR, we conducted 10 interviews with participants from 10 organisations.

Protocol. The interview protocol was designed following the guidelines from Hove and Anda [110], and consisted of 31 questions designed to support a natural conversation between the participants. All interviews were conducted online, through video calls

(8 interviews) or e-mail (2 interviews). To enable participants to become familiar with the interview objectives [110], we shared a short version of the interview three days in advance.

The interviews were structured in 5 sections. First, we described the research goals and background. Second, we asked participants to share information about their background and demographics. Third, we asked participants to select a project where they played the role mentioned in the second section, and describe the constraints and challenges faced in SA for ML. This part enabled a discussion about the challenges faced (and the solutions adopted), and was meant to complement the data obtained previously. Next, we asked participants to comment on each of the challenges from the SLR, evaluate their impact on SA, and propose solutions. Last, we asked participants to provide open-ended comments. We continued to refine the questions, and after the first three interviews the questions remained stable. Two questions were merged due to redundancy, and one was modified in order to be more descriptive.

Participants. The interview participants were recruited using purposeful sampling [192]. We contacted participants with experience in (re-) architecting systems with ML components, or involved in architectural decisions (e.g., had the role of architect, or a leading position in engineering), and who are working (or worked) for companies using ML. To identify the participants, we used our personal network of contacts. Moreover, we compiled a list of organisations that use ML from outlets such as Forbes or MIT Technology Review. Later, we traced back candidates from the organisations (holding positions linked to SA) through LinkedIn, and contacted them. The list of participants and the data regarding their background is presented in Table 2.3, while the data regarding their organisations are presented in Table 2.4. We observe that the participants' background is diverse, ranging from software and system architects to engineering leaders and CTOs. Moreover, the participants' and organisations' experience is diverse – ranging from start-ups to large organisations with vast experience in ML. Since the organisations had different profiles, we classified them into (i) *Technology (Tech.)* – focus on developing technology products, and (ii) *Non-Tech.* – do not focus on technology products, but use ML for their processes. We also note that many participants had experience in research, being directly involved or in close collaborations with research groups. We hypothesise that the research driven process for ML contributes to this result.

Data analysis. The interviews were processed using thematic analysis, a technique which consists of the five steps recommended by Cruzes and Dyba [60]: (i) data extraction – the interviews were transcribed, read, and key points were extracted, (ii) data coding – the initial SA challenges and tactics, as well as the impact of each challenge on SA (e.g., low or high impact) were defined, (iii) code to themes translation – for each transcript the initial codes were combined into potential themes (e.g., automated testing), (iv) high-order theme modelling – the themes were compared and merged, or dropped if the evidence was not sufficient (e.g., automated testing was merged in CI), (v) synthesis assessment – arguments for the extracted data were established, for example in terms of credibility (if the core themes were supported by the evidence) or confirmability (if there was consensus among the authors on the coded data).

Survey. To generalise the findings with a large sample size and augment the solutions, we ran an online survey. The survey was developed using the guidelines from Kitchenham

and Pfleeger [135] and Ciolkowski et al. [55]. We designed a cross-sectional observational study asking participants at the moment of taking the survey which solutions they adopt, for each challenge. Moreover, we asked participants about their background in order to assign them to groups; making the study a concurrent control study in which participants are not randomly assigned to groups.

Questionnaire. The questionnaire consisted of five sections. In (i) the preliminaries we asked participants about their background (5 questions), to select a recent project where they played a role in SA for ML, and to provide information regarding the challenges faced, the project constraints, and the data types used (3 questions). Next, we asked participants to (ii) select or propose new solutions for the challenges identified previously (20 questions). Since multiple solutions involved instrumentation, monitoring or alerts, we added a question regarding the architectural decisions for designing these modules (1 question). Afterwards, we asked participants to (iii) select the architectural style (if any) adopted in their project (1 question), and to (iv) link the solutions to software quality attributes (1 question). The questionnaire ended with a section where (v) participants could provide open ended feedback (1 question).

The answers allowed multiple choices, with the solutions extracted from the SLR and from the interviews. Besides, we provided an open answer called 'Others', where participants could propose new solutions. The quality attributes used in the fourth section were extracted from the ISO/IEC 25010 standard [118], which is widely regarded as mature. However, we found the "Installability" and "Replaceabiliy" attributes out-dated, and replaced them with "Deployability"; which better reflects deployment and roll-back.

Survey Pilot. Before distributing the survey, we invited four candidates to assess the survey in our presence, and suggest improvements. The participants did not consider any question redundant. Using their feedback, we added three new answers to the questions, rephrased four other answers, and two questions.

Distribution. To distribute the survey, we used a snowballing strategy. At first, we reached out to our network of contacts, asked them to fill in the survey and forward it to potential candidates. Second, we expanded the list of contacts from interview recruitment. In total, we sent 286 e-mails or private messages to potential participants. Third, we advertised the survey through open channels used by practitioners, i.e., Reddit and LinkedIn.

Data Analysis. We processed the standard answers using descriptive statistics, and the open-ended answers using thematic analysis. Moreover, we analysed the association between the adoption of solutions using the Phi coefficient.

2.4 Results

We present the results from the three stages of the study as follows: (i) the results from the SLR are presented in Section 2.4.1, (ii) the results from the interviews are presented in Section 2.4.2, and (iii) the results from the survey are presented in Section 2.4.3.

2.4.1 Results from the SLR

From the SLR, we identified an initial set of 18 challenges, introduced in Table 2.6. We note that the SLR data have numerical references. To classify the practices, we used a custom

taxonomy because the taxonomy for ML is different than that for traditional software development [8]. Moreover, ML taxonomies are divergent [245]. For example, Amershi et al. [8] present a nine-stage taxonomy for the ML process, while Sato et al. [230] use only six stages. These taxonomies have roots in the CRISP-DM model [294]. However, recent studies show these models are not fit for all contexts [97]. Since existing taxonomies are divergent, we constructed a taxonomy compatible with previous work, but with a SA focus.

The taxonomy was used to classify the challenges (and solutions) in: (i) *Requirements (Reqs.)* – requirements elicitation for ML components, mapped to model requirements and business understanding [8, 294], (ii) *Data* – data collection, preparation and validation, mapped to data taxonomies [8, 245, 294], (iii) *Design* – the system's structure, SA decisions and trade-offs, mapped to training and coding taxonomies [8, 245], (iv) *Testing* – testing and validation of software with ML components, mapped on the evaluation taxonomies [8, 294], and (v) *Operational (Ops.)* – deployment, monitoring and evolution, mapped to deployment taxonomies [8, 248, 294].

RQ1. Answering RQ1 from Table 2.1, we identified 18 challenges through the SLR, classified in five categories. The *Reqs.* challenges focus on the inability to understand a project and estimate the effort upfront. Moreover, the opaque nature of ML components, for which functional requirements are difficult to define, and which have regulatory restrictions, emerged as challenging.

The *Data* challenges relate to data preparation and data quality assessment. This result contrasts previous concerns from big data and analytics platforms [237], where the focus was on data storage and accessibility. Nonetheless, this result corresponds with the expectation that ML components are evolved from big data platforms, extend and overcome the challenges met there.

The largest category of challenges, *Design*, includes both traditional SA challenges, such as managing component coupling, and new ML specific challenges, such as managing inherent uncertainty, or designing for development automation (AutoML). We also notice a challenge regarding the integration of ML components with traditional software components (7), which finds it difficult to distinguish failures between the two.

In contrast to Design, the *Testing* challenges are ML specific. Here, the focus is on model testing – which goes beyond programming bugs – and on validation for production – which does not rely on new features or bug fixes, but on measurements that must meet multiple criteria, e.g., accuracy or robustness.

In the *Ops.* category, the challenges relate to deployment, maintenance, and resource usage between training and testing. We note that maintenance of ML components is based on retraining and deploying models trained with new data, which erodes the boundaries between maintenance and evolution.

RQ2. Answering RQ2 from Table 2.1, through the SLR we found distinct solutions to each challenge in Table 2.6. The complete list of solutions extracted from the SLR is delegated to the supplementary materials, while Table 2.7 presents the solutions from all stages of the study. We note that, in total, 54% of solutions came from the SLR, while the rest came from later stages of the study. Although the SLR accounts for the majority of solutions, the percentage is just over 50%, and suggests that many solutions were considered incomplete by interview paticipants.

2.4.2 Results from the interviews

The interviews were meant to qualitatively assess and complement the SLR data. As mentioned in Section 2.3, the interviews had specific questions to discover new challenges, to evaluate the impact of each challenges on SA, and to propose new solutions.

Two new challenges were added after the interviews, and several others were reinforced. The first new challenge, (19), relates to tracing back serving decisions to ML models and data, and to the ability to accurately reproduce past experiments. This challenge brings together two concepts – traceability and reproducibility – both known to raise issues in ML [206]. Only one interview participant mentioned this challenge can have a significant impact on SA. Nonetheless, we included it, in spite of the fact that we did not have convincing evidence, and sought validation with the survey.

The second challenge, (20), relates to managing multi disciplinary teams, which use heterogeneous technology stacks (e.g., ML frameworks, infrastructure scripts). Since this challenge does not fit any previous class, we defined a new class – *Organisation (Org.)* – which gathers organisation wide concerns that fall in the attributes of software architects. This class aligns with the view that software architects shall consult and bridge multiple teams, which solve problems beyond SA [141]. The challenge was mentioned by one participant, part of a large organisation with well established teams, who work at different levels of the technology stack. Therefore, the solution was to form multi-disciplinary teams which can work close together, and adopt standard ways of working. No participant from small organisations raised this challenge, which begs the question if small organisations are more agile, and can overcome it. The answer was sought with the survey.

We also asked participants about the most important architectural decision drivers, and about the data types used. The results are illustrated in the supplementary materials. We note that “Scalability", “Hardware" constraints, and “Data" concerns were mentioned as main decision drivers, followed by “Interpretability". Together with the data type used, we could also identify the main decision drivers for specific data types. Here, we note that participants using Images & Videos or Time Series found “Scalability" and “Hardware" constraints as the main decision driver. Moreover, participants using Simulations were also driven by “Hardware" constraints. We also observed a new decision driver – called “Generalisation" – which describes the ability of a ML component to maintain training performance in production. This driver is related to challenge (10) (Table 2.6), and the solution suggested by participants was to use n-versioning; i.e., multiple versions of ML components (some of which may be more trustworthy).

While evaluating the challenges extracted from the SLR, we asked participants to assess their impact on SA. The results are illustrated in Figure 2.1, and use an ordinal scale with three possible options: low, medium, or high impact. Challenge (3) could not be evaluated because the participants did not report regulatory restrictions. We believe this result is due to the fact that ML regulations are still in draft phase, and not yet enforceable [123]. Within the challenges with the highest impact, we observe one traditional challenges that is strengthened by ML (component coupling (8)), and multiple ML specific challenges. For example, opaqueness of ML components (11), or training-serving resource management (18). The highest impact on SA comes from the need to continuously retrain ML components (16), while the lowest impact comes from automation of ML tasks (13). Here, participants

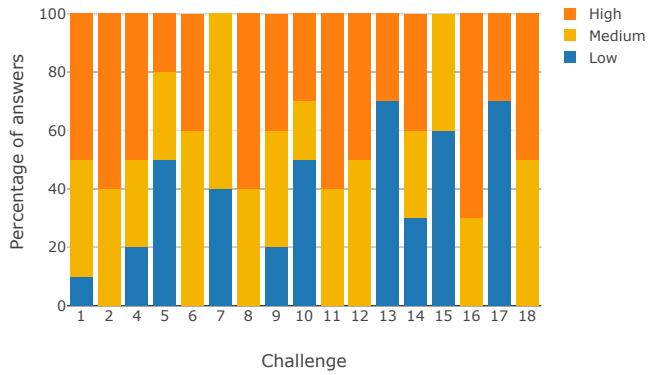


Figure 2.1: The impact of the challenges from Table 4.8 on SA, as assessed by interview participants.

reported that the information needed for this task is available from other sources.

Besides the challenges, the interviews allowed us to complement the initial set of solutions. We delegate the complete thematic analysis of the interviews to the supplementary materials, and mention that 46 % new solutions came from the interviews. Participants provided new solutions for all challenges besides challenges (2) and (3). While the solutions for (2) were regarded as complete, (3) was disregarded because participants did not reported regulatory constraints.

During the thematic analysis, we combined several solutions by bridging ML and SE terminology, while striving for conciseness. Here, we describe resulting themes which may be ambiguous due to name compression. Using the same type of interfaces for business logic and ML components, for all ML components, or within all projects in an organisation, was modelled as the use of “standard interfaces". Participants reported multiple techniques to standardise the interfaces, e.g., REST APIs, gRPC, or more general contracts for service oriented architectures. While the techniques are project specific, the architectural decision to unify the interfaces is singular.

Moreover, using multiple versions of a ML model – also called ensembles of models in ML – is similar to n-version programming. Therefore, we grouped these solutions in the “n-versioning" theme.

The separation of concerns and encapsulation of code was modelled as one theme: “design separate modules/services". Here, participants reported that the code was either developed as separate modules, or as independent services. The development implied encapsulation for reuse.

Furthermore, we defined the use of one middleware for all components in training and serving as “use one middleware", and the development of dashboards in “visualisations". A detailed description of the themes is provided in the supplementary materials.

2.4.3 Results from the survey

In total, we received 52 answers, from which we filtered out (using the preliminary questions) respondents who did not play a role in SA for ML. Moreover, we filtered out

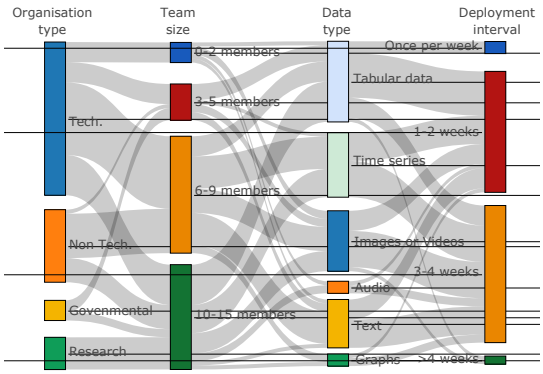


Figure 2.2: Distribution of survey respondents by demographics

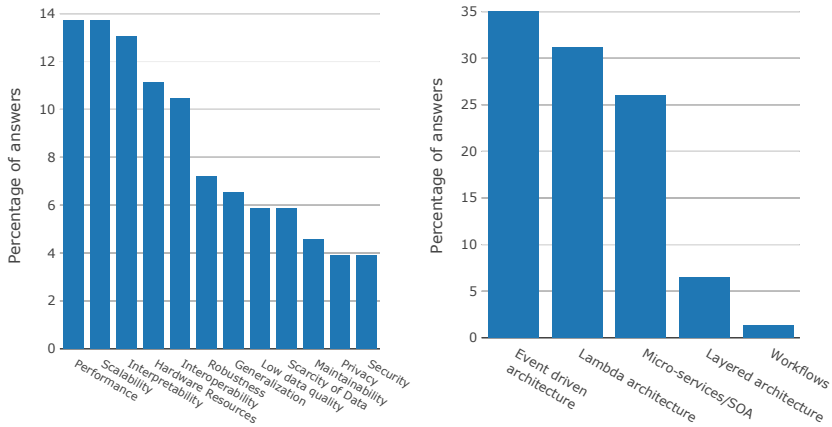
respondents who spent less than two minutes fulfilling the survey, and respondents who answered less than 50 % of the preliminaries, or less than 50% of the technical questions. This process ensured only thoughtful answers were used in the analysis, and entailed 47 complete answers.

Demographics. First, we grouped respondents by demographics. The complete analysis is delegated to the supplementary materials. We note that the majority of respondents (57%) work for Tech. organisations, and have between 3-5 (40%) or 1-2 (28%) years of experience. These results align with related work [117, 245], and are in line with expectations that Tech. companies are early adopters of ML technologies. Other groups are also represented, i.e., Non Tech. (28%), Governmental Org. (9%) and Research labs (6%). Similarly, beginners which just started (13%) and very experienced respondents, with 6-9 years of experience (19 %), are represented.

We also grouped respondents by regions into Europe (53%), North America (34%) and Asia (13%). Here, we observe a slight over-representation of Europe, and under-representation of Asia. The possible bias stemming from the grouping by regions will be discussed in Section 2.5.

As in the interviews, we asked respondents about their team size, data types used, and deployment intervals. This data is illustrated in Figure 2.2, where the height of the bars represents the percentage of respondents, and the height of the connections represents the percentage of respondent who fall in the target class. We observe that the majority of respondents belong to teams between 6-9 (43%) or 10-15 (34%) members. In particular, the majority of Tech. and Non Tech. teams have between 6-9 members, while the majority of Research teams are larger, between 10-15 members.

Regarding data types, we observe that, with the exception of Audio and Graphs, the data types have similar distributions. Moreover, the majority of respondents using Tabular data deploy new versions between 1-2 weeks, while respondents using Images, Videos, Audio or Text between 3-4 weeks. We conjecture that this result relates to the ML techniques suitable to each data type, i.e., Images, Videos, Audio or Text models are based on deep learning, require longer training times and the collection of larger data sets. In contrast,



(a) SA decision drivers.

(b) Architectural styles.

Figure 2.3: Characterisation of survey results using (a) SA decision drivers, and (b) architectural styles.

Tabular data can be processed with more traditional ML techniques (e.g., Random forest). These techniques require smaller data sets and training time.

Overall, the demographics indicate that our survey data is diverse, and resembles data from interviews and related work [117, 245].

Decision drivers. Second, we asked respondents about the most important decision drivers in their projects. This data is illustrated in Figure 2.3a. We observe that “Scalability" and “Hardware" are consistent with the interviews, and occupy leading positions. To better understand the data challenges, we divided them into "Low data quality" and "Scarcity of data". Taken together, data related concerns are consistent with the interviews. Separately, they were considered equally important, but none of them ranks high.

We also note that “Performance", “Interpretability", and “Interoperability" rank higher for survey respondents, while “Privacy" and “Security" rank very low. This result is cause of concern, since documents from policy makers and advisory bodies suggest these topics are paramount for trustworthy development of ML [106]. We conjecture that, although a large body of academic literature on security of ML exists, it is still limited in its applicability. For example, all defences against adversarial examples – a known threat for ML components – have been breached [45]. The data also indicates that respondents prioritise operational attributes, such as scalability or performance, and tend to neglect security and privacy.

Solutions to challenges in Table 2.6. Third, using the survey results we filtered out and ranked the solutions from previous stages of the study. In particular, we considered the solutions that were selected less than 5% of the time as not relevant, and filtered them out. Moreover, we used the number of times the solutions were selected by respondents to rank them. The ranking is reflected in Table 2.7 by the order in which the solutions are presented. A more comprehensive analysis of the answers and an elaborate description of the solutions is provided in the supplementary materials.

For all challenges, respondents could also suggest new solutions, or provide comments using the "Other" field. In total, we received 3 suggestions and open comments. We analysed

the results using thematic analysis, and found out that all suggestions were variations of the solutions provided, or comments suggesting some solutions do not apply. For example, one respondent mentioned that, due to tight performance constraints, it was not possible to apply n-versioning. The comment suggests that, given exceptional constraints, some solutions do not apply. This result is expected, since the first two stages of the study strove for generalisation; and outliers may exist. Nonetheless, the lack of novel suggestions for solutions brings evidence that the first two stages of the study entailed comprehensive solutions to all challenges.

We also note that some solutions are recurrent, and can be applied to multiple challenges. For example, the use of standard interfaces for ML components and business logic, or the use of interpretable models. These results are unsurprising, since architectural decisions may impact multiple elements of a system.

As mentioned previously, an extra question was added for decisions regarding instrumentation, monitoring and alerts. The solutions were inspired by interviews; where participants reported the development of independent logging, alert or visualisation modules. We delegate the survey answers to the supplementary materials, and note that the majority of respondents reported the development of independent modules/services for instrumentation and monitoring. Moreover, respondents reported on separating logging concerns between training and serving, and on the development of independent modules to aggregate and visualise logs. Only a small percentage of participants reported the use of external tools for instrumentation.

Associations between solutions. Fourth, we analysed the associations between the adopted solutions. For this analysis, we modeled the adoption of solutions as dichotomous variables, and analysed the Phi coefficient. For two binary variables, the Phi coefficient can be estimated using the Pearson correlation coefficient [95]. To determine the statistical significance of the observed associations, we performed Chi-Square tests with a significance level of 0.05. We found multiple significant medium to strong associations ($\phi > 0.4$), of which we report an illustrative selection. More results are provided in the supplementary materials, together with an analysis of the Jaccard similarity entailing analogous results.

For example, designing separate modules/services for data quality assessment (5) is associated with the the design of independent modules/services in component coupling (7) ($\phi = 0.43$). Moreover, the use of one communication middleware to reduce coupling (7) is associated with standardisation and reuse of model interfaces between training and serving (6) ($\phi = 0.51$). Design of independent modules/services in component coupling (7) is also associated with CI/CD in maintenance of ML component (16) ($\phi = 0.54$). These results indicate the solutions may be complementary, and suggest their joint adoption can be interdependent and incremental.

Similarly, model tests (15) are associated with data tests (15) ($\phi = 0.63$), and integration tests (15) are associated with test automation (15) ($\phi = 0.88$). However, ML tests (e.g., data tests) are not associated with traditional software tests (e.g., unit tests), or with test automation. These results indicate a separation between ML and SE concerns exists. Moreover, they indicate that mature teams jointly adopt test practices, as also noticed in [36].

Architectural styles for ML. Next, respondents were asked to select the architectural styles employed in their projects. The results are illustrated in Figure 2.3b, and indicate that the majority of respondents used the event-driven style. Nonetheless, the difference

Table 2.5: Solutions mapping to ISO/IEC 25010 model.

Characteristics	Sub-characteristics	Solutions
Func. Suitability	Func. Completeness	1, 2, 3
	Compliance	3
Performance efficiency	Capacity	2, 11
Compatibility	Co-existence	7, 8
	Interoperability	6, 12
Usability	User Error Protection	19, 20
Reliability	Availability	10, 16, 18
	Fault Tolerance	9, 10
Security	Accountability	19
Maintainability	Modularity	4, 5, 7, 8, 12
	Reusability	4, 5, 6, 8
	Analysability	11, 17
	Modifiability	4, 5
	Testability	5, 7, 9, 10, 11, 14, 15
Portability	Adaptability (including Scalability)	13, 16, 18
	Deployability (including Installability and Replaceability)	12, 16, 17, 18

between event-driven, lambda, and micro-service/SOA architecture styles is not large. Although we did not find significant associations between the architectural styles, the lambda architecture can be used concomitantly with other architectural styles. We also searched in literature for evidence to support the architectural styles, but did not find any study on this topic. Therefore, we abstain from drawing a firm conclusion regarding the most suited architectural style for software with ML components (if any), and propose to gather more data on this topic in future research.

Quality attributes. Last, respondents were asked to link the solutions to software quality attributes (characteristics) from ISO/IEC 25010 [118], which enabled to restate them as architecture tactics [22]. Tactics are architectural building blocks from which design patterns can be created, and represent architectural decisions that improve individual quality attributes [22, 99]. Therefore, the results of this analysis provide direct guidance for practitioners who aim to improve specific quality attributes of systems with ML components.

Since the solutions do not presume a ranked order, we considered all solutions equally important. The final results are presented in Table 2.5. We note that “Scalability" and “Interoperability" – considered important decision drivers (Figure 2.3a) – are addressed by multiple solutions. Similarly, “Maintainability", considered to have the biggest impact on SA by interview participants (Figure 2.1), is addressed by the largest number of solutions. We also observe that some quality attributes from the standard (e.g., Operability or Maturity) are not addressed by any solution, and note this result does not imply that missing quality attributes are not challenging. Instead, some quality attributes may not be applicable, or

require adaptation to accommodate ML components, as previously suggested by Kuwajima et al. [143]. We plan to investigate this conjecture in future research.

We also mention that “Compliance" sub-characteristics are not present in the quality standard because compliance is considered part of the overall system requirements. Therefore, “Compliance" spans all characteristics in Table 2.5. To avoid confusion, we represent “Compliance" as a sub-characteristic of “Functional Suitability".

2.5 Discussion

We comment on multiple aspects left open in the paper. First, regarding the challenges discovered from interviews, we analysed the percentage of survey respondents who did not have a strategy to tackle them. We found out that 23% of respondents had no strategy for challenge (19), and 22% of respondents had no strategy for challenge (20). These results show that more than 75% of respondents tackled these challenges, and bring evidence that both challenges are relevant, in spite of the fact that they were mentioned in one interview each. Moreover, the answers for challenge (20) have similar distributions for teams consisting of 6-9 and 10-15 members, suggesting the challenge is not motivated by team size. Similar analyses for smaller team sizes are planned for future research, as the data collected until now can not entail robust conclusions.

Second, both in the interviews and in the survey, the architectural decision drivers for trustworthy ML [106] – i.e., “Robustness", “Security", “Privacy" – were not considered important by respondents. McGraw et al. [169] argue that, from a security engineering perspective, the SA of systems with ML components is an important first step. However, our survey results show a different perspective. Besides the concerns stemming from neglecting these decision drivers, the result may indicate that decisions regarding trustworthy ML can be made after the SA of a system is defined. We plan to investigate this in future research.

Third, the solutions presented in Table 2.7 do not take into account functional dependencies, or presume a functional ranking. Nonetheless, the results from the association analysis (Section 2.4.3) indicate the solutions may be complementary, and their joint adoption interdependent or incremental. Similarly, by analysing the associations between solutions and architectural decision drivers, we found out that multiple solutions are associated with decision drivers. We expect that results from this type of analyses will provide step-wise guidance to practitioners searching for tactics to address individual drivers, and plan to develop such analyses in future research, once more data is collected.

We also noticed that some solutions have low adoption. For example, self-adaptation for managing inherent uncertainty (9) was used by less than 5% of respondents. While this result may seem surprising, according to Mahdavi-Hezavehi et al. [163] the number of self-adaptation techniques for “automated learning”, i.e., ML components, is small. We conjecture that a small number of solutions are not applied because they are not fit, or still prototypes in academia, and plan to investigate them in future research.

Last, when mapping the solution to software quality attributes, we used the mature and authoritative ISO/IEC 25010 standard [118]. At the moment, no similar model exists for ML components, although policy makers indicate such models are under development [123]. The results from our study indicate that, with the exception of “Interpretability", ML specific quality attributes are not considered important SA decision drivers. We expect this to

change once mature quality models for ML are available, and propose to extend our analysis to cover them in future research.

Threats to validity. We identified three potential threats to validity, corresponding to the three stages of the study. First, the SLR can be affected by missing or exclusion of relevant papers. To mitigate this threat, we used multiple digital libraries for information retrieval. Additionally, we complemented the results with grey literature (manual search), and through snowballing. The researchers' bias in the data extraction was prevented using a data extraction form, which allowed consistency in data analysis, and through discussions between authors with different backgrounds.

Second, the data from interviews may be subject to bias. To limit this bias, we analysed the participants' profiles and ensured they have relevant experience for the study. We recruited participants with diverse backgrounds and experience, working for organisations with distinct sizes and experience in ML. We also used two strategies to alleviate memory bias, i.e., we shared a short version of the interviews before the meeting, and asked participants to share their experience from a recent project in the preliminaries. Moreover, we assured participants of data confidentiality and anonymisation, in order to limit participants from answering the questions in a manner that would better position them, or the organisation they are part of.

Third, to limit the survey bias we included additional fields besides the answers from SLR or interviews (e.g., Other fields for all challenges). We also advertised the survey to diverse groups, in order to limit selection bias. Nonetheless, as shown in Section 2.4.3, some groups of respondents are under-represented, and may introduce selection bias. This bias can be removed by gathering more data, as we plan to do in the future. Last, to avoid researchers' bias, we used data triangulation from multiple sources.

2.6 Conclusions and future research

We studied how systems can be (re-)architected to enable robust adoption of ML components. We ran a mixed-methods empirical study consisting of: (i) a SLR which revealed 42 relevant articles, from which we complied 18 SA challenges (and solutions) for ML, (ii) 10 semi-structured interviews which revealed 2 new challenges and 46 new solutions, and (iii) a survey with 47 architects to quantitatively validate the solutions.

We reported on the impact of each challenge on SA, and the main SA decision drivers for ML. We found out, for example, that ML heighten decision drivers, such as privacy, are considered marginally important when compared to traditional decision drivers, such as scalability or interoperability. Moreover, we established a link between solutions and quality attributes from the ISO/IEC 25010 standard, which allowed us to provide practitioners with twenty architectural tactics for systems with ML components.

For future research, we plan to further increase the number of respondents of the survey, in order to perform more robust analyses. For example, we plan to create a stronger link between tactics and decision drivers using association analysis. Moreover, we plan to perform in depth analyses of the architectural styles suitable for ML. We also plan to add depth to the interpretation of our findings through validation interviews, and expand the quality attributes from ISO/IEC 25010 with ML specific quality attributes.

Table 2.6: List of SA challenges for ML.

Nr.	Category	Challenges	References
1	Reqs.	At design time the information available is insufficient to understand the customers or the projects.	[28, 51, 117, 154, 160, 282]
2	Reqs.	ML components lack functional requirements.	[28, 51, 63, 117, 160, 282]
3	Reqs.	ML projects have regulatory restrictions and may be subject to audits.	[79, 127, 186, 245]
4	Data	Data preparation may result in a jungle of scrapes, joins, and sampling steps, often with intermediate outputs.	[63, 151, 236]
5	Data	Data quality is hard to test, and may have unexpected consequences.	[63, 151, 168, 208, 312]
6	Design	Separate concerns between training, testing, and serving, but reuse code between them.	[8, 301, 314]
7	Design	Distinguish failures between ML components and other business logic.	[213, 304]
8	Design	ML components are highly coupled, and errors can have cascading effects.	[109, 191, 282]
9	Design	ML components bring inherent uncertainty to a system.	[12, 109, 191, 244, 248]
10	Design	ML components can fail silently. These failures can be hard to detect, isolate and solve.	[33, 248, 300]
11	Design	ML components are intrinsically opaque, and deductive reasoning from the architecture artifacts, code or metadata is not effective.	[109, 191, 232, 314]
12	Design	Avoid unstructured components which link frameworks or APIs (e.g., glue code).	[236]
13	Design	Automation and understanding of ML tasks is difficult (AutoML).	[151, 218, 245, 282, 301]
14	Testing	ML testing goes beyond programming bugs to issues that arise from model, data errors, or uncertainty.	[8, 14, 189, 216, 312]
15	Testing	Validation of ML components for production is difficult.	[230]
16	Ops.	ML components require continuous maintenance, retraining and evolution.	[25, 154, 191, 230, 282, 286, 312]
17	Ops.	Manage the dependencies and consumers of ML applications.	[23, 73, 109, 236, 304]
18	Ops.	Balance latency, throughput, and fault-tolerance, needed for training and serving.	[47, 151, 173, 287, 301]
19	Ops.	Trace back decisions to models, data and reproduce past results.	P10
20	Org.	ML applications use heterogeneous technology stacks which require diverse backgrounds and skills.	P1

Table 2.7: List of solutions.

Nr.	Solutions
1	Run simulations to gather data. Use past experience. Measure and document uncertainty sources.
2	Use metrics as functional requirements. Include understandability and explainability of the outputs.
3	Analyse regulatory constraints up-front. Adopt an AI code of conduct. Design audit trails.
4	Design separate modules/services for data collection and data preparation. Integrate external tools.
5	Design separate modules/services for data quality assessment. Integrate external tools.
6	Standardise model interfaces. Use one middleware. Reuse virtualisation, infrastructure and test scripts.
7	Separate business logic from ML components. Standardise interfaces and use one middleware between them.
8	Design independent modules/services for ML and data. Standardise interfaces and use one middleware. Relax coupling heuristics between ML and data.
9	Use n-versioning. Design and monitor uncertainty metrics. Employ interpretable models/human intervention.
10	Use metric monitoring and alerts to detect failures. Use n-versioning. Employ interpretable models.
11	Instrument the system to the fullest extent. Use n-versioning. Employ interpretable models. Design log modules to aggregate/visualise metrics.
12	Wrap components in APIs/modules/services. Use standard interfaces and one middleware. Use virtualisation.
13	Version configuration files. Design the log and versioning systems to support AutoML data retrieval.
14	Design model and data tests. Use CI/CD. Use integration and unit tests. Use data ownership for test modules.
15	Use metrics and CI/CD for validation. Use alerts, visualisations, human intervention. Design release processes.
16	Design for automatic continuous retraining. Use CI/CD. Use automatic rollback. Use infrastructure-as-code. Adopt standard release processes.
17	Encapsulate ML components in identifiable modules/services. Use authentication and access control. Log consumers of ML components.
18	Design for batch processing (training) and stream processing (serving), i.e., lambda architecture. Physically isolate the workloads. Use virtualisation.
19	Design for traceability and reproducibility; log pointers to versioned artifacts, version configurations, models and data.
20	Form multi-disciplinary teams. Adopt an AI code of conduct. Define processes for decision-making. Raise awareness about ML risks within the team.

3

Towards Using Probabilistic Models to Design Software Systems with Inherent Uncertainty

The adoption of ML components in software systems raises new engineering challenges. In particular, the inherent uncertainty regarding functional suitability and the operation environment makes architecture evaluation and trade-off analysis difficult. In this chapter, we propose a software architecture evaluation method called Modeling Uncertainty During Design (MUDD) that explicitly models the uncertainty associated to ML components and evaluates how it propagates through a system. The method is based on Bayesian networks, which enable both qualitative and quantitative assessments of software architectures. In particular, the method supports reasoning over how architectural patterns can mitigate uncertainty and enables comparison of different architectures focused on the interplay between ML and classical software components. While domain-agnostic and suitable for any system where uncertainty plays a central role, we validate our approach using as example a perception system for autonomous driving. For this system, we empirically demonstrate that a component-based design is over 10% more resilient to uncertainty than an end-to-end design. Moreover, we bring empirical evidence that architecture design patterns can help to significantly decrease the uncertainty associated to ML components.

This chapter has been published as *A. Serban, E. Poll, J. Visser, Towards Using Probabilistic Models to Design Software with Inherent Uncertainty, European Conference on Software Architecture, 2020* [244].

3.1 Introduction

With the emergent adoption of ML components in software systems, there is an increased need to tackle and reduce their inherent uncertainty. Particularly when developing safety-critical systems – e.g., autonomous vehicles – where new developments in ML and especially deep learning (DL) are used to a great extent.

For a long time, researchers in software architecture have developed methods to tackle uncertainty at design [71, 170] or at run-time [70]. Previous work focused primarily on tackling uncertainty related to the parameters used to model a software system, its context or to the instrumentation [70, 71, 170]. ML (and particularly DL) components add a new type of uncertainty that was only briefly explored previously. This uncertainty comes from the incapacity to verify that these components satisfy their intended functionality, and that they are able to cope with stochastic events coming from the operational environment.

In this chapter, we introduce a method to evaluate architecture design alternatives for software using both traditional and ML components. The proposal, called Modeling Uncertainty During Design (MUDD), is based on two guiding principles. First, the threats due to inherent uncertainty of ML components are evaluated both locally (for the specific components) and tracked as they propagate and influence other components in the system. Second, the prior information about uncertainty of ML components that is used at design time is considered incomplete and subject to continuous change.

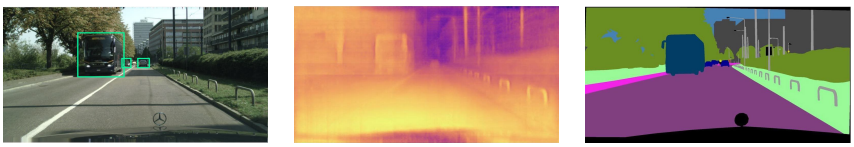
For modeling a software system using ML components we use probabilistic graphical models – in particular Bayesian networks (BNs) – which allow to express *beliefs* about variables (satisfying the second principle) and evaluate their *influence* on other connected variables (satisfying the first principle). The method only requires to annotate existing software architectures with design elements that express the inherent uncertainty of ML components. The formalism of BNs can then be used to obtain quantitative results for comparing architecture alternatives. We demonstrate our method using a visual perception system for autonomous driving, illustrated in Figure 3.1. The system is based on three components which can only be implemented using DL algorithms.

This chapter is organized as follows. In Section 3.2 we discuss background information and related work. Section 3.3 presents sources of uncertainty and BNs. Section 3.4 introduces MUDD and the running example. In Section 3.5 we develop a qualitative assessment of software architectures, followed by a quantitative assessment in Section 3.6, exhibiting MUDD. Section 3.7 uses the same method to assess the impact of using architectural design patterns to reduce uncertainty. A discussion and concluding remarks follow in Section 3.8.

3.2 Background and related work

We discuss three related research directions: (1) the use of uncertainty at design time for comparing architectural design alternatives, (2) the use of uncertainty at run-time for self-adaptation, and (3) the use of BNs in software architecture and reliability engineering.

At design time, the uncertainty in the parameters used to model a software system has been taken into account for evaluating the reliability of software architectures using robust optimisation [170], for comparing software architectures when the impact of architectural



(a) Object Detection (OD) (b) Depth Estimation (DE) (c) Semantic Segmentation (SS)

Figure 3.1: Visual perception system for autonomous driving.

decisions can not be quantified, using fuzzy methods [71] and for evaluating trade-offs specific to desired quality attributes such as performance, using sensitivity analysis [72].

The first two methods aim to achieve similar goals with ours, although they seem unsuitable for software using ML components. In the first case [170], the failure rate of ML components can not be evaluated as for traditional software components because in many scenarios these components will appear to behave as intended, although their outcomes will be erroneous or very uncertain [248]. Moreover, we are not concerned with deployment, as in [170], but with architectural design styles or patterns that can reduce the uncertainty stemming from defining software components and their interaction. Regarding [71], in this paper we take a bottom-up approach, assessing the uncertainty of each components and their impact on the architecture, instead of a top-down approach where the impact of architecture decisions is quantified at architecture level. We found it difficult to assess the impact of uncertainty on software architectures without first assessing the uncertainty of the components. Moreover, the uncertainty of ML components can not be precisely measured and it is subject to change (principle 2 from Section 3.1). Therefore, evaluating how the uncertainty of each component propagates in a system enables fine-grained reasoning about the sources of uncertainty that have a significant impact on the architecture, ways to remove them and the suitability of architectural patterns in doing so.

The evaluation of other quality attributes under uncertainty, such as performance [72] is complementary to the method introduced here and it is an interesting path to explore for future work.

Various sources of uncertainty [70] can be mitigated at run-time, of at design time, through self-adaptation. Self-adaptive systems collect data during operations and reconfigure or adjust their behavior in order to mitigate uncertainty [289]. Software architecture plays an important role in self-adaptation because an architectural model can be used at run-time to reason about self-adaptation. Although multiple uncertainty sources are used in self-adaptation, particularly relevant to us is the uncertainty related to "automated learning" [81, 290]. However, not many publications address this problem [163]. In this paper we tackle the problem at design time, and not at run-time, although some methods intended to work at run-time can be paired with the method introduced in this paper. We consider this an interesting direction to pursue in the future.

BNs have been previously used in software architecture as a support tool for design decisions – to quantify the impact of decisions on systems quality [308] or to measure the impact of changes [269]. Moreover, BNs have been extensively used in reliability engineering, to predict software reliability from architecture artifacts [222] or for fault detection [57]. In all cases, the model is based on the system's possible execution path, the

control flow graph or on the architectural structure of the system. While the execution paths are not available for ML components due to their opaque nature and lack of internal states, focusing on design level artifacts is related to the method introduced in this paper [222].

3.3 Uncertainty sources and Bayesian Networks

Perez-Palacin and Mirandola [204] and Esfahani and Malek [70] discuss multiple sources of uncertainty to be considered in the context of self-adaptation. Although many uncertainty sources are valid at design time, we are particularly interested in the uncertainty related to "automated learning" [81, 163, 290], i.e., the inherent uncertainty related to ML components.

In particular, we are interested in the uncertainty related to properties of ML components that are impossible to fully verify before deployment, i.e., (1) the ability of a ML component to always satisfy its intended functionality and (2) to cope with stochastic events in the operational environment. According to [204] these sources of uncertainty can be classified by nature in:

- *epistemic uncertainty (EU)* - captures our ignorance of the correct model that generates the data. If the training data for a ML component does not accurately represent the data generation distribution, the model will have high epistemic uncertainty. This uncertainty can be removed given enough training data. However, the bounds for the data set size needed to learn complex tasks, e.g., object recognition, are not achievable in practice.

- *stochastic uncertainty (SU)* - captures the response of a ML component to stochastic noise in the operational environment (e.g., noise in the observations). This uncertainty can not be removed with more training data.

We propose to use these two sources of uncertainty as architectural design elements, in order to evaluate design alternatives for systems using ML components. For modeling a software system we use the structure and formalism of BNs because they allow to express beliefs (or incomplete prior information) about a variable and evaluate its influence on other connected variables. Briefly, BNs are directed, acyclic graphs, that model a set of variables and their conditional dependencies. This model offers both a quantitative and a qualitative method to reason about a set of variables. The former, qualitative, analysis uses the topological structure of the BN where variables are represented as nodes of a graph and the dependencies between them as directed edges.

The latter, quantitative, analysis consists of specifying the conditional probability distributions (or tables in the discrete case) between dependent variables in the graph. Since a BN contains all assumptions about a model (the graphical structure, the conditional probabilities and other parameters), it has no hidden assumptions about inference rules. The network's structure defines a joint probability model where the rules of probability calculus enable conclusions based on observations. Probabilistic inference computes posterior probabilities for unobserved variables given observations of other variables in the model. For a formal introduction to BNs, we refer the reader to Pearl[203].

3.4 Modeling Uncertainty During Design

MUDD is a method to assess architecture design alternatives for software systems using both ML and classical components. In particular, the method focuses on reasoning about architectural design styles and patterns that can reduce the uncertainty stemming from defining software components and their interaction. MUDD evaluates the impact of uncertainty in a bottom-up manner, starting locally, as it impacts specific ML components, and tracked globally, as it influences other components in the system. This method allows fine-grained reasoning about design alternatives, where the changes between design alternatives can be evaluated at multiple levels.

Notably, MUDD supports reasoning over which design alternatives are less sensitive to uncertainty, and how design patterns can help mitigate it. Moreover, the method allows to evaluate hypothetical scenarios, in which the data about uncertainty used at design time is considered incomplete.

From a methodological perspective, MUDD only requires to annotate existing software architectures with the sources of uncertainty specific to ML components. Afterwards, the topology of the BN is used as a model for quantitative assessments and the formalism of BNs is used for quantitative assessments.

We emphasize that MUDD uses the two sources of uncertainty introduced in Section 3.3 because they are application and context *independent*, i.e., they are valid and can be extracted from any ML model. The methods used to measure them can be different, depending on the ML algorithm employed. Therefore, they are parameters rather than fixed elements of MUDD. Nonetheless, MUDD is not limited to these two types of uncertainty. In fact, any type of uncertainty, application or context specific can be used without any modification.

Throughout the paper we use an example from autonomous driving, inspired by [27, 34, 251] – the design of a perception system for scene understanding. The system performs the following three tasks:

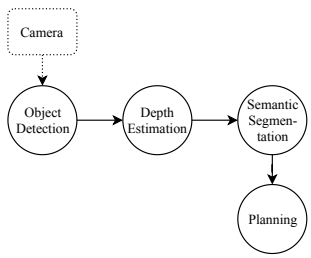
- *Object detection* (also called object localization) aims to identify the location of all objects in an input image, and classify them according to predefined classes. The output of object detection is an image with several boxes surrounding the objects, their labels and confidence scores for the classification.

- *Semantic segmentation* classifies each pixel of an input image to predefined classes, such as pedestrians or vehicles.

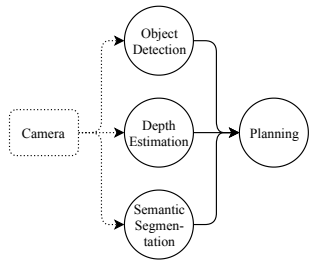
- *Depth estimation* (or understanding the geometry of the scene) is relevant to determine the position of other obstacles or the road surface.

All functions are illustrated in Figure 3.1. The outcome of the perception system is used in planning the next driving maneuvers of a vehicle. The functionality for all components is implemented using deep neural networks because no specification can be written for it, and other ML algorithms do not perform as good. We are interested to evaluate software architecture design alternatives and select the one which is the least sensitive to uncertainty.

In Figure 3.2a and Figure 3.2b we present two architecture candidates inspired from [251] and [27]. The relevant functional components are illustrated using circles, while the input

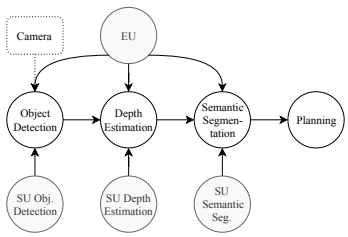


(a) End-to-end architecture.

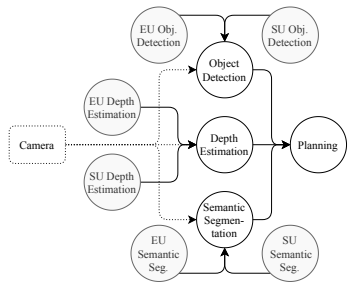


(b) Component-based architecture.

Figure 3.2: Functional architectures for a scene understanding system in autonomous vehicles.



(a) End-to-end annotated architecture, as required by MUDD.



(b) Component-based annotated architecture, as required by MUDD.

Figure 3.3: Uncertainty representation for the two architectures in Figure 3.2, where EU stands for epistemic uncertainty and SU for stochastic uncertainty.

coming from the camera is depicted with a rectangle. The latter will not be considered a node in the BN (therefore its shape).

The first figure illustrates the end-to-end paradigm, where all components of the system are jointly trained to form a representation relevant to planning. This corresponds to a pipe-and-filters implementation, recommended in [251] and adopted in [34]. The components are separated because they are trained using different objective functions and are subject to distinct drawbacks (also called multi-task learning in the ML literature). However, they all share a base network for feature extraction and have independent layers to decode the features for each task. An alternative architecture is presented in the Figure 3.2b, where the system is organized into distinct ML components and integrated during planning, corresponding to a component-based architecture from [27].

We have chosen these architectural styles as the only alternatives we could find in literature. However, MUDD is not limited to any architectural style.

For reasoning about uncertainties, we propose to annotate the the two architectures with the sources of uncertainty specific to each component. Since the architectures represent a directed graph, we only need to add and connect explicit nodes from the uncertainty sources to the components they influence.

Figure 3.3 departs from the functional view presented in Figure 3.2 by illustrating the uncertainty sources discussed in Section 3.3, for each component. In the first case, Figure 3.3a, one base encoder is used for all tasks. Therefore, only one node representing epistemic uncertainty (EU) influences all components.

Arguably, the internal representation of the encoder might hold different representations for each task, and be subject to distinct epistemic uncertainties. However, the internal representation is entangled and specific attributes corresponding to each task can not be easily extracted.

Different sources of stochastic uncertainty (SU) can impact the three tasks because one random event in the operational environment can influence segmentation, but not detection or depth estimation (and vice versa). Therefore, for each component there is a different variable for stochastic uncertainty. In the second scenario from Figure 3.3b, since the components process raw data from camera independently. Therefore, they are subject to distinct epistemic and stochastic uncertainties. We note that these decisions are not application and context specific. All ML components have these types of uncertainty.

The annotated architectures from Figure 3.3 represent the topology of a BN and can be used for qualitative analysis. For quantitative results, the topology needs to be enriched with probability data.

3.5 Qualitative architecture evaluation

Qualitatively, the annotated functional architectures from Figure 3.3 allow high level reasoning about threats to the intended functionality coming from inherent uncertainty. Nonetheless, the analysis is not limited to possible faults from uncertainty. Other quality properties, e.g., availability, can be assessed similarly.

In order to evaluate design alternatives qualitatively, architects can apply well known evaluation methods such as questioning techniques, scenarios or check-lists [66]. We develop an example of scenario-based analysis; a technique systematized in [128]. Scenarios analyze a use case or a change in a system. The change can describe how one or more components perform an activity, the impact of adding another component to perform the same (or another) activity, the impact of adding a connection between existing components or any composition of these factors. Describing the changes that are needed for a scenario is a qualitative method of architecture evaluation [66].

We analyze the scenario where high stochastic uncertainty in depth estimation leads to unsafe planning. From Figure 3.3, we can see that in the end-to-end architecture depth estimation is linked to semantic segmentation. Stochastic uncertainty can not be removed using more training data, as indicated in Section 3.3. Therefore, the alternatives are to use a different training objective or a different ML model altogether. However, since the scene understanding system is trained end-to-end, the new objective (or new model) can impact the internal representation of both segmentation and localization, which might increase their epistemic uncertainty, and ultimately lead to unsafe planning.

In the component-based architecture, depth estimation is only linked to planning. Therefore, high stochastic uncertainty in depth estimation can be treated in isolation, by either deploying a new DL component or using heterogeneous implementations to increase its confidence. If the model is replaced, it will have no impact on the other components in

the system. In this case, the change requires less effort than in the end-to-end architecture because changing the depth estimation component does not require to change or re-train other components.

We also analyze the use of LIDAR – as suggested in [27, 251] – to increase confidence in depth estimation and adopt the n-version programming pattern for safety critical systems to integrate it. The change consists in adding a new LIDAR component and a voting mechanism between the DL depth estimation component and the new LIDAR component [13]. An illustration is provided in Figure 3.5 and a more thorough discussion follows in Section 3.7. For both architectures, the output of the DL depth estimation components can be interpreted independently. Therefore, deploying the new components specific to LIDAR will have the same impact for both architectures. From the perspective of this change, both design alternatives are equivalent. However, the component-based architecture has a flexibility advantage when changing individual components.

During qualitative evaluation, it is important that different stakeholders and roles responsible for parts of the system participate. Such stakeholders may be algorithm developers, system and safety engineers or software architects. We can see that by explicitly modeling of uncertainty at design time, a broader community of practitioners can participate in design decisions, even though the domain knowledge needed to understand the inner workings of ML algorithms is not wide-spread. In the following section, we show how the same model enriched with quantitative measurements can be used to assess the impact of uncertainty on both individual components and on the system.

3.6 Quantitative architecture evaluation

As shown in Section 3.4, the enhanced functional architecture of a system has the topological structure of a BN. The probabilities needed to populate the network can be defined by experts or inferred through simulations – e.g., to estimate the effect of epistemic uncertainty on a ML model, we can use a test data set which was not used when training the model.

The random variables in the BN can take continuous or discrete values. In the former case, the system designer chooses an a priori distribution for each variable, before seeing any data, and updates its parameters once new observations are available. In the latter case, the variables take discrete values and are described by their probability mass functions.

For simplicity, we choose to model all variables through probability mass functions with two discrete values: *low* or *high* uncertainty. When the uncertainty is low, the system is likely to satisfy its intended functionality, and vice versa. Deciding if the uncertainty values are discrete or continuous is application and context specific. If the uncertainty in the ML components used in a system can be modeled as a probability distribution, and if the parameters of the distribution can be estimated well, a continuous approach may be better suited. Nonetheless, interpreting the parameters of a distribution requires more knowledge, and may complicate the architectural decision process.

Given the two proposed values for uncertainty, we are interested to evaluate the influence of different nodes in the network on planning, and obtain quantitative results for the qualitative evaluation. Both the probabilities and the thresholds can be decided by domain experts, or by simulation.

For the running example, we use a test data set (not used for training) to extract the uncertainty estimates from DL components, by averaging over samples in this data set. The thresholds between low and high represent the lowest uncertainty estimate from the incorrectly classified examples in the testing data set. The probability that a component has high (epistemic or stochastic) uncertainty will be the total number of test examples which have uncertainty higher than the threshold over the total number of testing examples. Note that the correctly classified examples with high uncertainty will contribute to the probability that a component has high uncertainty. This choice is deliberate, because the system we study is safety-critical, and uncertain decisions should be avoided.

The conditional probabilities – i.e., the influence of components to the connected components – can be evaluated in a similar manner. They represent the probabilities that a component has high uncertainty, given the uncertainty values of the parent variables. For example, $P(OD = \mathrm{H} | EP = \mathrm{H}, SU = \mathrm{H})$ is the probability that the object detector is highly uncertain when the model has high epistemic and high stochastic uncertainty. We use the same method and data set as before, but average the results when the parent variables have the same value. The thresholds are also chosen as before.

All experiments are carried out using the CityScape data set [58]. For the end-to-end architecture presented in Figure 3.2a, we train a variant of MultiNet [270] using an encoder based on the DenseNet [112] architecture, pre-trained on the ImageNet data set (as in [120]) with a dropout probability of $p = 0.2$. We use different loss functions in a multi-task learning setting for object detection, depth estimation and semantic segmentation. Epistemic uncertainty is approximated by casting a Bernoulli distribution over the model's weights and sample it at evaluation time using the dropout layers in the base encoder. The mean of the dropout samples is used for prediction and the variance to output the uncertainty for each class [4]. Stochastic uncertainty is extracted from the final layer of each task, as described in [130]. The geometry of the scene is interpreted using depth estimations from the base encoder, as in [69], while object detection and semantic segmentation have the same loss functions as in [270]. For the component-based architecture presented in Figure 3.2b, we use one independent encoder and decoder for each task. Training is performed by minimising the task specific loss function used in the multi-task setting described above. The implementation of DL components was done in Pytorch[1] and the BNs in Pomegranate[2]. The uncertainty estimates are presented in Table 3.1 for the system in Figure 3.2a and Table 3.2 for the system in Figure 3.2b.

The heuristics applied to populate the tables represent the prior knowledge we embed in the network. Depending on the context, software designers may choose to embed more domain knowledge, or rely on expert opinion. For example, in the context of autonomous driving we may choose to augment the testing set with common perturbations, specific to different seasons, driving conditions, or even malicious perturbations [103].

Given the probability tables, we can use the inference rules of BNs to answer questions about the proposed architectures. Coming back to the example presented in Section 3.5, we wish to get quantitative evidence about the impact of high stochastic uncertainty in depth estimation on planning. Setting depth estimation stochastic uncertainty to "High" ($SU_{DE} = \mathrm{H}$), we can compute the final impact on planning as follows. Let $\pi(x)$ represent

[1] https://pytorch.org/
[2] https://github.com/jmschrei/pomegranate

Table 3.1: Independent and conditional probabilities for the end-to-end architecture in Figure 3.3a. The acronyms used are OD – object detection, DE – depth estimation, SS – semantic segmentation, EU – epistemic uncertainty and SU – stochastic uncertainty. The uncertainty values are L - low and H - high.

$P(\cdot)$	EU	SU_{OD}	SU_{DE}	SU_{SS}
H	0.18	0.16	0.11	0.19

$P(Planning$	$SS)$
0.1	L
0.9	H

$P(OD$	EU	$SU_{OD})$
0.0	L	L
0.64	L	H
0.61	H	L
1	H	H

$P(DE$	EU	SU_{DE}	$OD)$
0.0	L	L	L
0.13	L	L	H
0.76	L	H	L
0.85	L	H	H
0.43	H	L	L
0.78	H	L	H
0.9	H	H	L
1	H	H	H

$P(SS$	EU	SU_{SS}	$DE)$
0.0	L	L	L
0.28	L	L	H
0.64	L	H	L
0.72	L	H	H
0.66	H	L	L
0.58	H	L	H
0.61	H	H	L
1	H	H	H

the parent variables of node x (the nodes that have a directed edge to it). The probability that planning will have high uncertainty is:

$$P(Planning = \text{H}) = P(SS|\ \pi(SS)) \cdot P(DE|\ \pi(DE)) \cdot P(OD|\ \pi(OD)) \cdot P(SU_{SS}) \cdot P(SU_{DE} = \text{H}) \cdot P(SU_{OD}) \cdot P(EU),$$

for the end-to-end architecture, and:

$$P(Planning = \text{H}) = P(SS|\ \pi(SS)) \cdot P(DE|\ \pi(DE)) \cdot P(OD|\ \pi(OD)) \cdot P(SU_{SS}) \cdot P(SU_{DE} = \text{H}) \cdot P(SU_{OD}) \cdot P(EU_{SS}) \cdot P(EU_{DE}) \cdot P(EU_{OD}),$$

for the component-based architecture, where the acronyms are as in Table 3.1.

Running the computation, we observe that the probability of uncertain planning is approximately 10% lower for the component-based architecture (Figure 3.2b) than for the end-to-end architecture. Moreover, through the same model we can analyze how high stochastic uncertainty in depth estimation impacts planning within the minimum and maximum bounds. We plot the probability that planning is uncertain given that depth estimation stochastic uncertainty is high, by varying $P(DE = \text{H}|SU = \text{H}, \cdot)$ in Tables 3.1 and 3.2 between $[0, 1]$ with a step size of 0.01. The results are illustrated in Figure 3.4a.

The plot represents the influence of high stochastic uncertainty on depth estimation and the way it propagates on planning. We observe that in the component-based architecture, stochastic uncertainty in depth estimation has a lower impact on planning than in the

Table 3.2: Independent and conditional probabilities for the component-based architecture in Figure 3.3b. The acronyms are defined in Table's 3.1 caption.

$P(\cdot)$	EU_{OD}	SU_{OD}	EU_{DE}	SU_{DE}	EU_{SS}	SU_{SS}
H	0.14	0.16	0.31	0.44	0.17	0.19

$P(OD$	EU_{OD}	$SU_{OD})$
0.0	L	L
0.57	L	H
0.41	H	L
1.0	H	H

$P(DE$	EU_{DE}	$SU_{DE})$
0.0	L	L
0.51	L	H
0.47	H	L
1	H	H

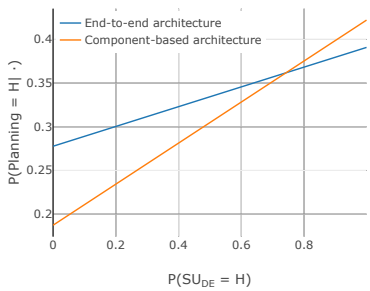
$P(SS$	EU_{SS}	$SU_{SS})$
0.0	L	L
0.11	L	H
0.42	H	L
1.0	H	H

$P(Planning$	SS	DE	$OD)$
0.0	L	L	L
0.34	L	L	H
0.34	L	H	L
0.66	L	H	H
0.34	H	L	L
0.66	H	L	H
0.66	H	H	L
1	H	H	H

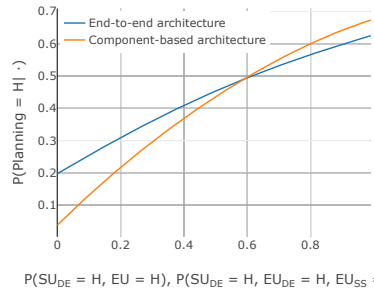
end-to-end architecture, for values up to ~ 0.7, after which the end-to-end architecture is more resilient to uncertainty. Depending on the operational environment, a software architect can choose the design that better fits the expected conditions. For example, if an autonomous vehicle operates in limited domains – e.g., inside a warehouse – where the probability of encountering stochastic events is low, the component-based architecture for the scene understanding system is more appropriate.

Using the same model, we can evaluate the influence of multiple sources of uncertainty on planning. We use the realistic assumption that the CityScape data set does not approximate all driving scenarios and thus may introduce high epistemic uncertainties. Therefore, we evaluate the influence of all epistemic uncertainty sources on planning in the scenario described above, where we assume high stochastic uncertainty in depth estimation. We use the same method as above to evaluate the probability that planning will have high uncertainty while we vary all epistemic uncertainty nodes simultaneously with the stochastic uncertainty in depth estimation. The uncertainties vary between $[0, 1]$, with a step size of 0.01. The results are plotted in Figure 3.4b.

As in the previous case, the end-to-end architecture is more resilient to high uncertainties, for all the components mentioned above. Moreover, the threshold where the end-to-end architecture becomes more resilient than the component-based architecture is lower. Nonetheless, the impact on planning remains high for values near the threshold, where both architectures behave similarly. With a 50% chance to plan actions that may lead to unintended outcomes, the system may not be usable. However, epistemic uncertainty



(a) Influence of high stochastic uncertainty in depth estimation on planning.



(b) Influence of high stochastic uncertainty for depth estimation and all epistemic uncertainties on planning.

Figure 3.4: Quantitative evaluation of uncertainty in software architecture.

can be removed using more training data, so the scenario in which epistemic uncertainty is low is more realistic. In this case, the component-based architecture is more resilient to uncertainty than the end-to-end architecture. This result implies that the component-based architecture is less likely to lead to unintended outcomes.

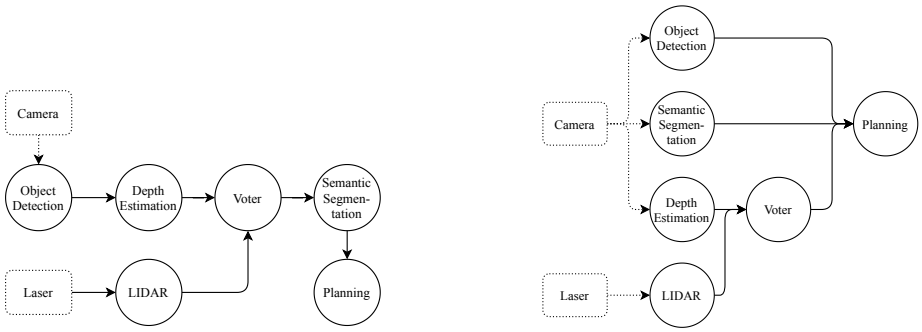
In the following section, we use the same method to assess the impact of using architectural patterns to remove uncertainty.

3.7 Using architectural patterns to mitigate uncertainty

Following the results from Section 3.6 and the changes suggested in Section 3.5, we evaluate the impact of using architectural design patterns that can decrease stochastic uncertainty in depth estimation. In particular, we explore the use of LIDAR – as suggested in [27, 251] – to complement computer vision for depth estimation. LIDAR is known to be more reliable than camera sensors and is explored by many autonomous vehicle manufacturers.

As discussed in Section 3.5, the output of depth estimation is independent for both architectures. Therefore, the impact of adding a new component is the same for both designs. Moreover, since the system we study is safety critical, the integration of LIDAR can be done using architectural patterns for safety critical systems [13]. In particular, we evaluate the use of the *n-versioning programming* pattern because it does not require any acceptance test. The development of acceptance tests for DL components is out of the scope of this thesis.

The proposed changes are illustrated in Figure 3.5. The Voter component in Figure 3.5a is added next to depth estimation, although all inference tasks for the DL components are performed at the same time. Before the depth estimation outcome is sent to planning, the outcome from the DL component is checked against the outcome from LIDAR. Similarly,



(a) End to end architecture.

(b) Component based architecture.

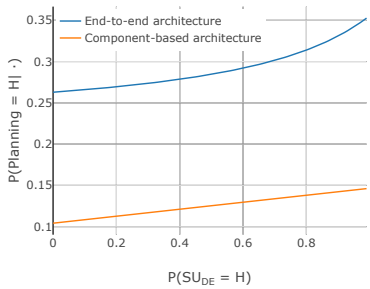
Figure 3.5: Functional architectures when using the n-version programming pattern to reduce stochastic uncertainty for depth estimation.

in the architecture from Figure 3.5b, the Voter component is added after depth estimation because all DL components are independent.

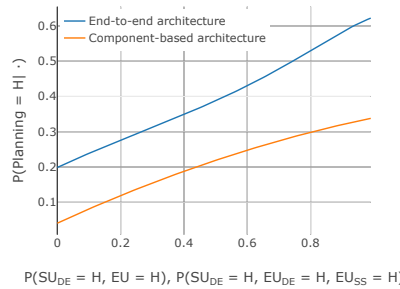
The probability tables have to be adjusted as follows. Because LIDAR has high accuracy, its uncertainty only depends on the distance we want to perform analysis for [283]. Since we do not have access to a LIDAR component, we use a proxy from literature [310] for its uncertainty values. The probability that the voter will be uncertain is a weighted average between the LIDAR and the number of times depth estimation has low error rate and low uncertainty (according to the thresholds introduced in the previous section), where the LIDAR component's contribution is of 90%. The final values are 0.09 in the first case, and 0.1 in the second case. The formalism of BNs allows prior information to be incomplete. Therefore this approximation is sufficient to perform the same computations as in the previous section and evaluate the design alternatives.

The results are presented in Figure 3.6a for the end to end architecture, and Figure 3.6b for the component based architecture. We can see that, in both cases, the component based architecture is significantly more resilient to uncertainty, even for high values of epistemic uncertainty (Figure 3.6b). A striking difference between the two designs is that the end to end architecture has a slight exponential curve for high levels of uncertainty, while the component based architecture has a linear and slightly logarithmic curve for high uncertainty, which shows the former increases faster.

When comparing the results before and after applying the n-version programming pattern (Figure 3.6 vs Figure 3.4), we observe that applying the pattern is successful in decreasing the uncertainty influence on planning. In particular for the component based architecture, where the decrease is more significant than for the end to end architecture.



(a) Influence of stochastic uncertainty for depth estimation on planning.



(b) Influence of stochastic uncertainty for depth estimation and all epistemic uncertainties on planning.

Figure 3.6: Quantitative evaluation of uncertainty in software architecture design using the n-version programming pattern.

3.8 Conclusions and Future Research

We introduce Modeling Uncertainty During Design (MUDD), a method to evaluate and compare architecture design alternatives for software systems that use ML components. In particular, we propose to explicitly model the inherent uncertainty specific to ML components at design time and evaluate how it propagates and influences other components in a system. The proposed information needed to quantify the uncertainty for each ML component is well studied both in the software architecture and in the ML literature. For modeling systems with both traditional and ML components, we use Bayesian networks (BNs), which allow to evaluate software architectures both qualitatively and quantitatively.

We validate MUDD using as example a perception system for autonomous vehicles. The system consists of 3 components which can only be implemented with deep learning algorithms. We bring empirical evidence that a component-based architecture is significantly more resilient to uncertainty than an end-to-end design. Moreover, we show that software architecture design patterns can be successfully used to decrease the uncertainty of a system using ML components. Nonetheless, the system studied can not be used to exhibit all strengths and possible scenarios in which MUDD can be used.

BNs are directed graphs and do not allow loops. Therefore, two components that exchange data between themselves can not be modeled with this method. For future research we propose to explore modeling alternatives that can overcome this limitation. Hybrid models, such as Markov random fields or factor graphs use both directed and un-directed edges and are promising alternatives.

The evaluation presented in Section 3.7 also suggests new research directions: it is interesting to validate which architectural patterns are more suitable to reduce uncertainty for ML components, and in which contexts. Following this research direction may lead to new architectural models and patterns that better support the integration between

traditional and ML software components.

Moreover, there are many common elements between tackling uncertainty at design time and at run-time. Several links and an integration between the two is compelling to explore in future research. For example, the models introduced in this paper can be integrated with run-time models in order to evaluate the uncertainty of decision pipelines using real-time data.

4

Adoption and Effects of Engineering Best Practices in Machine Learning

In this chapter we aim to empirically determine the state of the art in how teams develop, deploy and maintain software with ML components. Towards this goal, we mined both academic and grey literature, and identified 29 engineering best practices for ML applications. We conducted a survey among 313 practitioners to determine the degree of adoption for these practices, and to validate their perceived effects. Using the survey responses, we quantified practice adoption, differentiated along demographic characteristics, such as geography or team size. We also tested correlations and investigated linear and non-linear relationships between practices and their perceived effects. Our findings indicate, for example, that larger teams tend to adopt more practices, and that traditional software engineering practices tend to have lower adoption than ML specific practices. The statistical models can also predict perceived effects such as agility, software quality and traceability, from the degree of adoption for specific sets of practices. Combining practice adoption rates with practice importance, as revealed by statistical models, we identify practices that are important but have low adoption, as well as practices that are widely adopted but are less important for the effects we studied. Overall, the survey and the response analysis provide a quantitative basis for assessment and step-wise improvement of practice adoption by ML teams.

This chapter has been published as *A. Serban, K. van der Blom, H. Hoos, J. Visser, Adoption and Effects of Software Engineering Best Practices in Machine Learning, Empirical Software Engineering and Measurement, 2020* [245].

4.1 Introduction

The adoption of ML components in production-ready applications demands strong engineering methods to ensure robust development, deployment and maintenance. While a wide body of academic literature acknowledges these challenges [14, 117, 131, 160, 282], there is little academic literature to guide practitioners. In fact, a large part of the literature concerning engineering practices for ML applications can be classified as grey literature [82] and consists of blog articles, presentation slides or white papers.

In this chapter, we aim to determine the state of the art in how teams develop, deploy and maintain software solutions that involve ML components. Towards this goal, we have first distilled a set of 29 engineering best practices from the academic and grey literature. These practices can be classified as *traditional* practices, which apply to any software application, *modified* practices, which were adapted from traditional practices to suit the needs of ML applications, and completely *new* practices, designed for ML applications.

In order to validate the adoption and relevance of the practices we ran a survey among ML practitioners, with a focus on teams developing software with ML components. The survey was designed to measure the adoption of practices and also to assess the effects of adopting specific sets of practices. We obtained 313 valid responses, and analysed 29 practices and their influence on 4 effects.

The main contributions of this chapter are as follows. Firstly, we summarise academic and grey literature in a collection of best practices. This body of information can be used by practitioners to improve their development process and serves as a gateway to literature on this topic. Secondly, we determine the state of the art by measuring the adoption of the practices. These results are used to rank the practices by adoption level and can serve to assess the popularity of particular practices. Thirdly, we investigate the relationship between groups of practices and their intended effects, through different lenses – by training a linear regression model to check if the intended effect is dependent on the practices and by training more sophisticated regression models, using a variety of ML approaches to predict the effects from the practices. Lastly, we investigate the adoption of practices based on the data type being processed and based on the practice categories introduced above (traditional, modified, new).

Our results suggest that the practices apply universally to any ML application, and are largely independent of the type of data considered. Moreover, we found a strong dependency between groups of practices and their intended effect. Using the contribution of each practice to the desired effect (extracted from our predictive models) and their adoption rate, we outline a method for prioritising practice improvements tailored for achieving specific effects, such as increased traceability or software quality. While our study is restricted to ML, rather than the broader and less clearly delineated field of artificial intelligence (AI), many of our findings may have wider applications, as we will briefly discuss in Section 4.8.

This chapter is organised as follows. We first discuss background and related work (Section 4.2). Next, we describe the process and results of mining practices from literature (Section 4.3). A description of the design of our study (Section 4.4) is followed by a presentation of the survey results regarding the adoption of practices (Section 4.5) and a deeper analysis of the relationship between the practices and their effects (Section 4.6).

Finally, we discuss interpretation and limitations of our findings (Section 4.7) and close with general conclusions and remarks on future work (Section 4.8).

4.2 Background and related work

Engineering challenges posed by ML. As ML components are developed and deployed, several engineering challenges specific to the ML software development life-cycle emerge [14, 117, 131, 160, 282]. Arpteg et al. [14] identified a set of 12 challenges that target development, deployment and organisational issues. In particular, managing and versioning data during development, monitoring and logging data for deployed models and estimating the effort needed to develop ML components present striking differences with the development of traditional software components.

Similarly, Ishikawa and Yoshioka [117] as well as Wan et al. [282] have studied how software engineers perceive the challenges related to ML and how ML changes the traditional software development life-cycle. Both studies ran user surveys with a majority of respondents from Asia. We could not find a similar study without this regional bias. Nonetheless, both publications concluded that testing and ensuring the quality of ML components is particularly difficult, because a test oracle is missing, the components often behave nondeterministically, and test coverage is hard to define. To classify the challenges raised by ML components, Lwakatare et al. introduced a comprehensive taxonomy [160].

White and grey literature analysis. In search for ways to meet the challenges presented earlier, we mined the literature and collected SE best practices for ML. We observed that the majority of literature on this topic consists of so called grey literature [82] – i.e., blog articles, presentation slides or white papers from commercial companies – while there is relatively little academic literature. Garousi et al. [82] showed that, if used properly, grey literature can benefit SE research, providing valuable additional information. However, this literature must be used with care, because it does not contain strong empirical evidence to support its claims [83]. We decided to included the grey literature in our literature search, using the process described by Garousi et al. [82], because: (1) coverage of the subject by academic literature is rather incomplete, (2) contextual information is important for the subject of study – i.e., practitioners may have different opinions than scientists on what qualifies as best practices – and (3) grey literature may corroborate scientific outcomes with practical experience.

Related work. We focus on peer-reviewed related work that proposes, collects or validates engineering best practices for ML. One of the initial publications on this topic is the work of Sculley et al. [236], which used the framework of technical debt to explore risk factors for ML components. In particular, they argued that ML components have a stronger propensity to incur technical debt, because they have all maintenance problems specific to traditional software as well as a set of additional issues specific to ML. They also presented a set of anti-patterns and practices aimed at avoiding technical debt in systems using ML components. Compared to [236], we introduce a broader set of practices, applicable to more effects than technical debt. Nonetheless, some of their engineering specific suggestions are included in our catalogue of practices.

Breck et al. [36] introduced 28 tests and monitoring practices that target different stages of the development process for ML. They also proposed a list of benefits resulting

from implementing the tests and developed a model to score test practice adoption, aimed at measuring technical debt. Again, the practices dedicated to SE from [36] have been included in our catalogue. On the same topic, Zhang et al. introduced a survey on testing techniques for ML components [311], which – in contrast to the broader approach taken in [36] – only targets testing ML models.

To identify challenges faced by small companies in developing ML applications, de Souza Nascimento et al. ran interviews with 7 developers [63]. Afterwards, they proposed and validated a set of checklists to help developers overcome the challenges faced. Although the validation session is not thorough (it only included a focus group with 2 participants), some of the items in the checklists qualify as best practice candidates. Some practices are included in our catalogue and our survey further confirms their relevance and adoption.

Washizaki et al. [286] studied and classified software architecture design patterns and anti-patterns for ML, extracted from white and grey literature. Many of these patterns are application and context specific, i.e., they depend on the architectural style or on the type of data used. The patterns are of a general character and the ones similar to recommendations we found in literature were included in our catalogue of practices.

Amershi et al. conducted a study internally at Microsoft, aimed at collecting challenges and best practices for SE used by various teams in the organisation [8]. They reported on a broad range of challenges and practices used at different stages of the software development life cycle. Using the experience of the respondents and the set of challenges, they built a maturity model to assess each team. However, the set of challenges and reported practices are broad and often not actionable. Moreover, they represent the opinions of team members from Microsoft, where typically more resources are dedicated to ensuring adoption of best practices than within smaller companies. In our work, we aim to bridge this gap by running a survey with practitioners with various backgrounds and by presenting a set of actionable, fine-grained best practices.

4.3 Mining practices from literature

Document Search. In addition to the publications discussed in Section 4.2, we searched the academic and grey literature on the topic of SE best practices for ML applications. We used both Google and Google Scholar, for which we compiled a common set of queries. The keywords used for querying suggest different steps in the development cycle, e.g., development, deployment, operations, etc. For each query, we also developed two variants, by (1) replacing the term ‘machine learning’ with ‘deep learning’ whenever possible, and (2) removing stop words and composing a Boolean AND query from the remaining key words. As an example of the second variant, consider the query “software engineering” AND “machine learning”, stemming from the query “software engineering for machine learning”. All queries were submitted to Google and Google Scholar, and the first 5 result pages were manually inspected.

A total of 64 queries, including variants, were used, and 43 of the resulting articles were selected for initial inspection. In order to avoid search engine personalisation, all queries were sent from a public network, with an extension that removes browser cookies.

Document classification. Based on criteria formulated in [82], such as authoritativeness of the outlet, as well as objectivity of the style and content, we excluded low-quality

Table 4.1: Successful search queries. The table shows the base queries, for which any variant (described in text) led to a valid source and at least one practice.

Query	Documents
software engineering for machine learning	[8]
data labeling best practices	[6, 56, 210, 219]
machine learning engineering practices	[37, 230, 313]
software development machine learning	[124]
machine learning production	[223, 253]
machine learning production practices	[5, 24, 146, 171]
machine learning deployment	[65]
machine learning deployment practices	[229]
machine learning pipelines practices	[114]
machine learning operations	[268]
machine learning versioning	[279]
machine learning versioning practices	[104]

documents and classified the remaining documents as either academic literature or grey literature. Moreover, we filtered for duplicates, because chunks of information were sometimes reused in grey literature.

After classifying and filtering the results, we identified 21 relevant documents, including scientific articles, white papers, blogs and presentation slides, that – along with the publications introduced in Section 4.2 – were used to mine SE best practices for ML. Other relevant sources were selected through a snowball strategy, by following references and pointers from the initial articles.

Table 4.1 lists the successful search terms (without variants), from which at least one document passed the final selection. Whenever the queries had common results, we only considered relevant the first query. The second column shows the documents selected from the base queries and their variants.

Extracting a common taxonomy for the practices. Many of the selected documents provide, or implicitly presume, a grouping of practices based on development activities specific to ML. For example, Amershi et al. [8] present a nine-stage ML pipeline. Alternatively, Sato et al. [230] partition similar activities into six pipeline stages. All processes have roots in early models for data mining, such as CRISP-DM [294]. While no single partitioning of ML activities emerged as most authoritative, we were able to reconstruct a broad taxonomy that is compatible with all partitionings found in the literature. We will use this categorisation to group ML development practices and to structure our survey and subsequent discussion of our findings:

- Data - Practices that come before training, including collecting and preparing data for training ML models.

- Training - Practices related to planning, developing and running training experiments.

- Deployment - Practices related to preparing a model for deployment, deploying, monitoring and maintaining an ML model in production.
- Coding - Practices for writing, testing, and deploying code.
- Team - Practices related to communication and alignment in a software development team.
- Governance - Practices that relate to ensuring responsible use of ML, including accountability regarding privacy, transparency, and usage of human, financial, or energy resources.

Compiling a catalogue of practices. From the selected documents we compiled an initial set of practices using the following methodology. First, we identified all practices, tests or recommendations that had similar goals. In some articles, the recommendations only suggest the final goal – e.g., ensure that trained ML models can be traced back to the data and training scripts used – without providing details on the steps needed to achieve it. In other publications, the recommendations provided detailed steps used to achieve the goals – e.g., use versioning for data, models, configurations and training scripts [171, 253, 279]. In this example, traceability is an outcome of correctly versioning all artefacts used in training. Whenever we encountered similar scenarios, we selected or abstracted actionable practices and added the high-level goals to a special group, which we call "Effects" and describe in Table 4.6.

Next, we assessed the resulting practices and selected those specifically related to engineering or to the organisation of engineering processes. This initial selection gave us 23 practices, which naturally fall into 4 out of the 6 classes introduced above. While this set of practices reflected the ML development process, it lacked practices from traditional SE. Given that practitioners with a strong background in ML might be unaware of the developments in SE, in a third stage, we complemented the initial set of practices with 6 practices from traditional SE – three of a strictly technical nature, falling into the "Coding" class, and three relating to social aspects, falling into the "Team" class. We selected these practices because we consider them challenging, yet essential in software development.

The resulting 29 practices are listed in Table 4.8 and the effects in Table 4.6. The practices are available to practitioners in a more elaborate format in an online catalogue[1], consisting of detailed descriptions and concise statements of intent, motivation, related practices, references and an indication of difficulty. A curated reading list with these references, further relevant literature as well as a selection of supporting tools is maintained online[2].

4.4 Study design

We validated the set of practices with both researchers and practitioners through a survey. For this, we designed a descriptive questionnaire asking respondents if the *team* they are part of adopts, in their ML projects, the practices we identified earlier. Before distributing

[1] https://se-ml.github.io/practices/
[2] https://github.com/SE-ML/awesome-seml

the survey, we interviewed five practitioners with diverse backgrounds, in order to check if any information from the survey was redundant or whether any practices were missing.

Questionnaire. In designing the questionnaire used in our survey, we followed the recommendations of Kitchenham et al. [136] and Ciolkowski et al. [55]. We designed a cross-sectional observational study [136], i.e., participants were asked at the moment of filling the questionnaire if they adopted the recommended practices. Several preliminary questions were designed to specifically assign participants to groups. This renders the study a concurrent control study, in which participants are not randomly grouped [136].

The target audience were *teams* of practitioners using ML components in a project. Specific preliminary questions were added to allow filtering between teams that build and deploy ML applications, use ML and do not build an application or do not use ML at all. We consider that a significant amount of engineering is also needed in research (where ML may be used without building deployed applications), especially in running large-scale deep learning experiments, and would like to verify which practices are relevant in this context. Team profile (e.g., tech company, governmental organisation), team size (e.g., 1 person, 6-9 persons), team experience (e.g.„ most of us have 1-2 years of experience), and the types of data used (e.g., tabular data, images) were also included in the preliminaries. In total, the preliminaries contained 5 questions that were later used to group participants and filter out noise.

Then, 31 questions followed, with standard answers, mapped onto the practices from Table 4.8. In two cases, multiple questions map onto the same practice; for example, continuous integration is achieved by automating the build process and running it at each commit. In the questionnaire, we asked two questions, one for each action, although the answer was compiled to one practice.

We used standard answers, on a Likert scale with four possible answers, in order to avoid the middle null-point strategy of answering [108]. The labels were chosen in order to reflect the degree of adoption, rather than the level of agreement [220]. This allowed the practices to be expressed impartially – e.g., "our software build process is fully automated" – and the answers to express degrees of adoption – e.g., "not at all" or "completely" – instead of degrees of agreement such as "agree" or "strongly agree". This strategy eliminated confusing questions and answers, which may lead to an extreme null-point bias [108]. Whenever the answer scale did not match the full range of answers, we added specific answers which helped to avoid noisy results; for example, in the questions about data labelling, we added the answer "we do not use data labels", which accounts for unsupervised learning scenarios.

The questionnaire ended with a section on the perceived effects of adopting the practices. This enabled us to test the hypothesis that adopting a group of practices helps to achieve an effect. The four questions on perceived effects are shown in Table 4.6.

Although the questionnaire has 45 questions, we employ optimisation techniques, such as automatically moving to the next question once an answer is selected, to reduce the time required for completing our questionnaire to 7 minutes on average.

Pilot interviews. Before distributing the survey broadly, we invited five practitioners with distinct backgrounds – ranging from tech startups to large tech companies – to an interview. We asked them to answer a set of questions regarding the challenges they face and the most important engineering practices they adopt. All interviewees were also asked

Table 4.2: Profiles of the pilot interview subjects.

Id	Company Profile	Team Size	Experience
P1	Tech Startup	5-6 ppl.	1-2 years
P2	Tech company	10-15 ppl.	>5 years
P3	Research lab	5-6 ppl.	2-5 years
P4	Tech Startup	10-15 ppl.	2-5 years
P5	Non-tech company	6-9 ppl.	1-2 years

to fill out and comment on the questionnaire. Since the survey is focused on teams, in Table 4.2 we present the team and company profiles for each interviewee; all interviewees use ML for a project. Moreover, P4 is part of a team that builds platforms to support the ML process and uses distinct ML projects to test the platforms.

The biggest challenges faced by the interviewees were: ensuring data quality and data documentation (P5), data versioning and freshness (P2), scalability (P1, P4) and communication with other departments inside the company (P5). For each challenge mentioned, there is at least one practice addressing it in Table 4.8. The most important engineering practices mentioned were: using version control for data and models (P1, P4), continuous deployment (P2, P5) and model maintenance (P2). Several practices to address these challenges were already listed in Table 4.8.

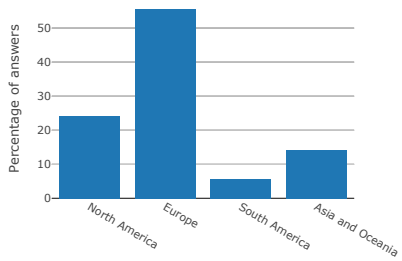
After completing the questionnaire, all interviewees agreed with the relevance of all the practices we listed and did not consider any of them redundant. The interviewees suggested that some questions needed additional allowable answers, to cover the range of possible responses and to avoid bias. For example, for a question about the labelling process, we added "we do not use labels" to avoid forcing users of unsupervised learning to resort to "not at all".

We used the feedback from the interviews to refine the questionnaire, adding answers to four questions and rewording others.

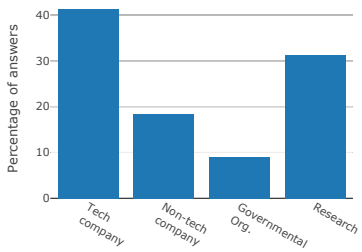
Distribution. After the pilot interviews, our survey was distributed using a snowball strategy. Initially, we reached out to our network of contacts and to the authors of the publications used to extract the practices, asking them to distribute the survey through their networks. Moreover, we openly advertised the survey through channels commonly used by practitioners, such as Twitter, Medium, HackerNoon, Dev.to and the Meetup groups for ML in several cities.

4.5 Findings on practice adoption

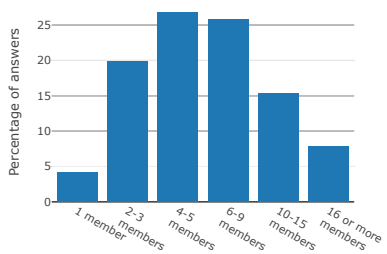
In total, we obtained 350 valid responses to our survey, after filtering out incomplete answers or respondents that spent too little time to have given serious answers (under 2 minutes). From this initial set, we discarded 12 answers from respondents who were not part of a team using ML. Moreover, we applied fine-grained filtering, using the percentage of questions that were answered in the prerequisites (at least 50 %) and in the practice adoption questions (at least 50 %), resulting in *313 complete responses*. Whenever not mentioned otherwise, the analysis will be based on these responses.



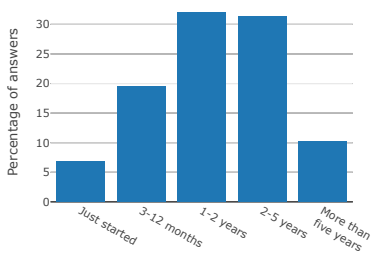
(a) Respondents grouped by regions.



(b) Respondents grouped by organisation type.



(c) Respondents grouped by team size.



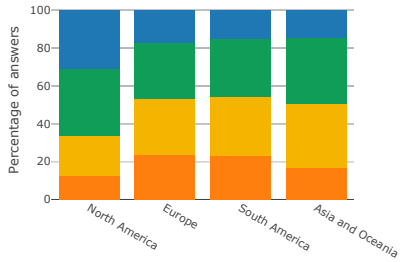
(d) Respondents grouped by team experience.

Figure 4.1: Demographic information describing the survey participants. All plots show the percentage of respondents, grouped by demographic factors.

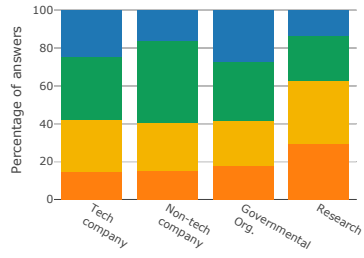
Demographics. Using the initial preliminary questions, we provide a demographic characterisation of the respondents in Figure 4.1. First, we grouped the answers using the location attributes and present the results in Figure 4.1a. We observe that Europe has an overall higher contribution, although other regions are also well represented. This possible bias will be discussed later in this section, when analysing the answers for each region.

Figure 4.1b illustrates the percentage of respondents grouped by the organisation type. The higher percentages are for teams working in tech companies (e.g., social media platforms, semiconductors) and research labs. These results are not surprising, since both research and adoption of ML is driven by these two classes of practitioners. Nonetheless, non-tech companies (e.g., banks) and governmental organisations are also well represented.

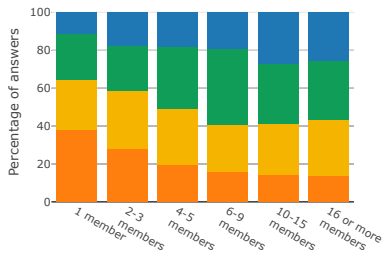
In the last two plots, we show the percentage of answers grouped by team size – Figure 4.1c – and team experience – Figure 4.1d. We observe that most teams have between 4-5 and 6-9 members, corresponding to normal software development teams (as recommended, for example, in Agile development). Similarly, most teams have between 1-2 years and



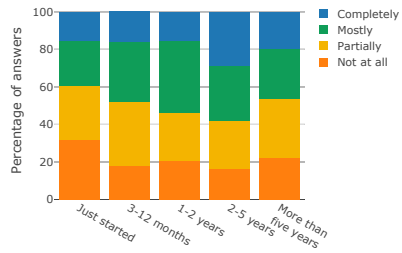
(a) Adoption of practices grouped by regions.



(b) Adoption of practices grouped by org. type.



(c) Adoption of practices grouped by team size.



(d) Adoption of practices grouped by team experience.

Figure 4.2: Adoption of practices grouped by various demographic factors. All plots show the percentage of answers, grouped by the answer types illustrated in the plot legend.

2-5 years of experience, which is an anticipated result, since these intervals correspond to the recent growth in popularity of ML among practitioners. Overall, the demographic information indicates that our data is well balanced and diverse.

Next, we analysed the adoption of practices grouped by the demographic factors introduced earlier. We display the answers from the practice questions in Figure 4.2, grouped and normalised using the Likert scale used in the survey. Figure 4.2a shows the percentage of answers grouped by regions. As discussed earlier, Europe is somewhat over-represented in our data set. However, the adoption of practices for Europe does not present striking differences when compared to South America or Asia. Conversely, the respondents from North America have a significant higher number of adopted practices (corresponding to answers “Completely” or “Mostly”) than other regions. Since this region is well represented in our set of responses, it is likely that practitioners from North America have a higher adoption of practices. Moreover, since Europe does not present striking differences with other regions, it is likely that little bias is introduced by its over-representation.

Figure 4.2b shows the adoption of practices grouped by the organisation type. We observe that tech companies have a higher rate of complete adoption than others. Research organisations tend to have lower practice adoption. This could reflect that they are aware of the practices, but only develop prototypes, for which adoption is not needed, or partial adoption is sufficient. In fact, for non-deployment practices only, adoption rates are similar.

For team size – Figure 4.2c – we observe a trend towards higher adoption of practices (and also lower percentage of practices that were not adopted at all) as team size increases. This could be caused by better task distribution among team members, or it could be a result of larger teams including members with different backgrounds. Similarly, for team experience, there is a trend towards higher adoption of practices as the team experience increases, as seen in Figure 4.2d. These results were anticipated, since more experience or a deeper involvement in technology exposes team members to the knowledge needed to adopt best practices. A contrasting trend can be observed only for teams with more than 5 years of experience, where the percentage of practices that are only partially or not at all adopted increases slightly. This result may reveal that practitioners who started very early may be unaware of practices that are developed recently.

The results presented above confirm our questions were clear and achieved their goals, and that the answer scale did not introduce bias.

Practice adoption ranking. We now explore the adoption of practices, based on the practice types discussed in Section 4.1. In particular, we are interested in finding out whether traditional SE practices are equally adopted in ML engineering and which new or modified practices are popular among practitioners. Moreover, we also comment on the least and most adopted practices.

The practices are classified as follows: (1) *new* practices, designed specifically for the ML process, (2) *modified* practices, derived from traditional SE practices, but adapted for the ML process and (3) *traditional* practices, applied equally in traditional SE and ML. This classification is illustrated in the "Type" column of Table 4.8.

In order to measure the adoption rate of the practices, we devised a ranking algorithm with the following steps:

1. Compute for each practice the percentage h of respondents with *at least high* adoption (counting "Completely" answers), the percentage m with *at least medium* adoption (counting "Completely" and "Mostly"), and the percentage l with *at least low* adoption (counting "Completely", "Mostly", and "Partially"). As an example, for practice 1 we obtained $h = 9.62\%$, $m = 34.04\%$, and $l = 65.92\%$.

2. Convert each percentage into a rank number. For practice 1, we obtained $r_h = 22$, $r_m = 23$, and $r_l = 19$.

3. Take the average of the three ranks for each practice and then rank the practices according to this average. For practice 1, rank 22 was obtained, as can be seen in Table 4.8.

Thus, the final rank is the average of: the practice rank on *at least high adoption*, its rank on *at least medium adoption*, and its rank on *at least low adoption*. By accumulating the answers in step 1, we expect to cancel out the noise stemming from fuzzy boundaries between subsequent answer types.

Table 4.3: Adoption of practices based on the practice type.

Practice Type	At least high adoption	At least medium adoption	At least low adoption
Traditional	15.6%	47.8%	76.8%
Modified	11.3%	42.0%	76.9%
New	**16.9**%	**50.0**%	**83.9%**

The results are presented in the supplementary materials and the online version of the publication [245]. We note that the most adopted practices (practices 6, 7) are related to establishing and communicating clear objectives and metrics for training. Versioning (practice 16), continuous monitoring of the model during experimentation (practice 14) and writing reusable scripts for data management (practice 3) complete the top 5 positions. It is interesting to note that the most adopted practices are either new or modified practices, and not traditional SE practices.

At the other end of the spectrum, we note that the least adopted practices (practices 9, 10) are related to feature management. Writing tests (practice 17), automated hyper-parameter optimisation (practice 13) and shadow deployment (practice 22) complete the 5 least adopted practices. In general, the least adopted practices require more effort and knowledge. Some practices, related to testing (practices 8, 17) or documentation (practice 9) are also known to raise issues in traditional SE. Moreover, shadow deployment (practice 22) and AutoML (practice 13) require advanced infrastructure.

In order to compare the adoption of practices grouped by their type, we averaged the three percentages described earlier (without transforming them into ranks), for each practice type. The results are presented in Table 4.3. We observe that the most adopted practices are new practices, specifically designed for ML, followed by traditional and modified practices. Traditional practices in the "Team" category are ranked highly, since collaborative development platforms have become common tools among practitioners and offer good support for information sharing and coordination inside a team. In contrast, traditional practices related to code quality, such as running regression tests (practice 17) or using static analysis tools to check code quality (practice 19), have low adoption.

Influence of data type on practice adoption. The practices presented in Table 4.8 are general and should apply to any context. However, the type of data being processed influences the choice of algorithms and might also influence the adoption of practices. For example, when processing images or text, it is common to rely on deep neural networks (DNNs), where training is not preceded by a feature extraction step. Conversely, for other types of ML algorithms, a feature extraction step is common. Here, we investigate the influence of the type of data to be processed on the adoption of practices. Moreover, we explore the practices with distinct adoption rates per data type.

The percentage of respondents per data type and the corresponding overall practice adoption rates are presented in Table 4.4. We employ the same percentages described earlier to assess the practice adoption rates per data type. We observe that, in our data set, tabular data, text, images and videos are predominant (each above 25%) and have very similar adoption rates. Audio and time series have lower representation (under 8%), making

Table 4.4: Adoption of practices based on the data type.

Data Type	Perc. of respondents	Adoption		
		At least high	At least medium	At least low
Tabular Data	31.7%	18.0%	50.1%	70.2%
Text	29.7%	19.3%	52.6%	71.4%
Images, Videos	26.4%	19.3%	50.5%	71.5%
Audio	8.8%	24.42%	55.8%	72.6%
Time Series	2.6%	28.2%	60.3%	72.6%
Graphs	0.5%	-	-	-

their adoption rates less reliable. Still, apart from the "At least high" category, adoption rates remain similar. The "Graphs" data type is used rarely (0.5%), making adoption rates too unreliable to report.

When comparing the adoption of individual practices, grouped by data type, we observed that several practices tend to have higher adoption for particular data types. For all comparisons, we used the "at least high" adoption rate. First, practice 13, on automatic hyper-parameter optimisation, has an adoption rate that is more than 8% higher for tabular data than for text or images. This result could be due to the the algorithms or tools used. The tool support for automatic hyper-parameter optimisation in more traditional ML methods, such as random forests or SVMs – which are popular for tabular data – is more mature than for newer techniques, e.g., DNNs. Second, practice 29, on enforcing privacy and fairness, has an adoption rate for tabular data that is more than 10% higher than that for text or images. Last, practice 12, on the capacity to run training experiments in parallel, has adoption rates for text and images that are over 10% higher than that for tabular data. Perhaps the infrastructure needed to run experiments with text or images, where DNNs are used extensively and parallelisation is required to achieve good results, facilitates the practice adoption.

4.6 Analysis of practices and effects

Following the practice adoption questions, in the questionnaire there were four questions about the perceived effects of adopting these practices. These questions were designed to test the hypothesis that adopting a set of practices will lead to a desired effect. A mapping between practices and effects, as initially hypothesised, can be found in Table 4.6.

Correlations among practices. First, we report results from an analysis of the correlation between practices. We employ the Spearman rank correlation coefficient, ρ, in light of the ordinal nature of the Likert scale used in our questionnaire. To determine the statistical significance of the observed correlations, we perform t-tests with a significance level of 0.01.

In total we found 244 statistically significant, moderate to strong correlations ($\rho \geq 0.35$), of which we report on the most informative ones. For example, writing reusable scripts

Table 4.5: Description of the effects studied.

Effects	Description
Agility	The team can quickly experiment with new data and algorithms, and quickly assess and deploy new models
Software Quality	The software produced is of high quality (technical and functional)
Team Effectiveness	Experts with different skill sets (e.g., data science, software development, operations) collaborate efficiently
Traceability	Outcomes of production models can easily be traced back to model configuration and input data

for data management (practice 3) correlates positively with testing for skews between different models (practice 23, $\rho = 0.35$). This suggests that the ability to reuse code for data transformation can facilitate model evaluation. Furthermore, sharing the training objectives within the team (practice 6) correlates positively with using a shared backlog (practice 27, $\rho = 0.38$) and using relevant metrics to measure the training objective (practice 7, $\rho = 0.43$). Testing the feature extraction code (practice 8) correlates positively with practices 9 ($\rho = 0.35$) and 10 ($\rho = 0.56$), on feature documentation and management. This indicates that practitioners tend to use advanced feature management methods concomitantly and that the feature management practices complement each other. As expected, practice 8 correlates positively with practice 17, on running regression tests ($\rho = 0.37$).

Performing peer review on training scripts (practice 11) correlates positively with all team practices – using collaborative development platforms (practice 26, $\rho = 0.40$), working against a backlog (practice 27, $\rho = 0.44$) and good team communication (practice 28, $\rho = 0.44$). This result is in line with our expectations, since collaborative platforms provide features for code review, and this is further enhanced by good communication within the team. Peer review also correlates positively with using static analysis tools for code quality (practice 19, $\rho = 0.48$), which suggests that teams prioritising code quality apply various techniques for this purpose.

The practices for deployment correlate positively between themselves, suggesting that teams with advanced deployment infrastructures tend to adopt all practices. For example, automated model deployment (practice 20) correlates positively with shadow deployment (practice 22, $\rho = 0.48$) and automated roll backs (practice 24, $\rho = 0.51$). Moreover, continuous monitoring of deployed models (practice 21) correlates positively with logging predictions in production (practice 25, $\rho = 0.51$). These results indicate that the deployment practices are complementary and that adopting some enables the adoption of others.

Linear relationship between practices and effects. Second, we used the initial mapping from practices to effects (presented in Table 4.6) to investigate the hypothesis that adopting a set of practices leads to each desired effect. For the analysis, we trained four simple, linear regression models, one for each set of practices and effects in Table 4.6. The effects description is presented in Table 4.5. For each model, we used the F-test for linear regression to test the null hypothesis that none of the practices are significant in determining the effect, with a significance level of 0.01. Since some of the data sets were

Table 4.6: Linear regression models describing the dependence of effects on the practices that were initially hypothesised to influence them. For each effect, we report the p-value from the F-test for regression and the R^2 coefficient of determination.

Effects	Practices	p-value	R^2
Agility	12, 18, 22, 24, 28	$7 \cdot 10^{-4}$	0.84
Software Quality	9, 10, 11, 17, 18, 19	$5 \cdot 10^{-3}$	0.95
Team Effectiveness	6, 26, 27, 28	$1 \cdot 10^{-5}$	0.98
Traceability	3, 5, 16, 25, 27	$4 \cdot 10^{-6}$	0.75

Table 4.7: Mean squared error (MSE), R^2 and Spearman correlation (ρ) between the predicted and the true outcomes for Random Forest Regression trained to predict the effects from the practices in the second column. The results are extracted from a test data set consisting of 25% of the data.

Effects	Practices	MSE / R^2 / ρ RF Grid Search
Agility	12, 18, 20, 21, 22, 28	0.25 / 0.80 / 0.92
Software Quality	9, 10, 11, 17, 18, 19	0.17 / 0.87 / 0.91
Team Effectiveness	6, 26, 27, 28	0.19 / 0.84 / 0.92
Traceability	3, 5, 16, 21, 25, 27	**0.21 / 0.83 / 0.93**

imbalanced, i.e., contained substantially more examples for the positive or negative effect, we applied random under-sampling to balance those sets.

The null hypothesis was rejected for all effects; the respective p-values from the F-test are shown in Table 4.6. We also performed t-tests to assess whether any of the coefficients in the regression models were statistically significantly different from zero, and found evidence that (at significance level 0.01) this was the case. For example, the t-value of practice 25 for traceability is 6.29. Moreover, the R^2 values, also shown in the table, are high for all effects, which indicates that the observed effects are rather well described by a linear model of the degree of adoption of the associated practices.

Non-linear relationship between practices and effects. Last, we report the results from training statistical models to predict each perceived effect from sets of practices. Unlike the linear regression models described earlier, here, we additionally considered ML models that do not assume a linear relationship between the practices and effects. Moreover, in order to strengthen the evaluation, we performed hold-out testing, using a test set of 25% of the data for each effect, which was only used for the final assessment of our models. We also revised the sets of practices associated with two of the effects (agility and traceability), in order to enhance the prediction accuracy of the models as assessed on validation data.

For evaluation, we consider a random forest (RF) regression model whose hyper-parameters were optimised using grid search. During training, we used 5-fold cross-validation on the training data (i.e., the 75% of the data retained after setting aside the test sets). For all experiments, we used under-sampling on the training data to remove class imbalance. We also experimented with the SMOTE over-sampling algorithm for

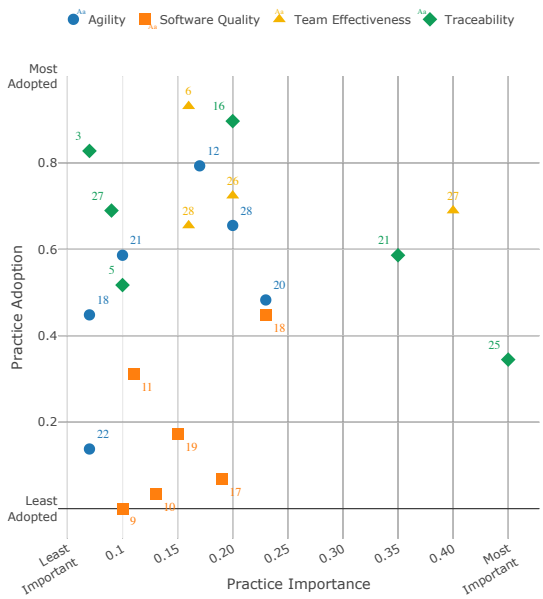


Figure 4.3: Practice adoption and importance, for each effect and practice. The practice importance is the Shapley value extracted from the grid search RF models in Table 4.7, using the test data set.

regression [50, 273], but did not observe significant increases in performance of our models. For grid search used for hyper-parameter optimisation of our RF models, we used 384 candidate configurations for each of the five folds.

The performance of the predictive model on test data is shown in Table 4.7. For all effects, we used three standard evaluation metrics: mean squared error (MSE), the R^2 coefficient of determination, and the Spearman correlation coefficient (ρ) for predicted *vs* true outcomes. We observe that, in all scenarios, the effects can be predicted from the practices with very low error and a high coefficient of determination.

Importance of practices. We also studied the contribution of each practice to the final effect, in order to determine the practices that are the most important for each effect. Towards this end, we used a well-known concept from cooperative game theory called the *Shapley value* to quantify the contributions of individual practices [78, 159]. In our case, the Shapley value intuitively reflects the increase in predictive accuracy caused by a single practice, averaged over all possible subsets of practices already considered in a given model. In order to maintain consistency across all effects, and because the models obtained from AutoML are ensembles that are more difficult to analyse, we performed All Shapley value computations are performed on the model from Table 4.7. We have computed Shapley values on training and test data, and obtained consistent results for all effects.

In order to showcase the importance of each practice for an effect, we contrast it with the adoption ranking of the practices from Section 4.5. We plot the Shapley values and the normalised ranks in Figure 4.3. The plot indicates, given our data, which practices are most important for achieving a desired effect. We observe that some very important practices

have low adoption rates, while some less important practices have high adoption rates. For example, practice 25 is very important for “Traceability", yet relatively weakly adopted. We expect that the results from this type of analysis can, in the future, provide useful guidance for practitioners in terms of aiding them to assess their rate of adoption for each practice and to create roadmaps for improving their processes. We note that our analysis currently does not take into account functional dependencies between the practices.

4.7 Discussion

We now comment on the relation between practice adoption and the challenges from Section 4.2, and discuss threats to the validity of our results.

Engineering challenges vs. practice support. When comparing practice adoption (Table 4.8) with the engineering challenges referenced in Section 4.2, we observe that many challenges are supported by well adopted engineering practices.

In particular, versioning the artefacts related to ML projects, considered a challenge by [14] and corresponding to practice 16 in our study, has a high adoption rate (rank 3). The challenges raised by experiment management [14] and prototyping [160], such as clearly specifying desired outcomes or formulating a problem (practices 6, 7), as well as monitoring experiments and sharing their outcomes (practices 14, 15), also have high adoption rates. These results suggest that these challenges have been met by practitioners.

In contrast, the challenge of testing ML artefacts [14, 117, 160], corresponds to practices 8 and 17, which have low adoption in our study. Although we do not detail all testing methods for ML, as done in [311], the adoption rates for the two testing practices in our study suggests that testing remains challenging.

Several practices presented in this study have low adoption and are not mentioned in previous studies as challenging; this is particularly the case for the practices related to feature management (practices 8, 9 and 10) as well as automating hyper-parameter optimisation and model selection (practice 13). Although these practices have been recommended in the literature, we plan to further validate their relevance through future participant validation (member check) interviews and by collecting additional data.

Threats to validity. We identify three potential threats to the validity of our study and its results. First, the data extracted from literature may be subject to bias. To limit this bias, several authors with different backgrounds have been involved in the extraction process. Also, the pilot interviews and survey produced no evidence suggesting that any of the practices we identified are not recognised by practitioners, nor did we find any indications that important practices were missing from our list. Nevertheless, in the future, we intend to test completeness and soundness of our catalogue of practices through validation interviews.

Second, the survey answers may be subject to bias. As shown in Section 4.5, some groups of respondents are over-represented and may introduce selection bias. In particular, although the adoption rates for respondents in Europe do not present striking differences when compared to those in South America or Asia, Europe remains over-represented. Also, some bias may stem from respondents in North America, for which the adoption patterns are different, while they are not equally represented to other groups. This bias can be removed by gathering more data, as we plan to do in the future.

Last, the measurements used to investigate the relationship between groups of practices and their intended effects may be subject to bias. Rather than measurements of *actual* effects, we used the *perceived* effects as evaluated by the survey respondents. We have not established that perceived effects indeed reflect actual effects, which is an important and ambitious topic for future research.

4.8 Conclusions and future research

We studied how teams develop, deploy and maintain software solutions that involve ML components. For this, we mined both academic and grey literature and compiled a catalogue of 29 SE best practices for ML, grouped into 6 categories. Through a survey with 313 respondents, we measured the adoption of these practices and their perceived effects.

Contributions. We reported on the demographic characteristics of respondents and the degree of adoption of (sets of) practices per characteristic. For example, we found that larger teams tend to adopt more practices, and that traditional SE practices tend to have lower adoption than practices specific to ML. We also found that tech companies have higher adoption of practices than non-tech companies, governmental organisations or research labs.

Further analysis revealed that specific sets of practices correlate positively with effects such as traceability, software quality, agility and team effectiveness. We also trained ML models that can predict with high accuracy the perceived effects from practice adoption.

We contrasted the importance of practices, i.e., their impact on desirable effects as revealed by these predictive models, with practice adoption, and thus indicating which practices merit more (or less) attention from the ML community. For example, our results suggest that traceability would benefit most from increased adoption of practice 25, the logging of production predictions with model versions and input data. At the level of teams or organisations, these same results can be used to critically assess current use of practices and to prioritise practice adoption based on desired effects. For example, a team with a strong need for *agility* and low adoption of associated practices may plan to increase adoption of those practices.

Future research. We plan to further increase the number of respondents of our survey, so we can perform even more fine-grained analyses. We may also add more questions, for example to better measure the effects of practices related to AutoML, a relatively new direction that is receiving sharply increasing attention in academia and industry. We also plan to better cover the traditional best practices from SE, using a process similar to the other practices. Through validation interviews with respondents, we plan to add depth to the interpretation of our findings, especially regarding the relationships between practices and their effects. We also intend to develop and test a data-driven assessment instrument for ML teams, to assess and plan their adoption of engineering practices. While our study is restricted to ML we may also investigate to which extent our findings are applicable for other domains within the broader field of AI. Overall, our hope is that this line of work can facilitate the effective adoption of solid engineering practices in the development, deployment and use of software with ML components, and thereby more generally contribute to the quality of AI systems. Furthermore, we are convinced that other areas of AI would benefit from increased attention to and adoption of such practices.

To strengthen the online catalogue in this direction, we extended it with various practices related, for example, to trustworthy AI development [247].

Table 4.8: SE best practices for ML, grouped into 6 classes.

Nr.	Title	Class
1	Use Sanity Checks for All External Data Sources	Data
2	Check that Input Data is Complete, Balanced and Well Distributed	Data
3	Write Reusable Scripts for Data Cleaning and Merging	Data
4	Ensure Data Labelling is Performed in a Strictly Controlled Process	Data
5	Make Data Sets Available on Shared Infrastructure (private or public)	Data
6	Share a Clearly Defined Training Objective within the Team	Training
7	Capture the Training Objective in a Metric that is Easy to Measure and Understand	Training
8	Test all Feature Extraction Code	Training
9	Assign an Owner to Each Feature and Document its Rationale	Training
10	Actively Remove or Archive Features That are Not Used	Training
11	Peer Review Training Scripts	Training
12	Enable Parallel Training Experiments	Training
13	Automate Hyper-Parameter Optimisation and Model Selection	Training
14	Continuously Measure Model Quality and Performance	Training
15	Share Status and Outcomes of Experiments Within the Team	Training
16	Use Versioning for Data, Model, Configurations and Training Scripts	Training
17	Run Automated Regression Tests	Coding
18	Use Continuous Integration	Coding
19	Use Static Analysis to Check Code Quality	Coding
20	Automate Model Deployment	Deployment
21	Continuously Monitor the Behaviour of Deployed Models	Deployment
22	Enable Shadow Deployment	Deployment
23	Perform Checks to Detect Skews between Models	Deployment
24	Enable Automatic Roll Backs for Production Models	Deployment
25	Log Production Predictions with the Model's Version and Input Data	Deployment
26	Use A Collaborative Development Platform	Team
27	Work Against a Shared Backlog	Team
28	Communicate, Align, and Collaborate With Multidisciplinary Team Members	Team
29	Enforce Fairness and Privacy	Governance

Part II

Designing robust components

5

Adversarial Machine Learning

In the the second part of the thesis we address the challenge of designing robust deep learning models in the small world. In particular, we investigate the design of robust deep learning based computer vision models against intentional perturbations (adversarial examples), and the design of robust deep learning based planning algorithms. In this chapter, we give a brief introduction to adversarial examples – inputs intentionally designed to decrease the performance of deep learning models, while being in close resemblance to training data. Given the surprisingly small size the perturbation needed to create adversarial examples, a wide body of literature conjectures on their existence, and how this phenomenon can be mitigated. A complete characterisation of adversarial examples can be found in our previous publication [242]. Here, we focus on describing the most common methods to generate and protect against adversarial examples, and discuss their relevance to safety and security of deep learning models.

This chapter is a short summary of *A. Serban, E. Poll, J. Visser, Adversarial Examples on Object Recognition: A Comprehensive Survey, ACM Computing Surveys, 2020* [242].

5.1 Introduction

We discuss the ability of DL models to cope with uncertainties in the operational environment, also called algorithmic robustness. Recent publications [198, 267] showed DL models exhibit low robustness, and triggered an impressive wave of publications. Notably, DL models are sensitive to small, *intentional*, perturbations – used to build inputs which substantially decrease their performance, while being in close resemblance to training data. The term *adversarial examples* was first used to describe such inputs by Szegedy et al. [267].

Since an *intention* is required, many publications claim security consequences, e.g., [96, 142, 187, 199], and hypothesize that commercial deployment is hindered by low robustness. In contrast, other publications show these claims are sometimes exaggerated and demand that clear security requirements are formulated before security consequences are claimed [45, 86]. In between, many publications investigate the existence of adversarial examples from a theoretical perspective and shed light on this particular behaviour of ML algorithms [39, 235]. Overall, there are two emergent reasons to study adversarial examples: (1) because attackers might use them to exploit ML algorithms and (2) because they show that ML algorithms are not robust, which may stop them from being adopted in some domains (particularly in safety-critical systems).

Although adversarial examples can be found for a variety of tasks, we restrict the presentation to object recognition because this task is particularly relevant to autonomous systems, and to the i-CAVE project. Nevertheless, adversarial example are constantly explored in other tasks. Of particular interest is malware detection [92, 111, 139] because it implies direct consequences on security. Other tasks such as speech recognition [43, 44], facial recognition [257] or video processing [152, 272, 288] are also explored.

This chapter is organised as follows. We start with a brief characterisation of adversarial examples in Section 5.2. Next, we introduce methods to create adversarial examples in Section 5.3 and defences in Section 5.4. We show that adversarial training – i.e., including adversarial examples in the training data set – is the most effective defence to date. The chapter ends with a general discussion about the implications of adversarial examples to robustness, safety and security of DL models in Section 5.5. Conclusions follow in Section 5.6.

5.2 ML background and adversarial examples

Prerequisites. A computer is said to learn from experience w.r.t. a task and a performance measure if its measured performance on the task increases with experience [179]. In this chapter, we focus on the task of object recognition: given a set of images defined on the input space $\mathcal{X}$ with their labels from the output space $\mathcal{Y}$, sampled from a fixed, but unknown probability distribution $\mathcal{D}$ over the space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, a ML algorithm attempts to find a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ which minimizes the number of misclassified samples. We assume that $\mathcal{X}$ is a metric space and we can define distance functions between two points of the space. The error made by a prediction $f(x_i) = \hat{y}_i$ when the true label is y_i is measured by a loss function $l : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$. Through learning, we select a function f^* from a hypotheses space $\mathcal{F}$ such that the expected loss $r(f) = \mathbb{E}_{(x,y)\sim\mathcal{D})}[l(f(x), y)]$ is minimal: $f^* = \arg\min_{f\in\mathcal{F}} r(f)$. In practice, $\mathcal{D}$ is not known and only a set of samples $\mathcal{S}$ (defined as a set of pairs $\{(x_i, y_i)\}_{i=1}^{n}$)

is available for training. Thus, a ML algorithm uses the empirical loss to approximate the expected loss:

$$\hat{f} = \arg\min_{f \in \mathcal{F}} \quad \mathbb{E}_{(x,y)\sim S}[l(f(x), y)]. \tag{5.1}$$

The hypotheses space $\mathcal{F}$ can be any mapping from $\mathcal{X}$ to $\mathcal{Y}$ such as a linear function or a DNN. Choosing $\mathcal{F}$ for a task adds an inductive bias from the algorithm designer and involves a trade-off between expressivity and generalization: if $\mathcal{F}$ is not expressive enough, the algorithm will not be able to learn complex hypotheses. On the opposite, if $\mathcal{F}$ is too expressive, the algorithm will overfit on the training data. The loss function is generally chosen to be zero when $f(x_i) = y_i$ and positive otherwise. The most common loss function for object recognition is the cross-entropy loss.

The Probably Approximately Correct (PAC) [278] theoretical model for statistical learning guarantees that given enough samples for a desired accuracy ϵ and for the probability of getting non-representative samples from the training distribution δ ($0 < \epsilon, \delta < 1$), the empirical risk will have an error less than or equal to ϵ with probability $1 - \delta$: $P(|r(\hat{f}) - r(f^*)| \leq \epsilon) \geq 1 - \delta$. In this framework, given the choice for ϵ and δ, we can derive the sample complexity for learning a hypothesis with minimal risk. An important assumption of this model is that training and test data are drawn from the same probability distribution $\mathcal{D}$. Moreover, all data are sampled independently from distribution $\mathcal{D}$ (also called independent and identically distributed (i.i.d)). A hypothesis behind the existence of adversarial examples is that they are sampled from a different distribution than the training data [85, 147, 172, 263]. However, this hypothesis was questioned by developing attacks that can easily bypass detectors which learn the distribution of adversarial examples [41].

Adversarial Examples. Adversarial examples are inputs intentionally designed to be in close resemblance with samples from the distribution $\mathcal{D}$, but cause a misclassification. Formally, given a classification function f and a clean sample x, which gets correctly classified by f with label y, an adversarial example x' is constructed by applying the minimal perturbation η to input x such that x' gets classified with a different label $\hat{y}$: $\arg\min_c f(x + \eta) = \hat{y}$. Similarly, in the initial paper on adversarial examples, Szegedy et al. [267] search for the perturbation solving the following optimisation problem:

$$\begin{aligned} &\min_{\eta}. \quad \eta = \|x' - x\|_p, \\ &s.t. \qquad f(x') = \hat{y}, \end{aligned} \tag{5.2}$$

where $\|\cdot\|_p$ is a distance function defined on the metric space $\mathcal{X}$. Searching for the minimal perturbation is often a complex task because the search space is non-linear and non-convex [145, 199]. However, many approximation solutions have been proposed. Finding solutions to Eq. 5.2 is illustrated in Figure 5.1. Some examples of perturbations are illustrated in Figure 5.2.

The distance function most commonly used for adversarial examples in the object recognition domain is the p-norm:

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \tag{5.3}$$

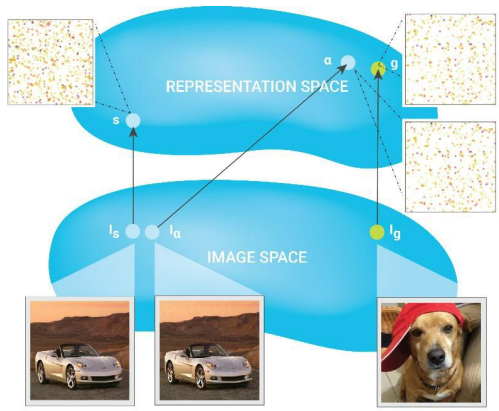


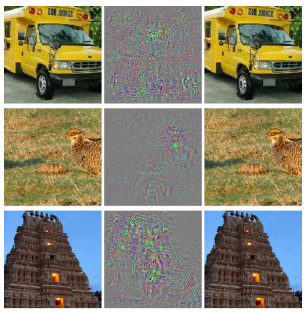
Figure 5.1: Adversarial example in input and representation space [225]. While the two pictures of cars are similar in the image space, the activation patterns of the second car are close to the activation patterns of the dog. Therefore, the second car gets classified as a dog. Moving the activation patterns from cars to dogs while keeping the representation in the image space similar is equivalent to searching for a solution to Eq. 5.2 and generating an adversarial example.

where $p \in \{0, 2, \infty\}$. The choice for p influences the coordinates changed in the initial sample as follows:

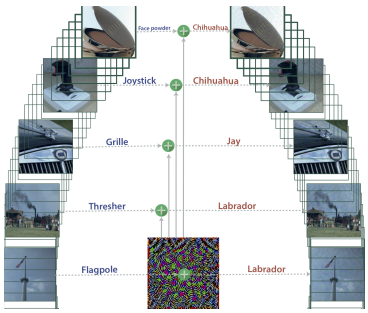
- when $p = 0$ the distance measures the number of different coordinates between the normal input and the adversarial; corresponding to the number of pixels altered in the original image.
- when $p = 2$ the distance measures the Euclidean distance between the original and the adversarial image. This metric remains small when there are many small changes to many pixels and increases when there is a big change in one or multiple pixels.
- when $p = \infty$ the distance measures the maximum change in any of the coordinates and is equivalent to the maximum bound for changing each pixel in an image, without restricting the number of changed pixels.

Historical considerations. Even though the term adversarial examples was first coined around 2014 in research by Szegedy et al. into DNNs [267], adversarial machine learning was established long before. Unfortunately, as other authors have also observed [30, 87], recent publications concerning DNNs seem unaware of the earlier research on adversarial machine learning and loose important perspective in this field. In particular, the importance of thread modelling to security is overlooked.

The first publication regarding adversarial ML was published in 2004, when Dalvi et al. [62], followed by Lowd and Meek [157], managed to fool linear classifiers for spam detection by making changes to spam e-mails [30]. Barreno et al. [20] first introduced a taxonomy for attacks and defences in adversarial settings, and later refined it in [21].



(a) Specific perturbations for each new input. The images in the first column are inputs correctly classified, the ones in the middle are the perturbations and the images on the last column are the resulting adversarial examples [267].



(b) Universal perturbations – only one perturbation can be applied to any picture on the left to generate adversarial examples on the right [182].

Figure 5.2: An illustration of adversarial examples.

This early taxonomy defines ML threat models and is comprehensive enough to include adversarial examples. However, the notion of minimal perturbation was not yet adopted.

Thereafter, a large body of publications discussed adversarial attacks against ML models at both *training* time [31, 224] and *test* time [88, 157] or defences against such attacks [38, 137]. Attacks at training time modify or *poison* the training data set (before training), while attacks at test time only modify the samples used for test (after training). In parallel to developing attacks and defences, several publications proposed methods to evaluate the security of ML models against adversarial attacks [21, 32]. Biggio and Roli [30] trace an interesting parallel between the evolution of adversarial ML and the rise of DNNs.

Adversarial examples represent attacks against machine learning models at test time. Moreover, they have a special trait: the perturbations used to fool classifiers are desired to be minimal, or as small as possible. In practice, such perturbations are very small and barely noticeable to human observers (see Figure 5.2). In this thesis we are concerned with recent literature, triggered by Szegedy et al. [267] and the widely adopted definition of adversarial examples presented in Eq. 5.2. This body of work focuses on DNNs and was triggered by the surprisingly small perturbations needed to fool such algorithms.

From a security standpoint, we can make another distinction between publications before and after Szegedy et al. [267]: generally, publications before Szegedy et al. look at attacks on systems providing *security functionality* (e.g., spam or virus detection), in contrast to more recent papers [30, 200] which look at *secure functionality* of *any* application of ML algorithms, i.e., if any application of ML algorithms is secure. This distinction will be further developed in Section 5.5.

Overviews. We published a comprehensive study of adversarial examples [242], which builds on previous work [3, 156] by relating the threats posed by adversarial examples to security, safety and robustness of MLs. Moreover, we discuss the hypotheses on the

existence of adversarial examples and their property of being transferable between different ML models. For detailed information please refer to the publication mentioned above [242].

5.3 Methods to generate adversarial examples

General optimisation algorithms – e.g., L-BFGS – can be successfully used to find solutions to Eq. 5.2 [267]. However, depending on the input size the computation may require a large computational budget. In order to speed up adversarial attacks, Goodfellow et al. [91] showed that taking a small step towards increasing the loss function w.r.t. to an input suffices to find perturbations. The procedure – called the fast gradient sign method (FGSM) – is defined as:

$$\eta = \epsilon \operatorname{sign}(\nabla_x l(\theta, x, y)), \tag{5.4}$$

where ϵ controls the perturbation budget. Although not optimal, FGSM only requires to evaluate the gradient w.r.t. an input once, which makes it very fast. However, FGSM searches for perturbations in one direction and is therefore not very effective. In order to strengthen FGSM, Madry et al. [162] proposed to iteratively apply FGSM and project the outcome in the norm ball defined using the perturbation budget ϵ:

$$\mathcal{U}_x = \{x' | \, \|x' - x\|_p < \epsilon\}. \tag{5.5}$$

This procedure – called projected gradient descent (PGD) – is defined as:

$$x'_n = \prod_{x+\eta} (x'_{n-1} + \epsilon \operatorname{sign}(\nabla_{x'_{n-1}} l(\theta, x'_{n-1}, y))). \tag{5.6}$$

Naturally, the quality of the resulting adversarial examples depends on the number of iterations n. Using a large number of iterations, PGD can better approximate the norm ball around an input, and lead to more representative adversarial examples. However, it negatively affects the computation time. When $n = 1$, PGD is equivalent to FGSM.

Besides the attacks based on optimisation algorithms (L-BFGS) and on FGSM, a large number of attacks have been proposed by optimising surrogate functions [42], using evolutionary algorithms [187, 265] or generative models [19, 209]. A common characteristic of these attacks is that they require access to the model parameters. Therefore, from a security point of view, they can be described as *white-box* attacks [242].

Szegedy et al. [267] showed that adversarial examples also transfer between different ML models. This property was later explored by Papernot et al. [198] in an attempt to generate *black-box*, attacks i.e., attacks which do not require access to the model parameters. The authors showed that adversarial examples crafted on one model can transfer between several ML techniques such as linear regression, support vector machines or DNNs. Therefore, the attacker can train a substitute model and craft adversarial examples without access to the parameters of the model under attack.

This investigation triggered a new generation of attacks which are more relevant to real world scenarios; in which attackers do not have access to the model parameters, but can query it. For example, Chen et al. [53] present an attack based on zeroth-order optimisation that is derivative free. This method can estimate the gradient across the perturbation's direction taking into consideration the value of the objective function at two neighbouring

points (corresponding to adding or subtracting a small perturbation). Thus it also eliminates the need to train substitute models. Similar approaches were developed by Narodytska and Kasiviswanathan [184], Ilyas et al. [115] or Rosenberg et al. [221] outside the object recognition domain.

More recently, attacks based on ensembles of diverse attacks have been proposed as an extension of PGD, in order to overcome some failures due to sub-optimal step sizes, or mis-specification of the objective function [59].

5.4 Defences against adversarial examples

Similar to adversarial attacks, a wide range of defences have been proposed based on different strategies [242]: e.g., detection of adversarial examples [93, 175], input transformations [52, 214], feature removal [80], or using formal methods to verify that adversarial examples can not be found within some bounds [67, 296].

However, most defences only alleviate a model's sensitivity to small changes in the input by minimising the gradients during the learning phase, or by constructing models without useful gradients. Nonetheless, forbidding access to gradient information is not enough to limit an attacker from constructing adversarial examples [199]. This phenomenon, called *gradient masking* [199, 200] was identified to give a false sense of security and leads to an improper evaluation of adversarial defences [15, 41, 274]. Defences that exploit gradient masking can be sometimes broken with stronger attacks [41, 45]. Moreover, defences which rely on formal verification can only provide guarantees for the training data set, and can be bypassed by slightly different test data [209].

The most effective adversarial defence to date (which does not rely on gradient masking) is adversarial training, and consists of adding a regularisation term to the loss function:

$$\tilde{l}(\cdot) = \alpha l(\theta, x, y) + (1 - \alpha) l(\theta, x', y), \tag{5.7}$$

where x' is an adversarial example generated from input x and α controls the contribution of adversarial examples to the loss function. The most effective choice of α is zero [162], which poses the learning problem as a min-max problem where the inner maximisation seeks to find the worst adversarial example for an input, and the outer minimisation seeks to strengthen the model against it:

$$f = \arg\min_{f \in \mathcal{F}} \mathbb{E}_{(x,y)\sim\mathcal{S}}[\max_{x' \in \mathcal{U}_x} l(\theta, x', y)], \tag{5.8}$$

The inner maximisation can be solved with any efficient algorithm from Section 5.3. In practice, only the PGD attack was used successfully. However, in order to obtain an accurate estimation of $\mathcal{U}_x$, PGD requires a large number of iterations. The number of iterations has a direct impact on training time because each iteration of PGD requires to compute the gradients. In order to limit this impact, PGD is commonly used with less than ten iterations for training, but with more iterations for testing. Faster ways to perform adversarial training exist, and will be discussed in Section 6.2.

We also mention that adversarial training brings benefits beyond robustness, such as more interpretable gradients [276] or better transferability [226]. However, a trade-off between robustness and accuracy is known to exist [276].

5.5 Implications of adversarial example to robustness, safety and security of ML

Given the large number of publications and claims regarding adversarial examples, we discuss the relevance of adversarial robustness to security, safety, and the economics of building more robust models. Moreover, since p-norm is the dominant similarity metric, we also comment on its relevance. This section ends with a discussion on the representations learned by DNNs, and how they impact adversarial examples.

On the relevance of robustness to security. Gilmer et al. [86] express some skepticism about whether adversarial examples are always a serious security concern. Here it is interesting to note again that much of the early work on adversarial machine learning [21, 157] concerned applications of ML for security tasks, such as detecting spam, malware, or network intrusions. In such applications there is by definition an attacker interested in causing misclassification, as the whole point of the system is to defend against such an attacker, and hence miss-classifications have a security impact. By contrast, most recent work on adversarial learning focuses on computer vision. While adversarial examples may seem worrying thinking of some applications of computer vision for autonomous systems, this does not imply that there is an interesting way for attackers to exploit it. For example, Eykholt et al. [74] use perturbed stop signs to attack autonomous vehicles. However, the perturbations are far from sensible and can be detected by human observers. Simply obscuring or removing the sign may be easier ways to achieve the same effect.

On the relevance of robustness to safety. From an engineering perspective, safety is the ability of a system to protect its users from harmful or non-desirable outcomes. The distinction between security and safety is that security protects a system against *intentional*, malicious, attacks while safety protects a system from *unintended* mishaps in the operational environment. Some publications aim to improve or validate the safety of DNNs – e.g., [84, 113, 158]. However, safety is an inherent property of a system and not of an algorithm solely. Moreover, safety becomes important when a system can produce physical or material damage to humans, assets or the environment. Talking about safety for systems without such impact – e.g., an image based search engine using ML – is futile. In order to guarantee safety, one should make sure that possible errors are detected and contained inside the system without affecting its normal operation. Take the example of an autonomous vehicle. If the outcome of its computer vision system is cross checked with information coming from maps, the effect of using adversarial examples on traffic signs can be detected and contained inside the system, reducing their impact on safety.

The discussion of ML safety in relation to robustness should take into account the operational environment of an algorithm because some perturbations (such as those needed to build adversarial examples) may never appear in some environments, but may be common

in others. Besides, it might also be interesting to benchmark an algorithm and increase its robustness to common corruptions and perturbations [103].

On the relevance of p-norm. The dominant similarity metric in the literature is the p-norm distance defined in Eq. 5.3. Choosing an adequate metric is still an open question. However, since there are no solutions to robustness for the p-norm distance, it is hard to believe that using other metrics will result in more robust models [45]. Nonetheless, there is an increased interest to explore new distance functions, e.g., the Wasserstein distance [297] or using physical parameters underlying the image formation process [155]. We argue that the p-norm remains relevant for experimental settings; however, it must be paired with relevant threat models in order to evaluate its impact in security, and search for operational environments where it impacts safety.

On the economics of defending against adversarial examples. Until now there seems to be a trade-off between accuracy and robustness to adversarial examples, inherent to the algorithms and the training methods used. This means robustness comes at a cost. Whether these cost are acceptable, and how high they can be, will depend on the application and the context. Given that the real impact of adversarial examples on safety and security is still to be determined, it remains to be seen which defences can be cost-effective in practice. Nonetheless, recent draft regulation emphasises robustness [77] and hints towards a future in which robustness will be needed for compliance.

On the representations learned by DNNs. The sensitivity of DNNs to adversarial examples raises questions about their ability to learn high level abstractions from data. Although it is believed that increasing the depth of a network helps increasing the level of abstraction and it was observed that early layers in convolutional networks learn filters that resemble contour extractors, while deeper layers learn more complex patterns, DNNs seem to learn superficial abstractions restricted to the space on which they operate. In object recognition, the training objectives lie in pixel space, and not in a conceptual or relational space. Pixel spaces are necessary for extracting first order information about the task, but seem to be insufficient for higher level abstractions needed to overcome complex perception systems. Moreover, the capacity to create adversarial inputs which are not intelligible by humans (as in [187]) shows that DNNs use different features than we wish for. Research in adversarial examples strengthen the conclusions from Jo and Bengio [122] which analysed convolutional networks in different regimes and showed they exhibit a tendency to learn surface regularities, rather than higher-level abstract concepts. Therefore, adversarial examples might be intrinsic to the methods used to solve ML tasks, or to the current training procedures. In this context, it is interesting to search for models which learn a better representation of the world and which may solve the sensitivity to adversarial examples as a side effect. Research carried out recently shows that robustness against adversarial examples can indeed be achieved this way, using supervision from language representations, and by training very large models [212]. However, this opens up a new range of adversarial attacks [89].

5.6 Conclusions

We provided a brief introduction to the adversarial examples phenomenon that will support the next two chapters. For a comprehensive study of this phenomenon, we refer the reader to our previous publication [242]. We note that adversarial examples are an intriguing phenomenon of DL algorithms, and their existence can raise both safety or security alarms. A key take away is that the phenomenon of adversarial examples has no generally accepted explanation or solution. Moreover, until now all defenses (including the ones using formal verification) have been broken. Therefore, the field remains active and spans several future research directions.

In the next two chapters, we focus solely on decreasing the impact of adversarial training on training time.

6

Deep Repulsive Prototypes for Adversarial Robustness

As discussed previously, the most compelling defence against adversarial examples is adversarial training, and consists of complementing the training data set with adversarial examples. Yet adversarial training severely impacts training time and depends on finding representative adversarial samples. In this chapter, we propose to train models on output spaces with large class separation in order to gain robustness without adversarial training. We introduce a method to partition the output space into class prototypes with large separation and train models to preserve it. Experimental results shows that models trained with these prototypes – which we call deep repulsive prototypes – gain robustness competitive with adversarial training, while also preserving more accuracy on natural samples. Moreover, the models are more resilient to large perturbation sizes. For example, we obtained over 50% robustness for CIFAR-10, with 92% accuracy on natural samples and over 20% robustness for CIFAR-100, with 71% accuracy on natural samples without adversarial training. For both data sets, the models preserved robustness against large perturbations better than adversarially trained models.

This chapter has been published as *A. Serban, E. Poll, J. Visser, Deep Repulsive Prototypes for Adversarial Robustness, arxiv, 2021* [240].

6.1 Introduction

As mentioned in Section 5.4, adversarial training is subject to several trade-offs. Firstly, the time needed to generate adversarial examples substantially increases training time. Recent attempts to generate adversarial examples faster exist [298]. However, they are (at the moment) unstable and introduce new issues such as catastrophic forgetting [10].

Secondly, a trade-off between accuracy on natural samples and robustness on adversarial examples is known to exist [307]. This trade-off implies that robustness against adversarial examples comes with a cost of losing accuracy on natural examples, and can be controlled through adversarial training [307]. Lastly, adversarial training overfits on training data and provides little robustness against data outside this distribution [217, 309].

A model robust to adversarial examples should provide: (i) inter-class separability, (ii) intra-class compactness, and (iii) marginalisation or removal of non-robust features [116, 260]. However, adversarial training does not impose explicit constraints for meeting these properties (e.g., inductive biases). Therefore, it depends only on finding representative adversarial examples for training.

In supervised classification, adversarial training uses the standard softmax cross-entropy loss. Recently, there is increasing evidence that softmax partitions the output space into class centroids situated at equal distance from the origin (inter-class separability), and that adversarial robustness can be improved by clustering the data points in the proximity of these centroids (intra-class compactness) [105, 196].

However, the distance between class centroids is insufficient to provide robustness, and even models with high intra-class compactness are vulnerable to adversarial attacks. In this chapter we tackle this issue by enforcing large inter-class separation prior to training using class prototypes [262]. Using this inductive bias we gain more control over the output space structure, and can decrease the number of training samples needed [174].

We show that training with class prototypes optimised to provide large inter-class separation helps to gain robustness competitive with adversarial training, *without* adversarial training. Moreover, training with class prototypes involves a smaller trade-off between accuracy and robustness, and a higher resilience against large perturbations. The prototypes are built prior to training with little overhead, through an optimisation procedure that increases the distance between their centres. As a result of this repelling optimisation procedure, and because we use deep neural networks for empirical validation, we call the prototypes *deep repulsive prototypes*. We test repulsive prototypes on CIFAR-10 and CIFAR-100, and observe consistent results on both data sets, with 51.3% and 20.5% robustness against iterative adversarial attacks.

This chapter is organised as follows. Initially, we introduce background information and discuss related work in Section 6.2. Later, we present repulsive prototypes in Section 6.3, followed by an evaluation against white and black-box attacks in Section 6.4. We conclude with a discussion in Section 6.5 and future work in Section 6.6.

6.2 Background and related work

As mentioned in Section 5.4, the quality of adversarial examples depends on the number of iterations n from Eq. 5.6, page 78. Using a large number of iterations, projected gradient

descent (PGD) can better approximate the space around an input we want to provide robustness to, and leads to more representative adversarial examples for training. However, it negatively affects training time.

Several attempts have been made to change the training procedure in order to enforce inter-class separability or intra-class compactness, and also decrease the impact of adversarial training. Mao et al. [164] used a triplet loss (inspired by metric learning), where one element of the triplet loss is an adversarial example. An attempt to reduce the impact of adversarial training on training time was made by only generating one adversarial example for each triplet data. Further inductive biases, such as careful negative sample selection for the triplet loss, help to improve robustness.

Papernot and McDaniel [196] showed that explicitly tailoring intra-class compactness using k-neighbours in the representation space helps to detect adversarial examples. Hess et al. [105] proved a similar result and proposed a method based on the Gauss kernel to enforce intra-class compactness and improve robustness. Pang et al. [193] introduced a loss function to enforces inter-class separability using the centroids of the Max-Mahalanobis distribution. During inference, the class centroid closer to the input's deep representation (measured using the Euclidean distance) was used to classify an input. The defence builds on earlier work by Pang et al. [194], where at inference time an input is interpolated with samples from the same predicted class, and from distinct classes in order to alleviate the impact of perturbations. Unfortunately, none of these defences proved effective [275].

Jin and Rinard [121] showed that manifold regularisation improves adversarial robustness significantly while retaining better accuracy on natural examples, without adversarial training. Their proposal induces local stability in the neighbourhood of natural inputs even if the model classifies the inputs incorrectly. This is in contrast with adversarial training, where a model is trained to classify correctly worst case adversarial examples.

Mustafa et al. [183] used class prototypes to enforce inter-class separability, by including a prototype separation constraint in the loss function. A convex polytope is assigned as prototype to each class and during training the distance between all class polytopes is maximised. Thus the class centroids are learned together with the internal representation. However, adding the distance maximisation term to the loss function does not suffice to improve robustness, and they propose to add similar constraints to hidden layers. When paired with adversarial training, robustness increases at a decreased cost for accuracy on natural samples.

Mettes et al. [174] showed that defining class prototypes a priori to training in order to enforce desired properties of the output space (e.g., large margin separation) improves training in several settings; such as few-shot classification, regression or joint classification and regression. Instead of constantly re-estimating and calibrating the prototypes – as in Mustafa et al. [183] or others [94, 262] – they propose to define the output space as a hyper-sphere and partition it into predefined class prototypes. During training, the distance between the model's output and class prototypes is minimised. The a priori definition of class prototypes enables control over several factors such as the output space size, or its shape. In this paper we take a similar path and define class prototype prior to training.

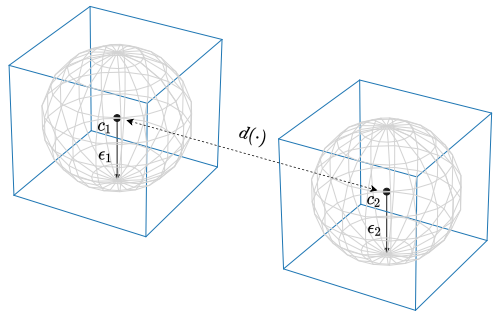


Figure 6.1: Repulsive prototypes. The grey spheres represent prototypes obtained using the L_2 distance, and the blue cubes are obtained using the L_∞ distance.

6.3 Repulsive prototypes for robustness

The idea behind building class prototypes for adversarial robustness is to explicitly design prototypes with large inter-class separation, and during training enforce intra-class compactness. To this end, the input or the output space is partitioned into hyper-planes specific to each class, to which we impose separation constraints. Similar approaches have been used in the past, e.g., by Schiilkop et al. [233] who used the smallest sphere enclosing the data to estimate the VC-dimension for support vector classifiers, or by Wang et al. [284] who used separating spheres in the feature space for classification.

We propose an approach similar to Wang et al. [284] and Nguyen and Tran [188], and more recent work by Mettes et al. [174], where the separation boundaries are imposed to the output space – instead of the input space – because the output space allows more flexibility and can achieve larger margin separation. Prior to learning, the D-dimensional output space is divided into k prototypes $P = \{p_1, \cdots, p_k\}$, where each prototype corresponds to a class. For a binary classification problem and a Euclidean output space, we wish to find two hyper-spheres with centres at c_1, c_2 – one enclosing samples from the positive class and the other enclosing samples from the negative class – and maximise the distance between them:

$$\begin{aligned} \min_{\epsilon_1,\epsilon_2,c_1,c_2} \quad & \epsilon_1^2 + \epsilon_2^2 - r\|c_1 - c_2\|^2 \\ s.t. \quad & \|f(x_i) - c_1\|^2 \leq \epsilon_1^2, \forall i, y_i = +1 \\ & \|f(x_i) - c_1\|^2 \geq \epsilon_1^2, \forall i, y_i = -1 \\ & \|f(x_i) - c_2\|^2 \leq \epsilon_2^2, \forall i, y_i = -1 \\ & \|f(x_i) - c_2\|^2 \geq \epsilon_2^2, \forall i, y_i = +1, \end{aligned} \tag{6.1}$$

where r is a constant that represents the repulsive degree between the two prototypes, and ϵ_1, ϵ_2 define the $d(\cdot)$-ball around the prototype centres for which we want to provide robustness (corresponding to the uncertainty set in Eq. 5.8, page 79).

For non-separable data sets, the constraints above can be relaxed by introducing slack variables and regularisation terms to the objective function. Although the objective in

Eq. 6.1 is not convex, it can be reformulated to have a convex form and solved using Lagrange multipliers. However, in practice the constraints can be relaxed and the problem can be solved in two steps: firstly find prototypes with large separation (to provide inter-class separability), and secondly train models to fit the data within the proximity of the prototype centres (to provide intra-class compactness). An approximate solution to the first problem can be found using gradient descent on the unconstrained objective:

$$\min_{\epsilon_1,\epsilon_2,c_1,c_2} \quad \epsilon_1^2 + \epsilon_2^2 - r\|c_1 - c_2\|^2,$$

with a generalisation to k-classes and any metric space:

$$\min_{\epsilon,c} \quad \epsilon^k - r \sum_{(i,j,i\neq j)\in k} d(c_i, c_j). \tag{6.2}$$

The choice of r can also be controlled using the learning rate μ for gradient descent. The choice of $d(\cdot)$ influences the prototypes and the classification regions defined in the output space. For example, using the Euclidean L_2 distance leads to hyper-spherical classification regions, and using the Chebyshev L_∞ distance leads to hyper-cubical regions. An illustration is provided in Figure 6.1, where the grey spheres represent regions for perturbations in the L_2 space and the blue cubes are regions for perturbations in the L_∞ space around the centres. Iterating over Eq. 6.2 is equivalent to increasing $d(\cdot)$ or adding slack variables to Eq. 6.1. Larger distances between class prototypes introduce buffers between classification boundaries and should improve robustness.

The second step – training models to fit the data within the proximity of the prototype centres – can be solved by minimising the distance between the prototype centres and the model's output. The choice for this distance function is part of the threat model and it is the same as $d(\cdot)$ from Eq. 6.2, which induces the following loss function:

$$l = \sum_{i=1}^{N} (1 - d(f(x_i), p_{y_i}))^2, \tag{6.3}$$

where p_{y_i} is the prototype specific to class y_i.

6.4 Empirical evaluation

All experiments are performed using a vanilla ResNet-18 network (the smallest variant of ResNet). Capacity is known to help adversarial robustness [162, 302]. Therefore, we avoid using larger networks. During training with repulsive prototypes only natural samples are used, i.e., *no* adversarial training is performed.

Firstly, we adopt a white-box threat model for testing, where attackers presumably have full knowledge of the model under attack, the training and the testing data [45]. To generate adversarial examples we use the PGD (Eq. 5.6, page 78) PyTorch implementation from Cleverhans, with different iterations and random restarts [197]. Testing against larger n is recommended, as it shows if the model exhibits a false sense of robustness or obfuscates attack vectors [45].

Table 6.1: Prototype selection on CIFAR-10.

Output Dimension (D)	Epochs	Clean Samples	PGD-20
50	50	90.3	37.2
100	50	90.6	39.9
200	50	89.5	40.7
100	100	91.0	48.7
200	100	91.1	38.7

Attackers are constrained to generate adversarial examples in the $\epsilon = 8$ (normalised) L_p norm ball around inputs – a common benchmark for adversarial robustness. Since the distance between prototypes is larger than ϵ, we expect models trained with repulsive prototypes to also exhibit resilience to higher ϵ values. To test this hypothesis, we use robustness curves obtained by step-wise increasing the size of the perturbation in the interval $[8, 16]$.

We compare with results from literature on two common data sets; CIFAR-10 and CIFAR-100 [140]. The first one consists of 60 000 32x32 colour images and 10 classes (with 5 000 images for training and 1 000 images for testing per class). The second data set consists of 60 000 32x32 images and 100 classes (with 500 training images for training and 100 images for testing per class). We use minimal pre-processing for training, consisting of random cropping and random horizontal flip. No pre-processing is used for testing.

For training, we use the cyclical learning rate [259], mixed precision arithmetic and early stopping, as they are reported to improve training time and prevent overfitting [217, 298].

Later in this section we also adopt a black-box threat model, where attackers can only observe the outcome of the models under attack. For evaluation we use the transferability attack, where adversarial examples are generated with a distinct model, and transferred to the models under attack [45, 264].

6.4.1 Prototype Selection

Several parameters influence the quality of the prototypes: the output space dimension D, the choice of ϵ, r, the learning rate μ and the number of epochs for solving Eq. 6.2. Since r and μ can be compressed to one constant, and (to some extent) the effect of the learning rate can be attenuated by running the optimisation longer, the most important parameters are the output space dimension and the number of epochs. As mentioned earlier, when not mentioned otherwise we use the same choice for $\epsilon = 8$ (normalised).

Previously, it has been shown that increasing D can benefit both classification and regression [174]. In order to determine the influence of D on robustness and accuracy, we run an experiment on the CIFAR-10 data set, training a ResNet-18 model for 50 epochs with different output sizes $D \in \{50, 100, 200\}$ – corresponding to multiplying the number of classes k with factors of $\{5, 10, 20\}$. Testing is performed with natural and perturbed samples using the PGD attack, with $n = 20$. In all cases, the prototypes are generated by

Table 6.2: CIFAR-10 results, $\epsilon = 8$. The models are [1]ResNet-18, [2]PreActResNet-18, [3]WideResNet-34-10, [4]ResNet-110.

Run	Epochs	Natural	PGD 20	PGD 100	Adv. Training
Regular[1]	120	93.7	0	0	None
Repulsive[1]	127	92.0	51.3	48.4	None
Madry[2]	200	87.2	45.8	-	PGD-7
Early Stop[3]	100	86.1	56.1	-	PGD-10
TRADES[3]	100	84.9	56.6	-	PGD-10
RHS[4]	300	91.8	42.6	-	PGD-7

running gradient descent on Eq. 6.2 for 100 epochs, with $\mu = 0.01$. This modest optimisation budget is sufficient to obtain large distances between the prototypes.

The results are presented in Table 6.1. We observe that increasing the output dimension D has almost no impact on accuracy on natural samples, but a significant impact on robustness, for all values of D except the last one. For the last two values of D we ran training longer and observe that the largest output space (i.e., 200) has a bigger tendency to overfit for adversarial examples, while maintaining similar accuracy on natural samples (corresponding to 100 epochs in Table 6.1). This phenomenon will be elaborated in Section 6.5.

The initial experiments on prototype selection reveal that the output size is important for adversarial robustness, but plays a marginal role for accuracy on natural samples.

6.4.2 CIFAR-10

Following the previous experiments, we present the results from training a ResNet-18 model on CIFAR-10 with the same parameters as earlier, but run the optimisation for longer and test it against stronger attacks. For all experiments, the output dimension is $D = 100$, corresponding to multiplying the number of classes by a factor of ten.

We benchmark our results against the following results from literature: (i) the initial results for adversarial training from Madry et al. [162], (ii) the improved results for adversarial training from Rice et al. [217] which use early stopping to prevent overfitting in adversarial training, (iii) the work of Zhang et al. [307] which trades more accuracy on natural samples in order to gain robustness, and (iv) the work of Mustafa et al. [183] which use class prototypes jointly optimised during training, and where the inter-class separation constraints are applied to multiple layers, and paired with adversarial training. We note that Zhang et al. report the highest robustness. However, Rice et al. showed that early stopping improves robustness, and reduces the gap between Madry et al. and Zhang et al., while also preserving more accuracy on natural samples.

While Madry et al. and Mustafa et al. use PGD adversarial training with $n = 7$, Zhang et al. and Rice et al. use $n = 10$. As mentioned above, adversarial training adds a non-trivial overhead, and a higher n further increases it. While faster methods to perform adversarial training exist, they achieve *at most* similar results to classical adversarial training. Therefore,

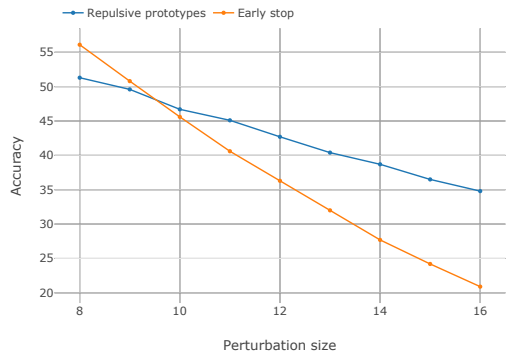


Figure 6.2: Robustness curves for CIFAR-10, obtained by testing with PGD-20, and various perturbation sizes (ϵ).

we compare our results with the state-of-the-art for classical adversarial training. Since our method does not add any significant overhead to training, whenever we discuss the impact on training we compare with Wong et al. [298], which is (at the moment) the fastest way to perform adversarial training, albeit not stable [10].

The results are presented in Table 6.2, where the acronyms follow the order above: (i) Madry [162], (ii) Early Stop [217], (iii) TRADES [307], (iv) RHS [183]. The Regular run was trained on natural samples with the softmax cross-entropy loss, and a multi-step learning rate scheduler that starts from 0.1 and decays by a factor of 0.1 at epochs 50 and 100. For the models in literature we present the reported results, since with the exception of Rice et al. the results could not be reproduced precisely.

For the model trained using repulsive prototypes (the Repulsive run) we report the robustness against the PGD attack with 20 and 100 iterations. During training, the cyclical learning rate was reduced by a factor of ten compared to Smith [259]. We found that using smaller learning rates benefits robustness and has little impact on natural accuracy. The reason for this is that larger updates may push the samples closer to the decision boundaries, where it is easier for adversarial perturbations to induce undesirable behaviour. For all models we also report the accuracy on natural samples, the number of epochs needed to reach the results and the architecture used for training.

We observe that training with repulsive prototypes yields higher accuracy on natural samples (92%) than methods based on adversarial training, and competitive robustness (51.3%) compared with the state-of-the-art (56.6%), at a relatively small increase of training epochs (+27). This is a gain even for Wong et al. [298], which uses the Fast Gradient Sign Method (FGSM) attack and thus requires at least two forward and backward passes at each epoch. Moreover, TRADES and Early Stop use $n = 10$ for adversarial training (which increases robustness over Madry), and use a WideResNet-34-10 architecture, which has over $34 * 10^6$ more training parameters than ResNet-18. Both capacity and a higher n are known to increase robustness [162].

Figure 6.2 illustrates the robustness curve obtained by testing the models with PGD $n = 20$, against different perturbation sizes. For comparison, we use the Early Stop model

Table 6.3: CIFAR-100 results, $\epsilon = 8$. The models are are [1]ResNet-18, [2]PreActResNet-18, [3]ResNet-110.

Run	Epochs	Natural	PGD 20	PGD 100	Adv. Training
Regular[1]	120	73.8	0	0	None
Repulsive[1]	106	71.7	20.5	20.0	None
Madry[2]	200	59.8	22.6	-	PGD-7
Early Stop[2]	100	52.7	28.1	-	PGD-10
Early Stop-R[2]	100	54.1	20.8	-	PGD-10
RHS[3]	300	68.3	20.2	-	PGD-7

by Rice et al., the only one for which the results could be reproduced with precise accuracy. We observe that training with repulsive prototypes yields models which are more resilient to higher perturbations than adversarially trained models (equivalent to a milder slope in Figure 6.2). Moreover, the overall decrease in accuracy is significantly smaller for models trained with repulsive prototypes; preserving more than 60% of the initial robustness when the perturbation size is doubled.

6.4.3 CIFAR-100

We perform and report complementary experiments on the CIFAR-100 data set. The key difference between the two is that the number of classes increases by a factor of ten. Therefore, the output space partitioning is more challenging.

Moreover, since the last fully connected layer of ResNet-18 has 512 nodes, we use a multiplicative factor of 50 instead of 100 for D in order to preserve a possible compression in the last layer, as for CIFAR-10. Experiments with different multiplicative factors, as those discussed in Table 6.1, are available in the project's repository.

The results are presented in Table 6.3, with the notable difference that TRADES was not tested on this data set neither in the original paper [307] or in the Early Stop paper [217]. Moreover, for Early Stop we could not reproduce the results reported in the paper. Therefore, we also report on a new benchmark, Early Stop-R, which is obtained using the model parameters shared in the project's repository by Rice et al. Also note that for CIFAR-100 Early Stop uses the PreActResNet-18 architecture instead of WideResNet.

We observe that training with repulsive prototypes yields significantly higher accuracy on natural samples (71.7%) compared with adversarial training methods, where the maximum is achieved by Madry et al. [162] (59.8%). Moreover, competitive robustness (20.5%) with adversarial training (22.6%) can be observed, at almost no increase in training epochs (+6). The results reported for Early Stop by Rice et al. show 7.6% more robustness than training with repulsive prototypes, at the cost of losing 17% accuracy on natural samples. A similar result can be observed for CIFAR-10, which indicates that training with repulsive prototypes trades less accuracy on natural samples, at the cost of a modest contraction in robustness.

Similarly to CIFAR-10, we present in Figure 6.3 the robustness curve obtained by testing

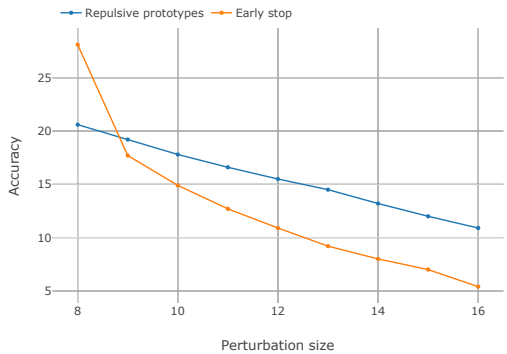


Figure 6.3: Robustness curves for CIFAR-100, obtained by testing with PGD-20, and various perturbation sizes (ϵ).

with different perturbation sizes. We compare the results with the Early Stop-R model, which uses the final parameters published by Rice et al.. We observe that, as for CIFAR-10, training with repulsive prototypes yields models resilient to large perturbation sizes, preserving more than half of the initial robustness when increasing the perturbation by a factor of two.

6.4.4 Black-box evaluation

Besides the white-box threat model investigated above, we evaluate the models in a black-box scenario. In particular, we use the transferability attack, in which an attacker trains a substitute model and uses it to craft adversarial examples.

Black-box attacks are used to evaluate the model's robustness, but also to detect if the defences employed give a false sense of security – e.g., due to obfuscating gradients [15]. Since training with repulsive prototypes does not add any transformation or randomisation which may have adverse effects (such as gradient obfuscation), we expect the defence to behave similarly to adversarial training – a defence known to have no side effects. Therefore, we compare transferability on repulsive prototypes with transferability on adversarially trained models.

Su et al. [264] showed that the architecture can impact transferability. Particularly when the network's building blocks are different (e.g., between the Inception architecture which uses different filter sizes and ResNet which uses invariant filter sizes and residual connections), robustness has higher variance. However, when the building blocks are the same, but the depth of the network increases (e.g., ResNet-50 vs. ResNet-101), robustness has smaller variance.

Therefore, since all the models from this paper use a variant of the ResNet architecture, we use a substitute model based on it. We also assume that an attacker has access to the training data set and does not need to apply data augmentation [198]. For crafting adversarial examples we use the Regular model from Tables 6.2 and 6.3. For testing, we use the PGD attack with $n = 20$ and compare with the Early Stop and Early Stop-R models.

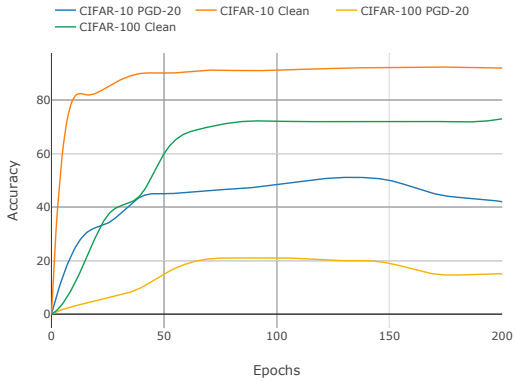


Figure 6.4: Overfitting in training with repulsive prototypes.

Table 6.4: Robustness against transferred adversarial examples. The models are [1]ResNet-18, [2]WideResNet-34-10, [3]PreActResNet-18

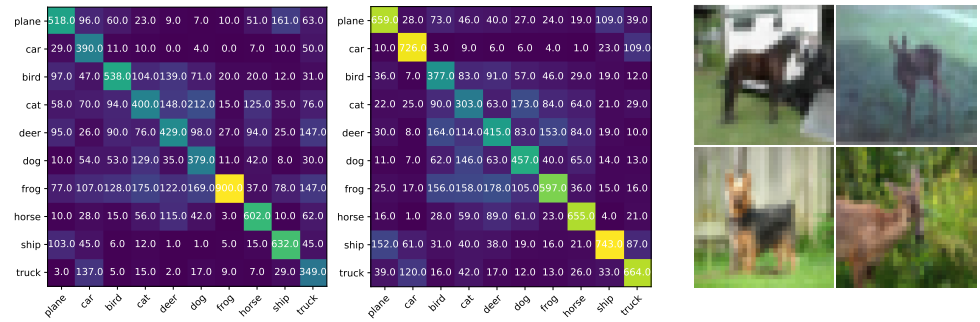
Source	Target	CIFAR-10	CIFAR-100
Regular[1]	Repulsive[1]	72.5	36.2
Regular[1]	Early Stop[2]	75.1	-
Regular[1]	Early Stop-R[3]	-	37.4

The results for both CIFAR-10 and CIFAR-100 are presented in Table 6.4. We observe that (i) robustness against black-box attacks is higher than robustness against white-box attacks, which indicates that both defences have no adverse side effects such as obfuscated gradients, and (ii) both models achieve similar robustness, consistent with the results from Tables 6.2 and 6.3. Since for CIFAR-10 Early Stop uses a more complex architecture, we expect the model to also have higher robustness (which corresponds with the results from Su et al. [264], and the higher gap in Table 6.4).

However, since the target models have different architectures and loss functions, we expect them to also have distinct internal representations. Therefore, the transferability results should have higher variance than those from Tables 6.2 and 6.3. Yet the results from Table 6.4 suggest that the target models have similar failing modes, and raise the question if some samples are more sensitive to perturbations than others, and if the samples are common between the target models. An enquiry follows in the next section.

6.5 Discussion

Firstly, we investigate if robustness is linked to properties of the testing data set, or of certain samples. Following the observation from the last section – that models trained with repulsive prototypes and with adversarial examples behave similarly against black-box attacks – we plot the confusion matrices for adversarial examples on CIFAR-10, for the



(a) Repulsive. (b) Early Stop. (c) Misclassified and nearest correctly classified samples.

Figure 6.5: Adversarial confusion matrices for the (a) Early Stop, and for the (b) Repulsive models in Table 6.2. Figure (c) shows misclassified samples (left) and the nearest correctly classified samples in the representation space (right).

Repulsive and Early Stop models (Figures 6.5b and 6.5a).

We observe that 60% of the top-1 misclassified classes are the same for both models (e.g., planes misclassified as ships). This percentage increases to 90% when we judge the top-2 classes (e.g., planes misclassified as ships or birds). We also analysed the overlap between misclassified adversarial examples by the two models and found that over 65% of the samples were common. This result indicates that some samples may be more sensitive to perturbations. For the Repulsive model, it indicates that for some samples training fails to provide intra-class compactness. When perturbed, these samples are easier to move to incorrect regions.

In order to further investigate this phenomenon, we performed random sampling on the misclassified examples for manual inspection. For all samples, we also extracted the closest examples from the predicted (wrong) class. Two such examples are displayed in Figure 6.5c, where the pictures on the left are the incorrectly classified examples and the ones on the right are the closest examples in the predicted class. We observe that both examples have many common characteristics with the closest sample in the incorrect classification regions. Similar results could be observed for other samples (suppressed due to space constraints). Previously, Jo and Bengio [122] showed that neural networks have a tendency to learn surface regularities rather than higher-level abstractions. Our initial investigation indicates that samples with similar surface regularities are also more sensitive to adversarial perturbations, even if the models using them are trained with distinct loss functions. A deeper investigation into this phenomenon is planned for future work.

Secondly, we note that training with repulsive prototypes for adversarial robustness is more sensitive than pursuing the highest accuracy on natural samples. This is a consequence of the loss function (Eq. 6.3) which measures a distance, and taking large steps towards minimising it may push samples closer to the classification boundaries, rather than closer to the prototypes. Larger steps are not relevant for natural samples, but are important for robustness. In order to alleviate this effect we used smaller learning rates and early stopping. However, this also means that models trained with repulsive prototypes are

prone to overfitting for robustness. Figure 6.4 shows the behaviour of the Repulsive models trained on both data sets studied. We observe that, while the models are stable on natural samples, they are prone to overfitting against adversarial examples. Moreover, the models seem likely to overfit faster when trained with repulsive prototypes than with adversarial training [217].

Lastly, we note that prototypes designed prior to training can also embed other properties in the output space. For example, Mettes et al. used word2vec [177] to designed prototypes. Adding more structure to the output space may lead to higher abstractions – e.g., to compositionality, as in word2vec – and it is an interesting avenue for future work.

6.6 Conclusions and future research

We introduce deep repulsive prototypes for adversarial robustness – a training method which partitions the output space prior to training into prototypes with large class separation, and train models to preserve it. Repulsive prototypes help models to gain robustness competitive to adversarial training, while removing the need to generate adversarial examples. Moreover, models trained with repulsive prototypes are less sensitive to large perturbations and trade less accuracy on natural samples for robustness.

Our results indicate that the output space size is important for robustness, and that test samples with similar surface regularities are more sensitive to adversarial perturbations. For future work we propose to search for better ways to design prototypes for robustness, which may embed other properties to the output space than large inter-class separation. Moreover, we plan to further investigate if mis-classified samples present similar surface regularities with samples in the predicted class, and find ways to remove the tendency of neural networks to rely on surface regularities.

7

Learning to Learn from Mistakes: Robust Optimisation for Adversarial Noise

In this chapter we aim to overcome the impact of adversarial training on training time using meta- and transfer-learning. Towards this goal, we train robust DL models in low data regimes, and transfer adversarial knowledge to new models. In particular, we train a meta-optimiser which learns to robustly optimise a model using adversarial examples, and it is able to transfer the knowledge learned to new models without the need to generate new adversarial examples. Experimental results show the meta-optimiser is consistent across different architectures and data sets, suggesting it is possible to learn and transfer knowledge about adversarial vulnerabilities.

This chapter This chapter has been published as *A. Serban, E.Poll, J. Visser, Learning to Learn from Mistakes: Robust Optimization for Adversarial Noise, International Conference on Artificial Neural Networks* [243].

7.1 Introduction

As mentioned in Section 5.4, although many defences against adversarial examples have been proposed, most solutions overfit on the training data and behave poorly against data outside this distribution [217, 309]. Theoretical investigations suggest these results are expected because training robust models requires more data [61], more computational resources [39] or accepting a trade off between accuracy and robustness [277]. Moreover, solutions to one vulnerability have a negative impact on others [119].

On a different path, designing ML algorithms capable to rapidly adapt to changes in the operational environment, to adapt to distribution shifts or capable to learn from few samples is an active research field. Particularly, the field of meta-learning investigate optimisation algorithms learned from scratch for faster training [11], with less resources [215] and for fast adaptation [76].

In this chapter, we show that meta-learning algorithms can be used to extract knowledge from a model's vulnerability to adversarial examples and transfer it to new models. Towards this goal, we train a meta-optimiser to learn how to robustly optimise other models using adversarial training. Later, when asked to optimise new models without seeing adversarial examples, the trained meta-optimiser can do it robustly. This process is analogous to learning a regularisation term for adversarial examples, instead of manually designing one. The experimental results suggest a broader horizon, in which algorithms learn how to automatically repair or treat vulnerabilities without explicit human design.

This chapter is organized as follows. In Section 7.2 we introduce related work. Section 7.3 formalizes meta-learning and the adversarial training problems, and gives implementation details. Section 7.4 presents experimental results on two distinct data sets, followed by a discussion in Section 7.5 and conclusions in Section 7.6.

7.2 Background and related work

Until recently, solving the outer minimisation objective from Eq. 5.8, page 79, relied on static, hand designed algorithms, such as stochastic gradient descent or ADAM. [132]. The literature surrounding this topic is focused on tailoring update rules for specific classes of problems. This line of research is driven by the no free lunch theorem of optimisation [295] which states that, on average, in combinatorial optimisation no algorithm can do better than a random strategy; suggesting that designing specific algorithms for a class of problems is the only way to improve performance.

Recent advancements in the field of meta-learning have taken a different approach, posing the problem of algorithm design dynamically and modelling it with neural networks [11], or as a reinforcement learning (RL) problem [149]. In both cases, the algorithms show empirically faster convergence and the ability to adapt to different tasks.

For example, in Andrychowicz et al. [11] the hand designed update rules are replaced with a parametrized function modelled with a recurrent neural network (RNN). During training, both the *optimiser* and the *optimisee* parameters are adjusted. The optimisation algorithm design now becomes a learning problem, allowing to specify constraints through data examples. Their results suggest the optimiser performs favourably against state-of-the-art optimisation methods and shows a high adaptability between different tasks.

The method has roots in the previous research of Schmidhuber [234], where RNNs were designed to update their own weights, and in Bengio et al. [29], which designed parametric update rules for neural networks. The results of back-propagation from one network were only later fed to a second network which learns to update the first [305]. Andrychowicz et al. [11] build on previous work using a different learning architecture. Similarly, Ravi and Larochelle [215] used this paradigm to train neural networks in a few shot regime, and Santoro et al. [227] augmented it with memory networks. Meta-learning has also shown promising results in training algorithms for fast adaptation [76].

Instead of using RNNs, Li and Malik [149] formulate the optimisation problem as a RL problem where the reward signal represents successful minimisation, and train an agent using guided policy search. Later, the authors refine their method for training neural networks [150]. In both cases the agent learns to optimise faster than hand designed algorithms and exhibits better stability.

Recent research in adversarial examples has also tackled the need to decrease training resources by either accumulating perturbations [254, 298] or by restricting back-propagation to some layers of the model [306]. While the latter method requires forward and backward passes, the former reduces the need to do backward passes in order to generate adversarial examples. Adversarial training has also been used in meta-learning, but with the goal of increasing the algorithm's robustness against adversarial examples [90, 303] and not to abstract or transfer information from adversarial training.

7.3 Meta adversarial learning

Meta-learning frames the learning problem at two levels: acquiring higher level (meta) information about the optimisation landscape and applying it to optimise one task. In this chapter, we are interested to learn robust update rules and transfer this knowledge to new tasks without additional constraints. We focus on training robust ML models through adversarial training, which is one of the most effective defences against adversarial examples [162]. Because generating adversarial examples during training is time consuming, especially for iterative procedures and can only provide robustness for the inputs used during training [309], through meta-learning we learn to optimise robustly without explicit regularisation terms and transfer the knowledge to new tasks, without the need to generate new adversarial examples.

At a high level, adversarial training discourages *local* sensitive behaviour – in the vicinity of each input in the training set – by guiding the model to behave constantly in regions surrounding the training data. The regions are defined by a chosen uncertainty set (as in Eq. 5.8, page 79). This procedure is equivalent to adding a prior that defines local constancy, for each model we want to train. In most cases, specifying this prior is not trivial and requires the design of new regularisation methods [180], new loss functions [296] or new training procedures [178]. Here, we take a different approach and try to learn a regularisation term automatically using meta-learning. During the meta-knowledge acquisition phase, the meta-optimiser learns to perform the updates robustly using adversarial training. Later, the knowledge acquired is transferred to new models using the meta-optimiser to train new models, without generating adversarial examples. In the next paragraphs we describe the meta-optimiser, some implementation details and the adversarial training

procedure.

Learning to optimise. The learning function defined in Section 5.2, $f(\cdot)$, is parametrised with a set of parameters θ. Upon seeing new data, we update the parameters in order to minimise the prediction errors. The update consists in moving one step in the opposite direction of the gradient:

$$\theta_{t+1} = \theta_t - \mu \nabla_{\theta_t} l(\cdot), \tag{7.1}$$

where μ (the learning rate) determines the size of the step. Different choices of μ or ways to automatically adapt it results in different optimisation algorithms such as stochastic gradient descent or ADAM [132].

In order to avoid overfitting or impose additional constraints such as constancy around inputs, it is common to add a regularisation term to the loss function which will be back-propagated and reflected on all parameter updates. Instead of looking for regularisation terms manually, we use a method to automatically learn robust update steps with regularisation included.

As discussed in Section 7.2, a parametrised update rule has been previously represented with a RNN [11, 215] or as a RL problem [149]. In this paper, we follow an approach similar to [11] and model the update rule with a RNN with long short-term memory (LSTM) cells:

$$\theta_t = \theta_{t-1} + c_t,$$

where :

$$c_t = f_t c_{t-1} + i_{t-1} \tilde{c}_t, \tag{7.2}$$

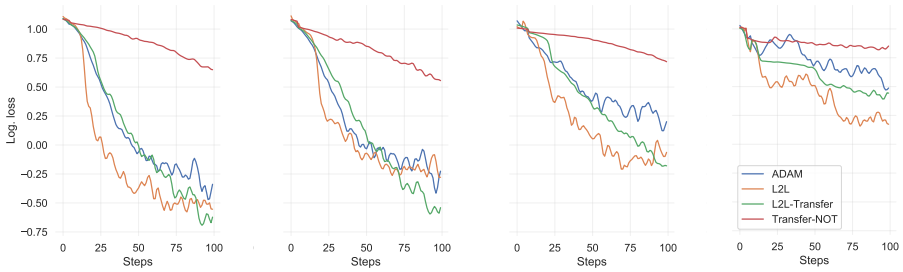
is the output of an LSTM network m with input $\nabla_{\theta_t}(l(\cdot))$:

$$\begin{bmatrix} c_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi), \tag{7.3}$$

and ϕ are the LSTM's parameters. In [215], the authors consider each term in Eq. 7.2 equivalent to each term in Eq. 7.1 – e.g., $f_t = 1$, $c_{t-1} = \theta_t$ – and disentangle the internal state of the LSTM, h_t, with special terms for individual updates of f_t and i_t. This type of inductive bias brings benefits in some cases and will be further discussed in Section 7.5. However, in this paper we try to avoid such biases whenever possible.

Parameters Sharing and Gradient Preprocessing. In order to limit the number of parameters of the optimiser, we follow a procedure similar to [11, 215] in which for each parameter of the function we want to optimise, $\theta_{i:n}$, we keep an equivalent internal state of the optimise, $h_{i:n}$, but share the weights ϕ between all states. This procedure allows a more compact optimiser to be used and makes the update rule dependent only on its respective past representation, thus being able to simulate hand designed optimisation features such as momentum.

Moreover, since gradient coordinates can take very distinct values, we apply a common normalization step in which the gradients are scaled and the information about their



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$ (d) $\epsilon = 0.3$

Figure 7.1: The loss landscape when training a neural network on the MNIST data set, perturbed with FGSM and different perturbation sizes (ϵ). The meta-optimiser is trained with adversarial examples – label L2L – transferred to a scenario where training is performed with normal and adversarial data, but tested with adversarial examples – label L2L-Transfer – and compared with a meta-optimiser trained with normal data and transferred to adversarial settings – label Transfer-NOT – and with ADAM. Best seen in colour.

magnitude and their direction is separated:

$$\nabla \longrightarrow \begin{cases} \left(\frac{\log(|\nabla|)}{p}, \text{sign}(\nabla)\right) & \text{if } |\nabla| \geq e^{-p} \\ (-1, e^{p}\nabla) & \text{otherwise.} \end{cases} \tag{7.4}$$

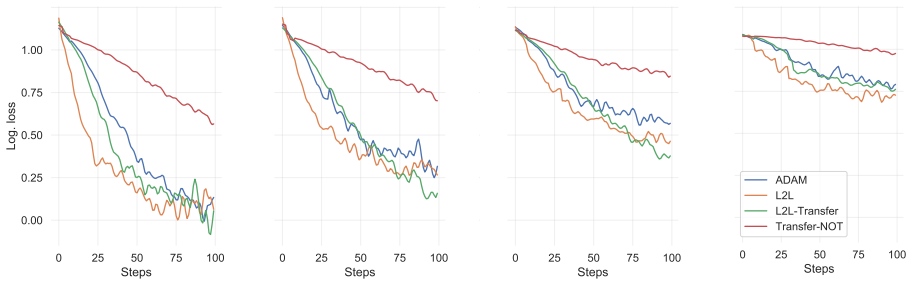
We experiment with different values for p by grid search and observe that increasing the size of p yields better results when the perturbations are larger. However, for consistency, we use $p = 10$ for all experiments.

The meta-optimiser's parameters are updated using an equivalent to Eq. 7.1. Since its inputs are based on the gradient information of the function to be optimised (the optimisee), the updates will require second order information about it (taking the gradient of the gradient). This information is commonly used for meta-learning – e.g., in [76, 291] – and will be further discussed in Section 7.5. However, in this paper only first order information is used, corresponding to limiting the propagation of the optimiser's gradient on the optimisee parameters (or stopping the gradient flow in the computational graph).

7.4 Empirical evaluation

In all experiments the optimiser consists of a two-layer LSTM network with a hidden state size of twenty. We compare the results on training two types of neural networks on two distinct data sets with the adaptive optimiser ADAM.

We focus on two experiments related to training neural networks, as in prior work on meta-learning [11, 149, 150, 215]. More experiments with minimising other functions – e.g., logistic regression – and an integration with the Cleverhans framework are available in the project's repository. In all cases, an optimiser is trained using normal and adversarial examples on a data set and tested by training a robust optimisee without generating adversarial examples. Several perturbation sizes are analysed, as introduced below.



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$ (d) $\epsilon = 0.3$

Figure 7.2: The loss landscape when training a neural network on the MNIST data set perturbed with the PGD method and different perturbation sizes (ϵ). The legend is detailed in the caption of Figure 7.1.

7.4.1 MNIST

We begin by training a small, fully connected, neural network with 20 units and ReLU activation on the MNIST data set. The perturbations take different values in the set $\epsilon \in \{0.05, 0.1, 0.2, 0.3\}$ for both attacks introduced earlier (Eq. 5.4) and Eq. 5.6). We experiment with different learning rates by grid search and find the best to be 0.001 for the meta-optimiser. Training is performed using the common cross entropy loss function, with a batch size of 128. We shuffle the training data set (consisting of 60.000 examples) and divide in two parts equally: the first is used to train a meta-optimiser using both normal data and adversarial examples and the second is used to test its performance while training with normal data and testing with perturbed data. Each experiment ran for 100 steps. The results are illustrated in Figures 7.1 and 7.2. All experiments are done using $\alpha = 0.5$ in Eq. 5.7, page 79, during training and $\alpha = 0.0$ during the meta-optimiser transfer phase, as first introduced in [91]. In addition to ADAM's performance compared to the meta-optimiser, we evaluate the performance of the meta-optimiser during training and the performance of training a meta-optimiser using $\alpha = 1$ and testing with $\alpha = 0$ (L2L and Transfer-NOT labels in Figures 7.1 and 7.2). Figure 7.1 illustrates the results from generating adversarial examples using the FGSM method (Eq. 5.4, page 78).

In all cases, the meta-optimiser is able to transfer the information learned during training and has comparable performance to ADAM (in some cases performing better). We remind that during testing the optimiser uses normal data, but the plots are generated by feeding adversarial perturbed data to the optimisee. This implies that the meta-optimiser proposes update rules which lead to smooth surfaces around the tested inputs. Moreover, it is able to learn a robust regularisation term during training and transfer it to new tasks without the need to generate new data. Also, the trained meta-optimiser exhibits more stable behaviour. These results bring evidence that adversarial training leads to more interpretable gradients [277].

When the optimiser is trained only with normal examples, but used to optimise the model using adversarial examples – Transfer-NOT label in Figure 7.1 – its performance decreases significantly. This result implies that a meta-optimiser is domain specific and

Table 7.1: MNIST PGD results, $\epsilon = 8$.

Run	Ep.	Natural	PGD 20	PGD 100
Regular	100	99.0	0	0
L2L	100	98.6	92.0	91.2
L2L-Transfer	100	97.8	91.8	90.9
Madry[2]	100	98.8	92.9	91.8

does not have the general behaviour of ADAM, an observations which will be further discussed in Section 7.5.

In Figure 7.2 we illustrate the results from running similar experiments, but generate adversarial examples using the PGD method from Eq. 5.6, page 78. Training with PGD is generally performed only using the perturbed examples (corresponding to $\alpha = 0$ in Eq. 5.7, page 79), as in the original paper [162]. We take a similar approach in this paper.

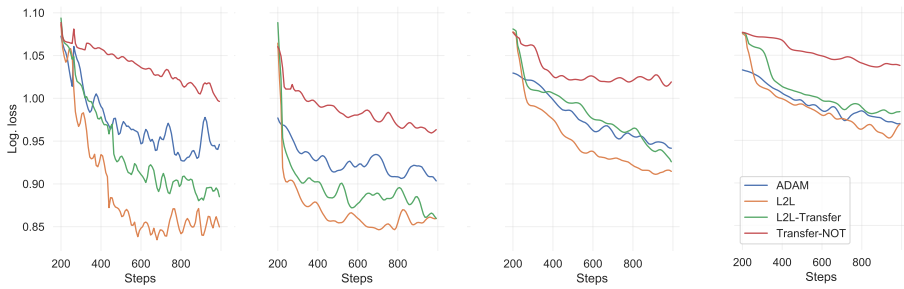
The results are consistent with the FGSM method, although the gap between ADAM and the transferred meta-optimiser is smaller. A constant decrease in performance is also observed, possibly corresponding to the decrease in performance specific to adversarial training [277]. Nevertheless, the results are consistent and bring evidence that the meta-optimiser is able to learn robust update rules.

The accuracy results – similar to the one in Table 6.2 – are presented in Table 7.1. The optimisation ran for 60 epochs, using cross-validation (as described earlier). Here, we observe that the meta-learning algorithm is able to preserve similar accuracy against the PGD attack, for different number of iterations. Only a small decrease is observed, which may once again correspond to the decrease in performance specific to adversarial training [277]. This result is expected, since unlike the method presented in the previous chapter, we now rely on adversarial training to extract knowledge during the meta-learning phase.

7.4.2 CIFAR-10

We present the results from training a model using both convolution and fully connected layers on the CIFAR-10 data set. The network consists of three convolutional layers with kernel size 3, a fully connected layer with 32 hidden units and a logits layer of size 10. All activation functions are ReLU, the loss is cross-entropy and batch normalization is used in the convolutional layers. The meta-optimiser is trained using a learning rate of 0.001.

Since there are striking differences between convolutional and linear layers, we use two sets of parameters for the meta-optimiser – one for optimising all convolutional layers and one for the linear layers. Moreover, since the CIFAR-10 problem is more difficult, we train the optimisee for 1000 steps and present the results in Figure 7.3 for perturbations generated with FGSM and in Figure 7.4 for PGD. The evaluation was performed as earlier, using 2-fold cross validation for training a meta-optimiser in adversarial settings and transfer it to training a model in normal settings, but tested with adversarial examples.



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$ (d) $\epsilon = 0.3$

Figure 7.3: The loss landscape when training a neural network on the CIFAR-10 data set, perturbed with the FGSM method and different perturbation sizes (ϵ). The legend is detailed in the caption of Figure 7.1.

We observe that in the case of FGSM, the transferred meta-optimiser (label L2L-Transfer, Figure 7.3) exhibits similar behavior as in the MNIST experiments: it has similar and sometimes better performance than ADAM. We remind that, in this case, no adversarial examples are used during training. The meta-optimiser trained normally, but tested with adversarial examples (Transfer-NOT label, Figure 7.3) performs visibly worse, which strengthens the observation that meta-learning optimisation is domain specific.

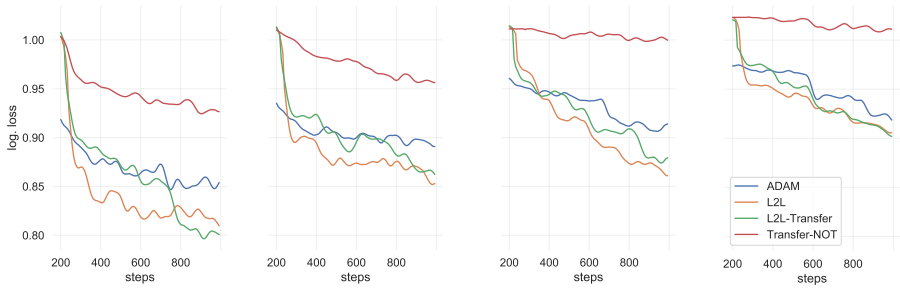
Figure 7.4 shows results from running the same experiment using perturbations generated with PGD, with a number of 7 steps, as in the original paper [162]. In all cases, the loss improvements are small, although the meta-optimiser exhibits better performance than ADAM both during training and testing. However, the improvements in training time are significant since after training an adversarial meta-optimiser, it can be applied to different models without the need to execute the PGD steps for each batch of data.

Table 7.2 shows the final results after training for 60 epochs. Once again, we observe that the meta-optimiser offers robustness competitive with adversarial training, while removing the need to generate adversarial samples.

7.5 Discussion

Typically used to rapidly adapt to new tasks or generalize outside the i.i.d assumption, meta-learning algorithms show promising results to reduce the training samples needed for adversarial training. The results presented in this paper suggest these algorithms can be used in the future to build adversarial defences with less computational resources and capable to adapt to new data.

We hereby note some weaknesses discovered during the process. Although capable of achieving better performance than hand crafted optimisers [11, 150] and, as discussed in Section 7.4, showing promising results in transferring information about adversarial examples, meta learning algorithms still suffer from broader generalization. In particular, trained optimisers can not generalize to different activation functions, or between architectures with noticeable differences [291]. This means that an optimiser trained for ReLU can not be used for sigmoid (or other) activation functions. Moreover, if the meta-optimiser is not



(a) $\epsilon = 0.05$ (b) $\epsilon = 0.1$ (c) $\epsilon = 0.2$ (d) $\epsilon = 0.3$

Figure 7.4: The loss landscape when training a neural network on the CIFAR-10 data set, perturbed with the PGD method and different perturbation sizes (ϵ). The legend is detailed in the caption of Figure 7.1.

Table 7.2: CIFAR PGD results, $\epsilon = 8$.

Run	Ep.	Natural	PGD 20	PGD 100
Regular	100	83.3	0	0
L2L	100	81.9	43.4	41.9
L2L-Transfer	100	81.6	42.9	41.5
Madry[2]	100	82.5	43.6	42.2

trained with specific data that will be later used, it does not exhibit general behaviour. For example, if the meta-optimiser does not use any adversarial examples during training, but it encounters such examples during testing, it faces difficulties. This behaviour is illustrated in the figures above with the label Transfer-NOT.

Second order information (taking the gradient of the gradient, as introduced in Section 7.3) was not used in this paper. As shown in [291], this information can help the meta-optimiser better generalize and induce more stable behaviour. However, it also introduces more complexity. Analysing the trade-off between the optimiser's complexity and its ability to learn and transfer knowledge related to adversarial vulnerabilities is left for future research.

7.6 Conclusions and future research

We introduce a method to learn how to optimise ML models robust to adversarial examples, in low data regimes. Instead of specifying custom regularisation terms, they are learned automatically by an adaptive optimiser. Acquiring meta information about the optimisation landscape under adversarial constraints allows the optimiser to reuse it for new tasks.

For future research, we propose to train the meta-optimiser concomitantly with different perturbation types – e.g., l_1, l_2-norm – and test if the optimiser can learn to robustly optimise under all constraints. Other perturbations, such as naturally occurring perturbations [103]

can also be included. Another research direction is to use the meta-optimiser to refine a trained model and evaluate if it is possible to robustly regularise it with less data.

8

Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks

In this chapter we tackle the problem of robust decision making for autonomous systems under uncertainty. This problem is relevant for any autonomous system for which the environment is not completely known, which is the case in most real-world scenario. In particular, we study strategy synthesis for partially observable Markov decision processes (POMDPs) – i.e., determine strategies that provably adhere to probabilistic temporal logic constraints. This problem is known to be computationally intractable. We propose a novel method to solve this problem by combining techniques from machine learning and formal verification. Firstly, we train a deep learning model to encode POMDP strategies. The deep learning model accounts for memory-based decisions without the need to expand the full belief space of a POMDP. Secondly, we restrict the deep learning based strategy to represent a finite-memory strategy and implement it on a specific POMDP. For the resulting finite Markov chain, formal verification techniques provide provable guarantees for temporal logic specifications.

This chapter has been published as *S. Carr, N. Jansen, R. Wimmer, A. Serban, B. Becker, U. Topcu, Counterexample-Guided Strategy Improvement for POMDPs Using Recurrent Neural Networks, International Joint Conference on Artificial Intelligence, 2019* [46].

8.1 Introduction

Autonomous agents that plan decisions under uncertainty and incomplete information can be mathematically represented as partially observable Markov decision processes (POMDPs). In this setting, when an agent makes decisions within an environment, it obtains *observations* and infers the likelihood of the system being in a certain state (also known as the *belief* state). POMDPs are effective in modelling a number of real-world applications – e.g., [126, 271] – and are the standard model for decision making in autonomous systems where the environment is not completely known.

Traditional POMDP problems seek to compute a strategy that maximizes a cumulative reward over a finite horizon. However, the agent's behaviour is often required to obey more strict specifications. For example, reachability, liveness or, more generally, specifications expressed in temporal logic (e.g., LTL [207]) describe tasks that cannot be expressed using reward functions [153]. For example, one would like to verify upfront that an autonomous agent has low probability of reaching a bad state and guarantee safe operation. Therefore, a robust decision making algorithm must satisfy with high probability the specifications.

Strategy synthesis for POMDPs is difficult, both from a theoretical and practical perspective. For infinite- or indefinite-horizon problems, computing an optimal strategy is undecidable [161]. Optimal action choices depend on the whole history of observations and actions, thus requiring an infinite amount of memory. When restricting the specifications to maximize accumulated rewards over a finite horizon and limiting the available memory, computing an optimal strategy is PSPACE-complete [195]. This problem is, practically, intractable even for small instances [176]. Moreover, even when strategies are restricted to be *memoryless*, finding an optimal strategy within this set is still NP-hard [280]. For more general specifications like LTL properties, synthesis of strategies with limited memory is even harder, namely EXPTIME-complete [48]).

The intractable nature of these problems gave rise to approximate [101], point-based [205], or Monte-Carlo-based [258] methods. However, none of these approaches provides guarantees for given temporal logic specifications. The tool PRISM-POMDP [190] does so by approximating the belief space into a fully observable belief MDP, but is restricted to small examples. Other techniques – such as those using satisfiability modulo theory solvers over a bounded belief space [285] or a label-based simulation over sets of belief models [98] – are also only restricted to small examples.

Although strategy synthesis for POMDPs is difficult, a *candidate strategy* resolves non-determinism and partial observability for a POMDP, and yields a so-called induced discrete-time Markov chain (MC). For this simpler model, verification methods are capable to efficiently certify temporal logic constraints and reward specifications for billions of states [18]. Tool support is also available via probabilistic model checkers such as Storm [64].

However, there is a tension between directly synthesizing an optimal strategy and efficient verification of a candidate strategy. Here, two questions arise: (i) how to generate a suitable strategy, and (ii) how to improve a strategy if verification refutes the specification? Machine learning (ML) and formal verification techniques address these questions separately. In this chapter, we bridge methods from both fields in order to guarantee that a candidate strategy learned through ML provably satisfies temporal logic specifications.

First, we learn a randomized strategy[1] via recurrent neural networks (RNNs) [107] – which we refer to as the *strategy network*. RNNs are a good candidate for learning a strategy because they can successfully represent temporal dynamic behaviour [202]. Second, we extract a concrete (memoryless randomized) candidate strategy from the RNN and use it directly on a given POMDP, resulting in the MC induced by the POMDP and the strategy. Formal verification reveals whether specifications are satisfied or not. In the latter case, we generate a counterexample [293], which points to parts of the MC (and by extension of the POMDP), that are critical for the specification. For those critical parts, we use a linear programming (LP) approach that locally improves strategy choices (without guarantees on the global behaviour). From the improved strategy, we generate new data to retrain the RNN. We iterate that procedure until the strategy network yields satisfactory results.

While the strategies are memoryless, allowing randomisation over possible choices and relaxing determinism is often sufficient to capture necessary variability in decision-making. The intuition is that *deterministic* choices at a certain state may need to vary depending on previous decisions, thereby trading off memory. However, randomization in combination with finite memory may supersede infinite memory even more for many cases [7, 125]. We encode finite memory directly into a POMDP by extending its state space. We can then directly apply our method to create finite-state controllers (FSCs) [176].

As previously discussed, the investigated problem is undecidable for POMDPs [161] and therefore the approach is naturally incomplete. Soundness is provided, as verification yields hard guarantees on the quality of a strategy.

This chapter is organised as follows. Section 8.2 introduces the formal foundations on POMDP and related work. Section 8.3 describes the strategy synthesis procedure, followed by details on how the strategy is trained using RNNs in Section 8.3.1. The method's effectiveness is demonstrated using a selection of temporal logic examples, as well as comparing to well-known benchmarks [261] in Sect. 8.4. Conclusions and future work follow in Section 8.5.

8.2 Background and related work

Preliminaries. A probability distribution over a finite or countably infinite set X is a function $\mu : X \to [0,1] \subseteq \mathbb{R}$ with $\sum_{x \in X} \mu(x) = \mu(X) = 1$. The set of all distributions on X is $Distr(X)$. The support of a distribution μ is $\text{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$.

A *Markov decision process (MDP)* M is a tuple $M = (S, Act, \mathcal{P})$ with a finite (or countably infinite) set S of *states*, a finite set Act of *actions*, and a *transition function* $\mathcal{P} : S \times Act \to Distr(S)$. We use a reward function $r : S \times a \to \mathbb{R}$. A finite *path* π of an MDP M is a sequence of states and actions; $\text{last}(\pi)$ is the last state of π. The set of finite paths of M is $Paths^M_{fin}$. A *discrete-time Markov chain (MC)* is an MDP with $|Act(s)| = 1$ for all $s \in S$.

A *strategy* γ for and MDP M is a function $\gamma : Paths^M_{fin} \to Distr(Act)$ with $\text{supp}\big(\gamma(\pi)\big) \subseteq Act\big(\text{last}(\pi)\big)$ for all $\pi \in Paths^M_{fin}$. A strategy γ is *memoryless* if $\text{last}(\pi) = \text{last}(\pi')$ implies $\gamma(\pi) = \gamma(\pi')$ for all $\pi, \pi' \in Paths^M_{fin}$.

[1] Also referred to as stochastic strategy or policy.

For an MDP $M = (S, Act, \mathcal{P})$ and a strategy $\gamma \in \Gamma^M$, the MC induced by M and γ is given by $M^\gamma = (Paths^M_{fin}, P^\gamma)$ where:

$$P^\gamma(\pi, \pi') = \begin{cases} \mathcal{P}(\text{last}(\pi), a, s') \cdot \gamma(\pi)(a) & \text{if } \pi' = \pi a s', \\ 0 & \text{otherwise.} \end{cases} \tag{8.1}$$

A *partially observable Markov decision process (POMDP)* is a tuple $\mathcal{M} = (M, Z, O)$, with $M = (S, Act, \mathcal{P})$ the *underlying MDP of* $\mathcal{M}$, Z a finite set of observations and $O : S \to Z$ the *observation function*. The set of all finite observation-action sequences for a POMDP $\mathcal{M}$ is denoted by $ObsSeq^{\mathcal{M}}_{fin}$. An *observation-based strategy* for a POMDP $\mathcal{M}$ is a function $\gamma : ObsSeq^{\mathcal{M}}_{fin} \to Distr(Act)$ such that $\text{supp}\big(\gamma(O(\pi))\big) \subseteq Act\big(\text{last}(\pi)\big)$ for all $\pi \in Paths^M_{fin}$. $\Gamma^{\mathcal{M}}_z$ is the set of observation-based strategies for $\mathcal{M}$.

A *memoryless* observation-based strategy $\gamma \in \Gamma^{\mathcal{M}}_z$ is analogous to a memoryless MDP strategy, formally we simplify to $\gamma : Z \to Distr(Act)$, i.e.,we decide based on the current observation only. Similarly, POMDP together with a strategy yields an induced MC as in Def. 8.1, resolving all nondeterminism and partial observability. A general POMDP strategy can be represented by *infinite-state controllers*. Strategies are often restricted to finite memory; this amounts to using finite-state controllers (FSCs) [176].

A k-FSC for a POMDP is a tuple $\mathcal{A} = (N, n_I, \gamma, \delta)$ where N is a *finite set* of k memory nodes, $n_I \in N$ is the initial memory node, γ is the action mapping $\gamma : N \times Z \to Distr(Act)$ and δ is the memory update $\delta : N \times Z \times Act \to N$. Let $\gamma_{\mathcal{A}} \in \Gamma^{\mathcal{M}}_z$ denote the observation-based strategy represented by the FSC $\mathcal{A}$. The product $\mathcal{M} \times \mathcal{A}$ of a POMDP and a k-FSC yields a (larger) "flat" POMDP where the memory update is directly encoded into the state space [125]. The action mapping γ is left out of the product. A memoryless strategy $\gamma \in \Gamma^{\mathcal{M}\times\mathcal{A}}_z$ then determines the action mapping and can be projected to the finite-memory strategy $\gamma_{\mathcal{A}} \in \Gamma^{\mathcal{M}}_z$.

8

Specifications. We consider linear-time temporal logic (LTL) properties [207]. For a set of atomic propositions AP, which are either satisfied or violated by a state, and $a \in AP$, the set of LTL formulas is given by:

$$\Psi a \mid (\Psi \wedge \Psi) \mid \neg\Psi \mid \bigcirc \Psi \mid \Box \Psi \mid (\Psi \, U \, \Psi). \tag{8.2}$$

Intuitively, a path π satisfies the proposition a if its first state does; $(\psi_1 \wedge \psi_2)$ is satisfied, if π satisfies both ψ_1 and ψ_2; $\neg\psi$ is true on π if ψ is not satisfied. The formula $\bigcirc\psi$ holds on π if the subpath starting at the second state of π satisfies ψ. The path π satisfies $\Box\psi$ if all suffixes of π satisfy ψ. Finally, π satisfies $(\psi_1 \, U \, \psi_2)$ if there is a suffix of π that satisfies ψ_2 and all longer suffixes satisfy ψ_1. $\Diamond\psi$ abbreviates $(\text{true} \, U \, \psi)$.

For POMDPs, one wants to synthesize a strategy such that the probability of satisfying an LTL-property respects a given bound, denoted $\varphi = \mathbb{P}_{\sim\lambda}(\psi)$ for $\sim \in \{<, \leq, \geq, >\}$ and $\lambda \in [0, 1]$. In addition, *undiscounted expected reward properties* $\varphi = \mathbb{E}_{\sim\lambda}(\Diamond a)$ require that the expected accumulated cost until reaching a state satisfying a respects $\lambda \in \mathbb{R}_{\geq 0}$.

If φ (either LTL or expected reward specification) is satisfied in a (PO)MDP $\mathcal{M}$ under γ, we write $\mathcal{M}^\gamma \models \varphi$, that is, the specification is satisfied in the induced MC, see Eq. 8.1. While determining an appropriate strategy is still efficient for MDPs, this problem is in

general undecidable for POMDPs [49]. In particular, for MDPs, to check the satisfaction of a general LTL specification one needs memory. Typically, tools like PRISM [144] compute the product of the MDP and a deterministic Rabin automaton. In this product, reachability of so-called accepting end-components ensures the satisfaction of the LTL property. This reachability probability can be determined in polynomial time. PRISM-POMDP [190] handles the problem similarly for POMDPs, but note that a strategy needs memory not only for the LTL specification but also for observation dependencies.

Finally, given a (candidate) strategy γ, checking whether $\mathcal{M}^\gamma \models \varphi$ holds can be done both for MDPs and POMDPs in polynomial time [18].

Related Work. Besides the publications mentioned in Section 8.1, previous work has used recurrent neural network (RNN) to synthesize strategies for POMDPs. These methods fall within the policy gradient class of algorithms specific to reinforcement learning (RL) [266]. In this setting, the strategy is parameterized and updated by performing gradient ascent on the error function (typically chosen to maximize the discounted reward).

In order to cope with arbitrary memory in POMDPs, policy gradients methods need some notion of memory. RNNs are suitable for this task because (1) they are differentiable end-to-end and (2) they are designed to exhibit dynamic temporal behavior. Indeed, [292] were the first to employ a RNN to learn (finite-memory) strategies for POMDPs. In particular, the authors used a long short-term memory (LSTM) architecture which is able to leverage both long and short term events in the past.

Recent progress in deep learning (DL) enabled scaling deep neural networks (DNNs) to solve complex problems. For example, [181] developed a DNN-based Q-learning algorithm able to play video games straight from video frames, under partial observability. Instead of using RNNs, the memory problem is solved by replaying a series of frames at every step. Later, [100] added an LSTM cell to enhance the algorithm's capacity with both long and short term memory. From there on, the field rapidly moved to explore new ways of improving the memory representation [201, 211, 228]. However, even though they yield good performance on a variety of tasks, these methods do not provide guarantees on the strategies learned. In fact, it is very hard to perform any reasoning about these strategies.

8.3 Strategy synthesis

For a POMDP $\mathcal{M}$ and a specification φ, where either $\varphi = \mathbb{P}_{\sim\lambda}(\psi)$ with ψ an LTL formula, or $\varphi = \mathbb{E}_{\sim\lambda}(\Diamond\, a)$, the problem is to determine a (finite-memory) strategy $\gamma \in \Gamma_z^{\mathcal{M}}$ such that $\mathcal{M}^\gamma \models \varphi$. If such a strategy does not exist, the problem is infeasible.

The workflow of the proposed method is illustrated in Fig. 8.1: We start by training a RNN using observation-action sequences generated from an initial strategy as discussed in Section 8.3.1. The trained *strategy network* represents an observation-based strategy, taking as input an observation-action sequence and returning a distribution over actions (Section 8.2). For a POMDP $\mathcal{M}$, we use the output of the strategy network in order to resolve nondeterminism. The strategy network is thereby used to extract a *memoryless strategy* $\gamma \in \Gamma^{\mathcal{M}}$ and as a result we obtain the induced MC $\mathcal{M}^\gamma$. *Model checking* of this induced MC evaluates whether the specification φ is satisfied or not for the extracted strategy. In the

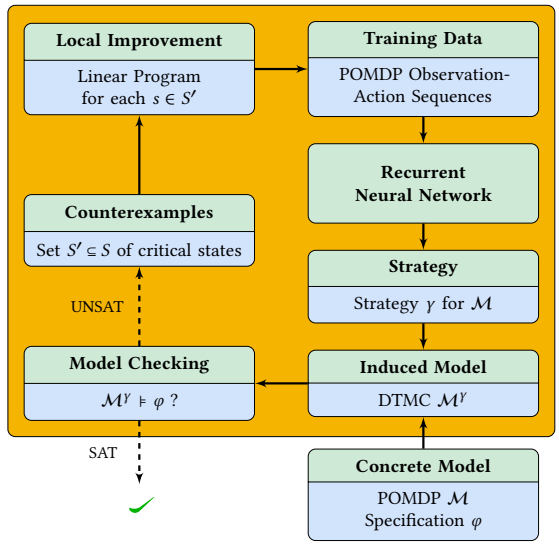


Figure 8.1: Flowchart of the RNN-based refinement loop.

former case, the synthesis procedure is finished. The extraction and evaluation is explained in Section 8.3.2.

If the specification is not satisfied, we obtain a *counterexample* highlighting critical states of the POMDP. We employ a linear programming (LP) approach that locally *improves* action choices of the current strategy at these critical states, described in Section 8.3.3. Afterwards, we retrain the RNN by generating new observation-action sequences obtained from the new strategy. We iterate this procedure until the specification is satisfied or a fixed iteration threshold is reached. For cases where we need to further improve, we use domain knowledge to create a specific memory-update function of a k-FSC $\mathcal{A}$ (Section 8.3). Then, we compute the product $\mathcal{M}' = \mathcal{M} \times \mathcal{A}$. We iterate with $\mathcal{M}'$ as starting point and thereby determine a concrete k-FSC including the action mapping.

8

8.3.1 Learning strategies with recurrent neural networks

As mentioned in Section 8.2, policy gradient algorithms are used to map observations to actions and are not well suited for POMDPs due to their inability to cope with arbitrary memory. To overcome this weakness, we make explicit use of memory using RNNs - a family of neural networks designed to exhibit dynamic temporal behaviour.

Constructing the Strategy Network. We use the LSTM architecture [107] in a similar fashion to policy gradient methods and model the output as a probability distribution on the action space (described formally by $\hat{\gamma} : ObsSeq_{fin}^{\mathcal{M}} \rightarrow Distr(Act)$). Having stochastic output units, we avoid computing gradients on the internal belief states, as it is, for example, done in [176]. Using back propagation through time, we can update the strategy during training. Thus, for a given observation-action sequence from $ObsSeq_{fin}^{\mathcal{M}}$, the model learns

a strategy $\hat{\gamma} \in \Gamma_z^{\mathcal{M}}$. The output is a discrete probability distribution over the actions Act, represented using a final softmax layer.

RNN Training. We train the RNN using a slightly modified version of sampling re-usable trajectories [129]. In particular, for a POMDP $\mathcal{M} = (M, Z, O)$ and a specification φ, instead of randomly generating observation sequences, we first compute a strategy $\gamma \in \Gamma^M$ of the underlying MDP M that satisfies φ. Then we sample uniformly over all states of the MDP and generate finite paths (of a fixed maximal length) from $Paths_{fin}^{M^\gamma}$ of the induced MC M^γ, thereby creating multiple trajectory trees. For each finite path $\pi \in Paths_{fin}^{M^\gamma}$, we generate one possible observation-action sequence $\pi_z \in ObsSeq_{fin}^{\mathcal{M}}$ such that $\pi = z_0, a_0, \ldots, a_{n-1}, z_n$ with $z_i = O(\pi[i])$, where $\pi[i]$ denotes the i-th state of π for all $1 \leq i \leq n$. We form the training data set $\mathcal{D}$ from a (problem specific) number of m observation-action sequences with observations as input and actions as output labels. Both input and output sets were processed using one-hot-encoding. To fit the RNN model, we use the Adam optimiser [132] with a cross-entropy error function.

Sampling Large Environments. In a POMDP $\mathcal{M}$ with a large state space ($|S| > 10^5$), computing the underlying MDP strategy $\gamma \in \Gamma^M$ affects the performance. In such cases, we restrict the sampling to a smaller environment that shares the observation Z and action spaces Act with $\mathcal{M}$. For example, consider a grid-world scenario with a moving obstacle that has the same underlying probabilistic movement for different problem sizes. Such a framework can provide a similar dataset regardless of the size of the grid.

8.3.2 Strategy extraction and evaluation

We first describe how to extract a memoryless strategy from the strategy network for a specific POMDP, then we formalize the extension to FSCs to account for finite memory. Finally, we explain how the strategies can be evaluated.

Given a POMDP $\mathcal{M}$, we use the trained strategy network $\hat{\gamma} : ObsSeq_{fin}^{\mathcal{M}} \rightarrow Distr(Act)$ directly as observation-based strategy. Note that the RNN is inherently a predictor for the distribution over actions and will not always deliver the same output for one input. While we always use the first prediction we obtain, one may also sample several predictions and take the average of the output distributions.

Extension to FSCs. As mentioned before, LTL specifications as well as observation-dependencies in POMDPs require memory. Consider therefore a general FSC $\mathcal{A} = (N, n_I, \gamma, \delta)$ (Section 8.2). We first predefine the memory update function δ in a problem-specific way, for instance, δ changes the memory node when an observation is repeated. Consider observation sequence $\pi_z \in ObsSeq_{fin}^{\mathcal{M}}$ with $\pi_z = z_0, a_0, \ldots, z_n$. Assume, the FSC is in memory node $n_k \in N$ at position i of π_z. We define $\delta(n_k, z_i, a_i) = n_{k+1}$, if $\pi_z[i] = (z_i, a_i)$, and there exists a $j < i$ such that $\pi_z[j] = (z_j, a_j)$ with $z_i = z_j$. Similarly, we account for specific memory choices akin to the relevant LTL specification.

Once δ has been defined, we compute a product POMDP $\mathcal{M} \times \mathcal{A}$ which creates a state space over $S \times N$. The training process is similar to the method outlined above but instead of

generating observation-action sequences from $ObsSeq^{\mathcal{M}}_{fin}$, we generate observation-node-action sequences
$(z_0, n_0), a_0, \ldots, a_{n-1}, (z_n, n_n)$ from $ObsSeq^{\mathcal{M}\times\mathcal{A}}_{fin}$. In this case, the RNN is learning the mapping of observation and memory node to the distribution over actions as an FSC strategy network: $\hat{\gamma}_{\text{FSC}} : ObsSeq^{\mathcal{M}\times\mathcal{A}}_{fin} \times N \longrightarrow Distr(Act)$

In order to extract the memoryless FSC $\mathcal{A}$ from the FSC strategy network $\hat{\gamma}_{\text{FSC}}$, we collect the predicted distributions across the product set of all possible observations $z \in Z$ and all possible memory nodes $n \in N$. From this prediction, the FSC $\mathcal{A}$ is constructed from the action mapping $\gamma(z, n) = \hat{\gamma}_{\text{FSC}}(z, n)$ and the predefined memory update function δ.

Evaluation. We assume that for POMDP $\mathcal{M} = (M, Z, O)$ and specification φ, we have a finite-memory observation-based strategy $\gamma \in \Gamma^{\mathcal{M}}$ as described above. We use the strategy γ to resolve all nondeterminism in $\mathcal{M}$, resulting in the induced MC $\mathcal{M}^{\gamma}$, see Def. 8.1. For this MC, we apply model checking, which in polynomial time reveals whether $\mathcal{M}^{\gamma} \models \varphi$. For the fixed strategy γ we extracted from the strategy network, this provides hard guarantees about the quality of γ regarding φ. As mentioned before, this strategy is only a prediction obtained from the RNN – so the guarantees necessarily do not directly carry over to the strategy network.

8.3.3 Improving the strategy

Next we describe how to compute a *local improvement* for a strategy that does not satisfy the specification. In particular, we have POMDP $\mathcal{M} = (M, Z, O)$, specification φ, and the strategy $\gamma \in \Gamma^{\mathcal{M}}$ with $\mathcal{M}^{\gamma} \not\models \varphi$. We then create diagnostics on why the specification is not satisfied.

First, without loss of generality, we assume $\varphi = \mathbb{P}_{\leq\lambda}(\psi)$. Let $\gamma(z)(a)$ denote the probability of choosing action $a \in Act$ upon observation $z \in Z$, under the strategy γ. Let $\Pr^*(s)$ denote the probability to satisfy ψ within the induced MC $\mathcal{M}^{\gamma}$. For some threshold $\lambda' \in [0, 1]$, a state $s \in S$ is *critical* iff $\Pr^*(s) > \lambda'$. We define λ' as a function $\lambda' : S \times \lambda \longrightarrow \mathbb{R}$ with respect to the threshold λ from the original specification and the state s. We define the set of critical decision under the strategy γ.

A probability $\gamma(z)(a) > 0$ according to an observation-based strategy $\gamma \in \Gamma$ is a *critical decision* iff there exist states $s, s' \in S$ with $s \in O^{-1}(z)$, $\mathcal{P}(s, a, s') > 0$, and s' is critical. Intuitively, a decision is critical if it may lead to a critical state. The set of critical decisions serves as *counterexample*, generated by the set of critical states and the strategy γ. Note that even if a specification is satisfied for γ, the sets of critical decisions and states may still be non-empty as they depend on the definition of the criticality-threshold λ'.

For each observation $z \in O$ with a critical decision, we construct an optimisation problem that minimises the number of different (critical) actions the strategy chooses per observation class.

In particular, the probabilities of action choices under γ are redistributed such that the critical choices are minimised.

$$\max_{\gamma(z)(a), a \in Act} \min_{s \in S} p_s \tag{8.3}$$

$$\text{s.t.}$$
$$\forall s \in O^{-1}(z). \quad p_s = \sum_{a \in Act} \gamma(z)(a) \cdot \sum_{s' \in S} \mathcal{P}(s,a,s') \cdot p^*(s')$$

If the objective function is zero, then we have found an observation-based strategy, as there are no choices that are inconsistent with the observations any more. Otherwise, we select a class for which at least two different actions are necessary and then we generate a new set of paths starting from the critical states. After converting these new paths into observation-action sequences, we retrain the RNN. By gathering more data from these apparently critical situations, we locally improve the quality of the strategies at those locations and gradually introduce observation-dependencies.

8.3.4 Correctness and termination

Correctness of our approach is ensured by evaluating the extracted strategy on the POMDP using model checking. As the investigated problem is undecidable for POMDPs [161], our approach is naturally *incomplete*. In order to enforce termination after finite time, we abort the refinement loop after a specified number of iterations, or as soon as the progress from one iteration to the next (in terms of the model checking results) falls below a threshold.

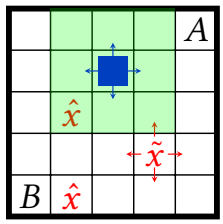
8.4 Empirical evaluation

We evaluate tthe RNN-based synthesis procedure on benchmark examples that are subject to either LTL specifications or expected cost specifications. For the former, we compare to the tool PRISM-POMDP, and for the latter we compare to PRISM-POMDP and the point-based solver SolvePOMDP [281]. Recall that, in general, a strategy over the continuous belief space induces an infinite memory strategy for POMDPs. PRISM-POMDP employs a discretization (we chose the default level of discretization) of that belief space which technically induces a finite-memory strategy. Therefore solutions from PRISM-POMDP are approximate; the tool computes an upper and lower bound on the optimum.

We selected the two solvers from different research communities because they provide the possibility for a straightforward adaption to our benchmark setting. In particular, the tools support undiscounted rewards and have a simple and similar input interface. Extended experiments with, for instance, Monte-Carlo-based methods [258] are interesting but beyond the scope of this paper.

For a fair comparison, instead of terminating the synthesis procedure once a specification is satisfied, we always iterate 10 times, where one iteration encompasses the (re-)training of the RNN, the strategy extraction, the evaluations, and the strategy improvement as in Sect. 8.3. For instance, for a specification $\varphi = \mathbb{P}_{\leq\lambda}(\psi)$, we leave the λ open and seek to compute $\mathbb{P}_{\min}(\psi)$, that is, we compute the minimal probability of satisfying ψ for a strategy that satisfies φ. We cannot guarantee to reach that optimum, but we rather improve as far as possible within the predefined 10 iterations. The notions are similar for $\mathbb{P}_{\geq\lambda}$ and $\mathbb{P}_{\max}$ as well as for expected cost measures $\mathbb{E}_{\leq\lambda}$ ($\mathbb{E}_{\geq\lambda}$) and $\mathbb{E}_{\min}$ ($\mathbb{E}_{\max}$).

We will now shortly describe our experimental setup and present detailed results for both types of examples.



(a)

Problem	$\|S\|$	$\|Act\|$	$\|Z\|$
Navigation (c)	c^4	4	256
Delivery (c)	c^2	4	256
(c)	c^2	4	256
Maze(c)	$3c+8$	4	7
Grid(c)	c^2	4	2
RockSample[4,4]	257	9	2
RockSample[5,5]	801	10	2
RockSample[7,8]	12545	13	2

(b)

Figure 8.2: (a) Example environment and (b) Benchmark metrics

Implementation and Setup. We employ the following Python toolchain to realize the full RNN-based synthesis procedure. First, we use the deep learning library Keras [54] to train the strategy network. To evaluate strategies, we employ the probabilistic model checkers PRISM (LTL) and STORM (undiscounted expected rewards).We evaluated on a 2.3 GHz machine with a 12 GB memory limit and a specified maximum computation time of 10^5 seconds.

8.4.1 Temporal logic examples

We examined three problem settings involving motion planning with LTL specifications. For each of the settings, we use a standard grid-world formulation of an agent with 4 action choices (cardinal directions of movement), see Fig. 8.2a. Inside this environment there are a set of static ($\hat{x}$) and moving ($\tilde{x}$) obstacles as well as possible target cells A and B. Each agent has a limited visibility region, indicated by the green area, and can infer its state from observations and knowledge of the environment. We define observations as Boolean functions that take as input the positions of the agent and moving obstacles. Intuitively, the functions describe the 8 possible relative positions of the obstacles with respect to the agent inside its viewing range.

1. **Navigation with moving obstacles** – an agent and a single stochastically moving obstacle. The agent task is to maximize the probability to navigate to a goal state A while not colliding with obstacles (both static and moving): $\varphi_1 = \mathbb{P}_{\max}(\neg X\ U\ A)$ with $x = \hat{x} \cup \tilde{x}$,

2. **Delivery without obstacles** – an agent and static objects (landmarks). The task is to deliver an object from A to B in as few steps as possible: $\varphi_2 = \mathbb{E}_{\min}(\Diamond(A \wedge \Diamond B))$.

3. **Slippery delivery with static obstacles** – an agent where the probability of moving perpendicular to the desired direction is 0.1 in each orientation. The task is to maximize the probability to go back and forth from locations A and B without colliding with the static obstacles $\hat{x}$: $\varphi_3 = \mathbb{P}_{\max}(\Box \Diamond A \wedge \Box \Diamond B \wedge \neg \Diamond X)$, with $x = \hat{x}$,

Evaluation. Fig. 8.3 compares the size of counterexample in relation to the probability of satisfying an LTL formula in each iteration of the synthesis procedure. In particular,

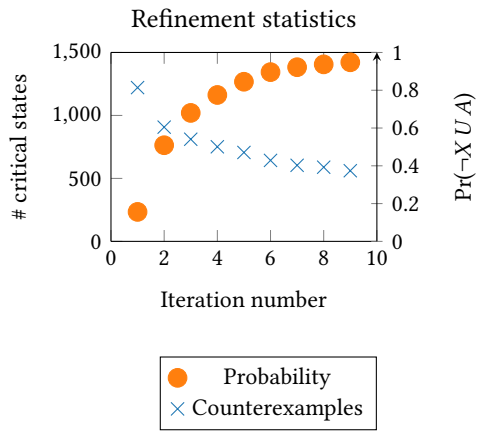


Figure 8.3: Progression of the number of critical states and the probability of satisfying an LTL specification as a result of local improvement steps.

we depict the size of the set $S' \subset S$ of critical states regarding $\varphi_1 = \mathbb{P}_{\max}(\neg X\ U\ A)$ for the *Navigation* example with grid-size 6. Note that even if the probability to satisfy the LTL specification is nearly one (for the initial state of the POMDP), there may still be critical intermediate states. As can be seen in the figure, while the probability to satisfy the LTL formula increases, the size of the counterexample decreases. In particular, the local improvement (Eq. 8.3, Sect. 8.3.3) is demonstrated to be effective.

Table 8.1 contains the results for the above LTL examples. Note that the sizes of the FSCs were included to demonstrate the trade-off between computational tractability and expressivity: a larger FSC means that the strategy can store more information, which may lead to better choices. However, larger FSCs require more computational effort and may require more data for training the RNN. We convey this trade-off in the experiments, as the size of the FSC is often problem-specific. Naturally, the strategies produced by the procedure will not have higher maximum probabilities (or lower minimum expected cost) than those generated by the PRISM-POMDP tool. However, they scale for significantly larger environments and settings. In the larger environments (*Navigation*(15) and upwards indicated by a star) we employ the sampling technique outlined at the end of Sect. 8.3.1 on a dataset with grid-size 10. The strategy still scales to these larger environments even when trained on data from a smaller state space.

Also in Table 8.1, we compare the effect of increasing the value of k for several k-FSCs. In smaller instances with grid-sizes of 4 and 5, memory-based strategies significantly outperform memoryless ones in terms of quality (the resulting probability or expected cost) while not consuming significantly more time. The increase in performance is due to additional expressiveness of an FSC-based strategy in these environments with a higher density of obstacles.

Summarized, our method scales to significantly larger domains than PRISM-POMDP with competitive computation times. As mentioned before, there is an inherent level of randomness in extracting a strategy. While we always take the first shot result for our experiments, the quality of strategies may improved by sampling several RNN predictions.

Table 8.1: Synthesizing strategies for examples with LTL specs.

Problem	States	Type, φ	RNN-based Synthesis Res.	RNN-based Synthesis Time (s)	PRISM-POMDP Res.	PRISM-POMDP Time (s)
Navigation (3)	333	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.74	**14.16**	**0.84**	73.88
Navigation (4)	1088	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.82	**22.67**	**0.93**	1034.64
Navigation (4) [2-FSC]	13373	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.91	47.26	–	–
Navigation (4) [4-FSC]	26741	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.92	59.42	–	–
Navigation (4) [8-FSC]	53477	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	**0.92**	85.26	–	–
Navigation (5)	2725	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.91	**34.34**	MO	MO
Navigation (5) [2-FSC]	33357	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.92	115.16	–	–
Navigation (5) [4-FSC]	66709	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.92	159.61	–	–
Navigation (5) [8-FSC]	133413	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	**0.92**	250.91	–	–
Navigation (10)	49060	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.79	**822.87**	MO	MO
Navigation (10) [2-FSC]	475053	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.83	1185.41	–	–
Navigation (10) [4-FSC]	950101	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	**0.85**	1488.77	–	–
Navigation (10) [8-FSC]	1900197	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	0.81	1805.22	–	–
Navigation (15)	251965	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	**0.91**	**1271.80***	MO	MO
Navigation (20)	798040	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	**0.96**	**4712.25***	MO	MO
Navigation (30)	4045840	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	**0.95**	**25191.05***	MO	MO
Navigation (40)	–	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_1	TO	TO	MO	MO
Delivery (4) [2-FSC]	80	$\mathbb{E}^{\mathcal{M}}_{\min}$, φ_2	6.02	35.35	**6.0**	**28.53**
Delivery (5) [2-FSC]	125	$\mathbb{E}^{\mathcal{M}}_{\min}$, φ_2	8.11	**78.32**	**8.0**	102.41
Delivery (10) [2-FSC]	500	$\mathbb{E}^{\mathcal{M}}_{\min}$, φ_2	**18.13**	**120.34**	MO	MO
Slippery (4) [2-FSC]	460	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_3	0.78	67.51	**0.90**	**5.10**
Slippery (5) [2-FSC]	730	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_3	0.89	84.32	**0.93**	**83.24**
Slippery (10) [2-FSC]	2980	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_3	**0.98**	**119.14**	MO	MO
Slippery (20) [2-FSC]	11980	$\mathbb{P}^{\mathcal{M}}_{\max}$, φ_3	**0.99**	**1580.42**	MO	MO

8.4.2 Comparison to existing POMDP examples

For comparison to existing benchmarks, we extend two examples from PRISM-POMDP for an arbitrary-sized structure: *Maze*(c) with $c+2$ rows and *Grid*(c) – a square grid with length c. We also compare to *RockSample* [258] (see Table 8.2b for problem metrics).

These problems are quite different to the LTL examples, in particular the significantly smaller observation spaces. As a result, a simple memoryless strategy is insufficient for a useful comparison. For each problem, the size of the k-FSC used is given by: Maze(c) has $k = (c+1)$; Grid(c) has $k = (c-1)$ and *RockSample* with b rocks has $k = b$.

Our method compares favorably with PRISM-POMDP and pomdpSolve for Maze and Grid (Table 8.2). However, the proposed method performs poorly in comparison to pomdpSolve for *RockSample*: An observation is received after taking an action to *check* a particular rock. This action is never sampled in the modified trajectory-tree based sampling method (Sect. 8.3.1). Note that our main aim is to enable the efficient synthesis of strategies under linear temporal logic constraints.

8.5 Conclusions and future research

We introduced a new RNN-based strategy synthesis method for POMDPs and LTL specifications. While we cannot guarantee optimality, our approach shows results that are

Table 8.2: Comparison for standard POMDP examples.

		RNN-based Synthesis			PRISM-POMDP		pomdpSolve	
Problem	Type	States	Res	Time (s)	Res	Time (s)	Res	Time (s)
Maze (1)	$E^{\mathcal{M}}_{\min}$	68	4.31	31.70	**4.30**	**0.09**	4.30	0.30
Maze (2)	$E^{\mathcal{M}}_{\min}$	83	5.31	46.65	5.23	2.176	**5.23**	**0.67**
Maze (3)	$E^{\mathcal{M}}_{\min}$	98	8.10	58.75	7.13	38.82	**7.13**	**2.39**
Maze (4)	$E^{\mathcal{M}}_{\min}$	113	11.53	58.09	8.58	543.06	**8.58**	**7.15**
Maze (5)	$E^{\mathcal{M}}_{\min}$	128	14.40	**68.09**	13.00	4110.50	**12.04**	132.12
Maze (6)	$E^{\mathcal{M}}_{\min}$	143	22.34	**71.89**	MO	MO	**18.52**	1546.02
Maze (10)	$E^{\mathcal{M}}_{\min}$	203	100.21	**158.33**	MO	MO	MO	MO
Grid (3)	$E^{\mathcal{M}}_{\min}$	165	2.90	38.94	2.88	2.332	**2.88**	**0.07**
Grid (4)	$E^{\mathcal{M}}_{\min}$	381	4.32	79.99	4.13	1032.53	**4.13**	**0.77**
Grid (5)	$E^{\mathcal{M}}_{\min}$	727	6.623	91.42	MO	MO	**5.42**	**1.94**
Grid (10)	$E^{\mathcal{M}}_{\min}$	5457	**13.630**	**268.40**	MO	MO	MO	MO
RockSample[4,4]	$E^{\mathcal{M}}_{\max}$	2432	17.71	35.35	N/A	N/A	**18.04**	**0.43**
RockSample[5,5]	$E^{\mathcal{M}}_{\max}$	8320	18.40	**43.74**	N/A	N/A	**19.23**	621.28
RockSample[7,8]	$E^{\mathcal{M}}_{\max}$	166656	20.32	**860.53**	N/A	N/A	**21.64**	20458.41

often close to the actual optimum, with competitive computation times for large problem domains. For future research, we are interested in extending our method to continuous state spaces, together with abstraction techniques that would enable to employ our model-based method.

9

Conclusions

We tackled several challenges in the design, development and deployment of robust autonomous systems. We regard autonomous systems as software systems capable to perceive the environment they operate in, reason about it and plan future actions – where both perception and planning are based on deep learning models. Since robustness has broad implications along each stage of the software development life cycle for autonomous systems, we proposed a holistic approach to achieve robustness that incorporates (i) a macro, system wide, perspective, and (ii) a micro, algorithmic, perspective.

Autonomous systems consist of multiple traditional software components, with which deep learning components for perception and planning have to be integrated. Therefore, we started from a macro, system wide perspective. In Chapter 2 we studied how software systems can be (re-)architected to support robust integration of deep learning components. Through a mixed-methods empirical study we identified twenty architectural tactics that can be used by practitioners to satisfy quality requirements of systems with deep learning components. These tactics represent an empirical framework that support the process of (re-)architecting software systems with deep learning components.

In Chapter 3 we observed that deep learning components complicate traditional software architecture design because of the inability to verify that deep learning components will satisfy their intended functionality and can cope with stochastic events coming from the operational environment. Since traditional software architecture analysis methods do not consider this complication, they fall short for systems with deep learning components. To facilitate the comparison of architecture alternatives for systems with deep learning components, we proposed a software architecture evaluation method that makes uncertainty a central decision driver. The method supports reasoning over how architectural patterns can mitigate uncertainty and enables comparison of different architectures focused on the interplay between deep learning and traditional software components. We also showed that design patterns used in safety-critical systems can be used to build robust autonomous systems with deep learning components.

In Chapter 4 we study the challenges raised by deep learning components at all stages of the development life cycle. Through an empirical study, we compiled a catalogue of engineering best practices for deep learning applications. Moreover, we studied the effects

of adopting the practices and the importance of each practice for the effects. These results provide a basis for quality assessment and improvement for teams developing software with deep learning components.

In the second part of the thesis, we zoomed in on the challenges raised by the development of robust deep learning components from a micro, algorithmic perspective. In particular, we tackled the development of computer vision models robust against adversarial examples and of verifiable deep learning based planning algorithms.

We focused on reducing the impact of adversarial training – the most effective defence against adversarial examples – on training time. Since adversarial training relies on finding representative adversarial samples for training, a procedure that slows down training considerably, it is imperative to find methods to develop robust computer vision models by alleviating the impact of adversarial training. To tackle this challenge, we proposed in Chapter 6 to train models on output spaces with large class separation, in order to gain robustness without adversarial training. We introduced a method to partition the output space into class prototypes with large separation and train models to preserve the separation. Experimental results show that models trained with these prototypes – which we call deep repulsive prototypes – gain robustness competitive with adversarial training, while also preserving more accuracy on natural samples. Moreover, the models are more resilient to large perturbation sizes.

In Chapter 7 we proposed a method to train robust classifiers with small training data sets and transfer the knowledge learned about robustness between different models. Towards this goal, we trained a meta-optimiser which learns to robustly optimise a model using perturbed samples and used it to transfer the knowledge learned to new models. Thus, the method eliminates the need of adversarial training once the meta-optimiser is trained. We show empirically that the meta-optimiser is consistent across different architectures and data sets, suggesting it is possible to transfer knowledge about robustness between different models.

In Chapter 8 we investigated the robustness of deep learning based planning algorithms, where formal verification can not be applied directly because the planning algorithms are too complex. In this context, we studied the problem of strategy synthesis for partially observable Markov decision processes, where the strategy is synthesised by a deep learning algorithm. To determine if the strategies adhere to (probabilistic) temporal logic constraints is computationally intractable and theoretically hard. In order to overcome this limitation, we introduce a method that combines techniques from deep learning and formal verification. The strategy is learned using a recurrent neural network and restricted to represent a finite memory strategy, which can be implemented on a specific partially observable Markov decision process. This allows formal verification techniques to be used in order to provide guarantees against temporal logic specifications.

9.1 Contributions

Several contributions were made in this thesis, as follows:

Chapter 2 introduced an empirical study of software architecture for deep learning, which resulted in:

- evidence that traditional software architecture challenges remain relevant for software systems with deep learning components, although new architectural challenges for deep learning also emerge;
- twenty solutions to the software architectural challenges for deep learning found both in literature and practice;
- a link from architectural solutions to software quality attributes which allowed the definition of twenty architectural tactics that can be used to satisfy individual quality attributes of software systems with deep learning components.

Chapter 3 introduced a method to compare software architectures with deep learning components, which resulted in:

- evidence that traditional software architecture evaluation methods do not take into account the uncertainty related to deep learning components;
- a method to compare software architectures with deep learning components that takes into account this uncertainty;
- a case study that demonstrates this method can be used to compare software architectures with deep learning components and evidence that software architecture design patterns for safety critical systems can be used to decrease the uncertainty of systems with deep learning components.

Chapter 4 introduced a study of software engineering best practices for deep learning, which resulted in:

- a catalogue of twenty-nine software engineering best practices for machine and deep learning applications;
- evidence that adoption of practices leads to measurable effects, such as traceability;
- an analysis of the contribution of each practice to the effects, which allows practitioners to estimate the return of investment for adopting the practices.

Chapter 5 introduced a brief summary of adversarial examples, based on a previous publication that provides:

- a comprehensive and self-contained survey of the field of adversarial examples which compares more than 100 attacks and defences;
- a comprehensive presentation of the hypotheses behind the existence of adversarial examples;
- a detailed discussion of the implications of adversarial examples to safety, security and robustness of deep learning.

Chapter 6 introduced a method to train robust deep learning models against adversarial examples without adversarial training, which resulted in:

- evidence that the impact of adversarial training can be alleviated by choosing inductive biases;
- evidence that the output space of classifiers is a good inductive bias for training robust deep learning models;
- a method to partition the output space into class prototypes that can be used to train models as robust as those adversarial trained, without the need to generate adversarial examples for training.

Chapter 7 introduced a meta-learning method to transfer knowledge about adversarial examples between models, which resulted in:

- evidence that knowledge about adversarial examples can be learned by a meta optimiser;
- evidence that the knowledge learned by a meta optimiser can be transferred to other models;
- a method to train adversarial robust models with meta-learning, which reduces the impact of adversarial training significantly.

Chapter 8 introduced a method to formally verify strategies of deep learning based planning algorithms, which resulted in:

- evidence that the strategies learned with recurrent neural networks can be reduced to formally verifiable problems;
- a method to formally verify these strategies;
- a method to improve the strategy using counter-examples generated by formal verification tools.

9.2 Reflection on research questions

We provide a reflection on the research questions from Table 1.1 in light of the contributions of this thesis.

How can software systems be (re-)architected to enable robust integration of deep learning components? The research presented in Chapter 2 designed to answer this question revealed that, although traditional software architecture challenges remain relevant for software with deep learning components, new challenges specific to deep learning also emerge. While traditional challenges have been well studied, we observed that new challenges have little support in the literature. Therefore, we designed and performed a broader study involving practitioners, which allowed us to present an extensive set of

solutions for all challenges. Software architecture is not always a fixed stage of the development life cycle. For example, in Agile development the software architecture is designed as the software is developed, while in Waterfall development the software architecture is designed upfront. Trends such as “Just enough software architecture" [75] propose to make architecture decisions as new risks are identified during software development.

The results of our study revealed that practitioners adopt a similar process for designing the software architecture of systems with deep learning components, tackling new concerns once they emerge. Unfortunately, this results in practitioners focusing more on traditional architecture concerns such as scalability or performance and less on concerns that are emphasised by deep learning, such as robustness. We believe that a method that bridges upfront and continuous architecture design is more suitable for systems with deep learning components, in order to embrace issues that arise with deep learning and the experimental development life cycle.

We made efforts to link architectural decisions and solutions to quality attributes of a system, such that the outcomes of architecture decisions can be better understood and mapped to quality attributes of deep learning components. A *tactic driven* approach seems suitable for (re-)architecting systems to enable robust integration of deep learning, as tactics can be used both upfront and continuous. Moreover, the effects of some tactics can be measured directly by instrumenting the system or the deep learning components. Therefore, evidence for the outcomes of the tactics employed can be gathered rapidly. Our initial efforts in this direction allowed us to compile a set of twenty architectural tactics that can be used to satisfy quality attributes of systems with deep learning components. Nonetheless, this set of tactics is not complete and we expect new tactics that address new quality attributes to emerge soon.

How to compare software architectures with deep learning components? As we just mentioned, practitioners still focus on solving traditional software architecture challenges, such as scalability or performance. Therefore, it is likely that traditional architecture evaluation methods tailored for these quality attributes can be used for software with deep learning components.

However, by analysing the literature we observed a lack of methods to evaluate how well an architecture copes with uncertainty of DL components, also called uncertainty due to “automated learning". This type of uncertainty is inherent to all deep learning components. Therefore, architecture evaluation methods should take into account this type of uncertainty, which is particularly relevant for safety critical systems, where deep learning components can appear to be functioning normally but produce wrong or uncertain predictions. In such scenarios, traditional architecture evaluation methods fall short, because the deep learning components will appear to be functioning normally.

To overcome this challenge, we proposed an architecture evaluation method that takes into account the uncertainty of deep learning components, both locally, as it impacts one component, and globally, as it propagates through a system. We validated our approach through a use-case, which brought evidence that different software architectures can lead to distinct uncertainty outcomes. We believe developments in this direction are needed in order to complement traditional architecture evaluation methods and enable comparisons of architectures with deep learning components. Moreover, since uncertainty can be

measured or approximated at run-time, these methods can be extended to analyse systems in operation and dynamically reconfigure them. This direction is tackled in the research field of self-adaptive systems, although the lack of studies involving systems with deep learning components is surprising. Nonetheless, we expect new developments in this field as deep learning becomes prevalent.

How do teams design, develop and deploy software with deep learning components? Software architecture is just one stage of the software development life cycle. Since all other stages can impact the robustness of autonomous systems, we performed a broader study to understand how teams design, develop and deploy systems with deep learning components. We note that in this case deployment includes maintenance and retirement of these systems.

To understand the team processes, we mined a broad set of best practices for each stage of the development life cycle, including team organisation and governance aspects. By running a global survey with over 300 participant teams, we could understand how teams adopt these practices and how these practice relate to various effects. We observed that practice adoption increases with team size and experience.

We also observed the adoption of best practices for engineering is rather low. In a subsequent study we extended the initial set of practices with practices related to trustworthy development of deep learning [247]. These practices included, among others, assurance of robustness. Unfortunately, we observed that the practices related to trustworthy deep learning have even lower adoption. These results are currently a cause of concern. Nonetheless, we expect that popularisation of best engineering practices together with new incentives (such as new regulations) will increase practice adoption and lead to more robust autonomous systems.

How can we reduce the impact of adversarial training on robust computer vision algorithms? The sensitivity of deep learning models to adversarial examples has been studied intensively in the last five years. However, most defences against adversarial examples have been broken and no technical solution for this issue exists. The most effective defence is still adversarial training, which involves complementing the training data set with adversarial examples.

Nonetheless, the process to generate adversarial examples is resource intensive and slows down the training process significantly. To alleviate this limitation, we proposed two methods. The first one is based on adding an inductive bias for learning, by partitioning the output space into class prototypes with large class separation. When new models are trained to preserve this separation, they become almost as robust against adversarial examples as adversarially trained models without the need to generate adversarial examples.

Also, the deep learning community seems to be firmly against inductive biases, in spite of the fact that they bring benefits. Subsequent experiments, performed at larger scale [212], showed that similar prototypes can be learned with supervision from language models and brings additional evidence that by specifying constraints on the output space we can develop robust models for computer vision. The second method we introduce is based on meta-learning and shows that information about the optimisation landscape of adversarially trained models can be transferred to new models without the need to generate

adversarial examples. While both methods reduce or remove the impact of adversarial training, we can not yet train completely robust models. Therefore, this remains an open problem.

Can we reduce the complexity of deep learning based planning algorithms and allow formal verification? Our work on reducing the complexity of deep learning based planning algorithms to allow formal verification shows it is possible to do so. The resulting strategy is correct, as each strategy prediction is evaluated using model checking, but not complete. The method is scalable, but not optimal.

Moreover, the method is limited to scenarios in which the underlying structure of the problem is known or can be uncovered. For large problems – such as performing reinforcement learning from unstructured raw observations (e.g., images) – we still do not have solutions that can allow formal verification. When training reinforcement learning algorithms from unstructured raw observations, the strategies learned are competitive. However, they can not be formally verified. We believe this interplay between reducing the representation of the world from unstructured raw observations to allow formal verification will play an important role in safe reinforcement learning. However, the solutions will likely be context-dependent, because a universal solution is not tractable.

9.3 Future research

Concrete future research directions were discussed in each chapter. We here provide higher level comments regarding our view on robustness of autonomous systems and the two main topics studied: software engineering for machine and deep learning and robust computer vision and planning.

With regard to to software engineering for machine and deep learning, we believe it is necessary to continue to expand and develop operational practices that can be directly applied by practitioners. We already made efforts to expand the catalogue of practices presented in Chapter 4 with practices regarding trustworthy development of deep learning components [247]. Further efforts to expand this catalogue of practices to tackle other engineering concerns should be paired with advanced techniques for measuring practice adoption. More fundamentally, a definition of quality for machine and deep learning components, compatible with previous quality definitions for traditional software, is needed. Research along these lines will provide practitioners with comparative measurements and benchmarks for the quality of their systems and propose step-wise improvements for distinct quality goals. Our initial exploration on this subject showed that applying more advanced data processing techniques for subjective adoption measurements coming from survey data (e.g., using item response theory) can be successfully used as an assessment and step-wise improvement instrument. Nevertheless, more concrete measurements that can be applied to code, data, infrastructure or team processes are needed.

Through various interactions with practitioners, we also observed a lack of educational materials in the area of software engineering for machine learning. Data scientists and machine learning practitioners have distinct backgrounds – ranging from physics, astronomy, to computer science or engineering – and their curriculum does not always provide education in software engineering. We made efforts to communicate the results gathered

in the first part of the thesis in a manner that is easily accessible to practitioners with distinct backgrounds and believe this material can serve as support for future courses and educational materials on the topic of software engineering for machine learning.

With regard to to the vulnerability of computer vision models to adversarial examples, we observed that research in this area is, for the most part, driven by benchmarks. While this approach helps to define comparative evaluation methods for adversarial robustness, we believe a more fundamental approach is needed. The vulnerability to adversarial examples shows the representations learned by neural networks are not consistent with the real world, are not semantically consistent and not criticisable. We believe that (i) using concepts from topology that can add structure to the internal representations or (ii) using multi-modal learning to pair representations learned from different data types will bridge the gap between the real world and the internal structure of neural networks. A first step in this direction has shown that large scale multi-modal learning benefits adversarial robustness [212]. However, it opens up a new range of adversarial attacks [89].

Since security is generally considered an arms race, we expect that clever attackers will find new methods to generate adversarial examples, no matter how sophisticated machine and deep learning models are and how well they perform on standard benchmarks. Therefore, we believe research in adversarial machine learning should borrow more concepts from traditional security, such as defining threat models, performing risk analyses and deciding when the vulnerability to adversarial examples is an important security threat.

With regard to to planning algorithms, we observed that deep learning based planning algorithms, such as the family of algorithms developed in deep reinforcement learning, can achieve better performance when not limited by the constraints imposed for making formal verification possible. However, such algorithms become impossible to formally verify. From an engineering perspective, the trade-offs between using planning algorithms that can or can not be formally verified should be discussed when the software architecture of a system is defined. Methods such as the one presented in Chapter 3, which make uncertainty a central decision driver, can be directly used or adapted depending on the use case. From a research perspective, a line of work that we find interesting is learning state representations from the environment, which can be later used in formal verification [9]. Our initial efforts in this direction showed it is possible to learn world representations for formal verification for simple use cases. However, scaling this method to larger worlds is left for future research.

Overall, we believe that the development of robust autonomous systems will always have to span multiple aspects, corresponding to different stages of the development life cycle. While there is no silver bullet to system or algorithmic robustness, the integration of contributions from software engineering and machine learning provides promising perspectives.

Bibliography

References

[1] *Towards Using Probabilistic Models to Design Software Systems with Inherent Uncertainty - Supplementary materials*, September 2020. Zenodo. doi: 10.5281/zenodo.4700095. URL https://doi.org/10.5281/zenodo.4700095.

[2] *An Empirical Study of Software Architecture for Machine Learning - Supplementary Material*, February 2021. Zenodo. doi: 10.5281/zenodo.4564113. URL https://doi.org/10.5281/zenodo.4564113.

[3] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 2018.

[4] Vijay Badrinarayanan Alex Kendall and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. In *Proceedings of the British Machine Vision Conference (BMVC)*, pages 1–12, 2017.

[5] Algorithmia. Best practices in machine learning infrastructure. https://algorithmia.com/blog/best-practices-in-machine-learning-infrastructure, 2019. [Online; accessed 22-03-2021].

[6] Altexsoft. How to organize data labelling for machine learning. https://www.altexsoft.com/blog/datascience/how-to-organize-data-labeling-for-machine-learning-\approaches-and-tools/, 2018. [Online; accessed 22-03-2021].

[7] Christopher Amato, Daniel S Bernstein, and Shlomo Zilberstein. Optimizing fixed-size stochastic controllers for POMDPs and decentralized POMDPs. *AAMAS*, 21(3): 293–320, 2010.

[8] Saleema Amershi, Andrew Begel, Christian Bird, Robert DeLine, Harald Gall, Ece Kamar, Nachiappan Nagappan, Besmira Nushi, and Thomas Zimmermann. Software engineering for machine learning: A case study. In *ICSE-SEIP*, pages 291–300. IEEE, 2019.

[9] Ankesh Anand, Evan Racah, Sherjil Ozair, Yoshua Bengio, Marc-Alexandre Côté, and R Devon Hjelm. Unsupervised state representation learning in Atari. In *NeurIPS*, volume 32, pages 3104–3112, 2019.

[10] Maksym Andriushchenko and Nicolas Flammarion. Understanding and improving fast adversarial training. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *NeurIPS*, volume 33, pages 16048–16059, 2020.

[11] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, pages 3981–3989, 2016.

[12] Adina Aniculaesei, Jörg Grieser, Andreas Rausch, Karina Rehfeldt, and Tim Warnecke. Toward a holistic software systems engineering approach for dependable autonomous systems. In *International Workshop on Software Engineering for AI in Autonomous Systems*, pages 23–30. IEEE, 2018.

[13] Ashraf Armoush. *Design patterns for safety-critical embedded systems.* PhD thesis, RWTH Aachen University, 2010.

[14] Anders Arpteg, Björn Brinne, Luka Crnkovic-Friis, and Jan Bosch. Software engineering challenges of deep learning. In *Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 50–59. IEEE, 2018.

[15] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *ICML*, 80: 274–283, 2018.

[16] Cigdem Avci, Bedir T, and Ioannis Athanasiadis. Software architectures for big data: a systematic literature review. *Big Data Analytics*, 5(1):1–53, 2020.

[17] Felix Bachmann, Len Bass, and Mark Klein. Deriving architectural tactics: A step toward methodical architectural design. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2003.

[18] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.

[19] Shumeet Baluja and Ian Fischer. Learning to attack: Adversarial transformation networks. In *AAAI*, 2018.

[20] Marco Barreno, Blaine Nelson, Russell Sears, Anthony D. Joseph, and J. Doug Tygar. Can machine learning be secure? In *ASIACCS*, pages 16–25. ACM, 2006.

[21] Marco Barreno, Blaine Nelson, Anthony D. Joseph, and JD. Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[22] Len Bass, Paul Clements, and Rick Kazman. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[23] Anatoliy Batyuk, Volodymyr Voityshyn, and Volodymyr Verhun. Software architecture design of the real-time processes monitoring platform. In *International Conference on Data Stream Mining & Processing*, pages 98–101. IEEE, 2018.

[24] Denis Baylor, Eric Breck, Heng-Tze Cheng, Noah Fiedel, Chuan Yu Foo, Zakaria Haque, Salem Haykal, Mustafa Ispir, Vihan Jain, Levent Koc, et al. TFX: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1387–1395, 2017.

[25] Denis Baylor, Kevin Haas, Konstantinos Katsiapis, Sammy Leong, Rose Liu, Clemens Menwald, Hui Miao, Neoklis Polyzotis, Mitchell Trott, and Martin Zinkevich. Continuous training for production ML in the tensorflow extended (TFX) platform. In *2019 USENIX Conference on Operational Machine Learning (OpML 19)*, pages 51–53, 2019.

[26] Edmon Begoli and James Horey. Design principles for effective knowledge discovery from big data. In *Joint Working Conference on Software Architecture and European Conference on Software Architecture*, pages 215–218. IEEE, 2012.

[27] Sagar Behere and Martin Törngren. A functional reference architecture for autonomous driving. *Information and Software Technology*, 73:136–150, 2016.

[28] Hrvoje Belani, Marin Vukovic, and Željka Car. Requirements engineering challenges in building ai-based complex systems. In *International Requirements Engineering Conference Workshops (REW)*, pages 252–255. IEEE, 2019.

[29] Samy Bengio, Yoshua Bengio, Jocelyn Cloutier, and Jan Gecsei. On the optimization of a synaptic learning rule. In *Optimality in Artificial and Biological Neural Networks*, pages 6–8. Univ. of Texas, 1992.

[30] Battista Biggio and Fabio Roli. Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331, 2018.

[31] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *ICML*, page 1467–1474, 2012.

[32] Battista Biggio, Giorgio Fumera, and Fabio Roli. Security evaluation of pattern classifiers under attack. In *TKDE*, volume 26, pages 984–996. IEEE, 2013.

[33] Alessandro Biondi, Federico Nesti, Giorgiomaria Cicero, Daniel Casini, and Giorgio Buttazzo. A safe, secure, and predictable software architecture for deep learning in safety-critical systems. *IEEE Embedded Systems Letters*, 12(3):78–82, 2019.

[34] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. arXiv:1604.07316, 2016.

[35] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[36] Eric Breck, Shanqing Cai, Eric Nielsen, Michael Salib, and D Sculley. The ML test score: A rubric for ML production readiness and technical debt reduction. In *International Conference on Big Data (Big Data)*, pages 1123–1132. IEEE, 2017.

[37] Cristiano Breuel. ML Ops: Machine learning as an engineered disciplined. https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline\-b86ca4874a3f, 2019. [Online; accessed 22-03-2021].

[38] Michael Brückner, Christian Kanzow, and Tobias Scheffer. Static prediction games for adversarial learning problems. *JMLR*, 13(1):2617–2654, 2012.

[39] Sébastien Bubeck, Yin Tat Lee, Eric Price, and Ilya Razenshteyn. Adversarial examples from computational constraints. In *ICML*, pages 831–840, 2019.

[40] David Budgen, Mark Turner, Pearl Brereton, and Barbara A Kitchenham. Using mapping studies in software engineering. In *PPIG*, volume 8, pages 195–204, 2008.

[41] Nicholas Carlini and David Wagner. Adversarial examples are not easily detected: Bypassing ten detection methods. In *AISec*, pages 3–14. ACM, 2017.

[42] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *S&P*, pages 39–57. IEEE, 2017.

[43] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *S&P Workshops*, pages 1–7. IEEE, 2018.

[44] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden voice commands. In *USENIX Security*, pages 513–530, 2016.

[45] Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian J. Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. *arXiv:1902.06705*, 2019.

[46] Steven Carr, Nils Jansen, Ralf Wimmer, Alexandru C. Serban, Bernd Becker, and Ufuk Topcu. Counterexample-guided strategy improvement for pomdps using recurrent neural networks. In *IJCAI*, pages 5532–5539, 2019.

[47] Simon Chan. A design pattern for machine learning with Scala. https://www.youtube.com/watch?v=hhXs4AOGRpI, 2020. [Online; accessed 22-03-2021].

[48] Krishnendu Chatterjee, Martin Chmelík, Raghav Gupta, and Ayush Kanodia. Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In *ICRA*, pages 325–330, 2015.

[49] Krishnendu Chatterjee, Martin Chmelík, and Mathieu Tracol. What is decidable about partially observable Markov decision processes with ω-regular objectives. *Journal of Computer and System Sciences*, 82(5):878–911, 2016.

[50] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *JAIR*, 16:321–357, 2002.

[51] Marsha Chechik. Uncertain requirements, assurance and machine learning. In *International Requirements Engineering Conference (RE)*, pages 2–3. IEEE, 2019.

[52] Jiefeng Chen, Xi Wu, Yingyu Liang, and Somesh Jha. Improving adversarial robustness by data-specific discretization. *arXiv:1805.07816*, 2018.

[53] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. ZOO: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *AISec*, page 15–26. ACM, 2017.

[54] François Chollet. Keras. https://keras.io, 2015.

[55] Marcus Ciolkowski, Oliver Laitenberger, Sira Vegas, and Stefan Biffl. Practical experiences in the design and conduct of surveys in empirical software engineering. In *Empirical methods and studies in software engineering*, pages 104–128. Springer, 2003.

[56] Cloudfactory. The ultimate guide to data labeling for ml. https://www.cloudfactory.com/data-labeling-guide, 2019. [Online; accessed 22-03-2021].

[57] Daniele Codetta-Raiteri and Luigi Portinale. Dynamic bayesian networks for fault detection, identification, and recovery in autonomous spacecraft. *IEEE Transactions on Systems, Man, and Cybernetics*, 45(1):13–24, 2014.

[58] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE CVPR*, pages 3213–3223, 2016.

[59] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *ICML*, pages 2206–2216, 2020.

[60] Daniela S Cruzes and Tore Dyba. Recommended steps for thematic synthesis in software engineering. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 275–284. IEEE, 2011.

[61] Daniel Cullina, Arjun Nitin Bhagoji, and Prateek Mittal. Pac-learning in the presence of adversaries. In *NeurIPS*, volume 31, 2018.

[62] Nilesh Dalvi, Pedro Domingos, Sumit Sanghai, Deepak Verma, et al. Adversarial classification. In *KDD*, page 99–108. ACM, 2004.

[63] Elizamary de Souza Nascimento, Iftekhar Ahmed, Edson Oliveira, Márcio Piedade Palheta, Igor Steinmacher, and Tayana Conte. Understanding development process of machine learning systems: Challenges and solutions. In *ESEM*, pages 1–6. IEEE, 2019.

[64] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *CAV*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

[65] Google Devs. Testing and debugging in machine learning. https://developers.google.com/machine-learning/testing-debugging/pipeline/production, 2019. [Online; accessed 22-03-2021].

[66] Liliana Dobrica and Eila Niemela. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[67] Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. *UAI*, 1 (2):1–38, 2018.

[68] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008.

[69] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, pages 2650–2658, 2015.

[70] Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013.

[71] Naeem Esfahani, Sam Malek, and Kaveh Razavi. Guidearch: guiding the exploration of architectural solution space under uncertainty. In *ICSE*, pages 43–52. IEEE, 2013.

[72] Leire Etxeberria, Catia Trubiani, Vittorio Cortellessa, and Goiuria Sagardui. Performance-based selection of software and hardware features under parameter uncertainty. In *International Conference on Quality of Software Architectures*, pages 23–32, 2014.

[73] Julian Everett. Daisy architecture. https://datalanguage.com/blog/daisy-architecture, 2019. [Online; accessed 22-03-2021].

[74] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning visual classification. In *CVPR*, pages 1625–1634. IEEE, 2018.

[75] George Fairbanks. *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd, 2010.

[76] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, pages 1126–1135, 2017.

[77] Luciano Floridi. Establishing the rules for building trustworthy ai. *Nature Machine Intelligence*, 1(6):261–262, 2019.

[78] Alexandre Fréchette, Lars Kotthoff, Tomasz P. Michalak, Talal Rahwan, Holger H. Hoos, and Kevin Leyton-Brown. Using the shapley value to analyze algorithm portfolios. In *AAAI*, pages 3397–3403, 2016.

[79] Freeandopenmachinelearning. ML reference architecture. https://freeandopenmachinelearning.readthedocs.io/en/latest/architecture.html, 2020. [Online; accessed 22-03-2021].

[80] Ji Gao, Beilun Wang, Zeming Lin, Weilin Xu, and Yanjun Qi. Deepcloak: Masking deep neural network models for robustness against adversarial samples. *ICLR Workshops*, 2017.

[81] David Garlan. Software engineering in an uncertain world. In *FSE/SDP Workshop on Future of Software Engineering Research*, pages 125–128, 2010.

[82] Vahid Garousi, Michael Felderer, and Mika V Mäntylä. Guidelines for including grey literature and conducting multivocal literature reviews in software engineering. *Information and Software Technology*, 106:101–121, 2019.

[83] Vahid Garousi, Michael Felderer, Mika V Mäntylä, and Austen Rainer. *Benefitting from the Grey Literature in Software Engineering Research*, pages 385–413. Springer, 2020.

[84] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *S&P*, pages 3–18. IEEE, 2018.

[85] Partha Ghosh, Arpan Losalka, and Michael J. Black. Resisting adversarial attacks using gaussian mixture variational autoencoders. In *AAAI*, pages 541–548, 2018.

[86] Justin Gilmer, Ryan P. Adams, Ian Goodfellow, David Andersen, and George E. Dahl. Motivating the rules of the game for adversarial example research. *arXiv:1807.06732*, 2018.

[87] Justin Gilmer, Luke Metz, Fartash Faghri, Samuel S. Schoenholz, Maithra Raghu, Martin Wattenberg, and Ian Goodfellow. Adversarial spheres. *arXiv:1801.02774*, 2018.

[88] Amir Globerson and Sam Roweis. Nightmare at test time: robust learning by feature deletion. *ICML*, pages 353–360, 2006.

[89] Gabriel Goh, Nick Cammarata, Chelsea Voss, Shan Carter, Michael Petrov, Ludwig Schubert, Alec Radford, and Chris Olah. Multimodal neurons in artificial neural networks. *Distill*, 6(3):e30, 2021.

[90] Micah Goldblum, Liam Fowl, and Tom Goldstein. Robust few-shot learning with adversarially queried meta-learners. *arXiv:1910.00982*, 2019.

[91] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv:1412.6572*, 2014.

[92] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. Adversarial perturbations against deep neural networks for malware classification. *arXiv:1606.04435*, 2016.

[93] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. On the (statistical) detection of adversarial examples. *arXiv:1702.06280*, 2017.

[94] Samantha Guerriero, Barbara Caputo, and Thomas Mensink. Deep nearest class mean classifiers. In *ICML, Worskhop Track*, 2018.

[95] Joy Paul Guilford. Psychometric methods. 1954.

[96] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. *ICLR*, pages 1–12, 2018.

[97] Mark Haakman, Luís Cruz, Hennie Huijgens, and Arie van Deursen. Ai lifecycle models need to be revised. an exploratory study in fintech. *arXiv:2010.02716*, 2020.

[98] Sofie Haesaert, Petter Nilsson, Cristian Ioan Vasile, Rohan Thakker, Ali-akbar Aghamohammadi, Aaron D. Ames, and Richard M. Murray. Temporal logic control of POMDPs via label-based stochastic simulation relations. In *ADHS*, volume 51(16), pages 271–276. Elsevier, 2018.

[99] Neil B Harrison and Paris Avgeriou. How do architecture patterns and tactics interact? a model and annotation. *Journal of Systems and Software*, 83(10):1735–1758, 2010.

[100] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable MDPs. *arXiv:1507.06527*, 2015.

[101] Milos Hauskrecht. Value-function approximations for partially observable Markov decision processes. *JAIR*, 13:33–94, 2000.

[102] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE, 2016.

[103] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *ICLR*, pages 1–16, 2019.

[104] David Herron. Principled machine learning: Practices and tools for efficient collaboration. https://dev.to/robogeek/principled-machine-learning-4eho, 2019. [Online; accessed 22-03-2021].

[105] Sibylle Hess, Wouter Duivesteijn, and Decebal Mocanu. Softmax-based classification is k-means clustering: Formal proof, consequences for adversarial attacks, and improvement through centroid based tailoring. *arXiv:2001.01987*, 2020.

[106] High-Level Expert Group on AI. Ethics guidelines for trustworthy AI. https://ec.europa.eu/digital-single-market/en/news/ethics-guidelines-trustworthy-ai, 2019. [Online; accessed 22-03-2021].

[107] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[108] Joeri Hofmans, Peter Theuns, Sven Baekelandt, Olivier Mairesse, Niels Schillewaert, and Walentina Cools. Bias and changes in perceived intensity of verbal qualifiers effected by scale orientation. *Survey Research Methods*, 1(2):97–108, 2007.

[109] Angela Horneman, Andrew Mellinger, and Ipek Ozkaya. Ai engineering: 11 foundational practices. Technical report, Carnegie Mellon University Pittsburgh, 2020.

[110] Siw Elisabeth Hove and Bente Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *International Software Metrics Symposium*, pages 10–20. IEEE, 2005.

[111] Weiwei Hu and Ying Tan. Generating adversarial malware examples for black-box attacks based on gan. *arXiv:1702.05983*, 2017.

[112] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *CVPR*, pages 4700–4708. IEEE, 2017.

[113] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *CAV (1)*, volume 10426 of *LNCS*, pages 3–29. Springer, 2017.

[114] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, Kaoutar El Maghraoui, Anupama Murthi, and Punleuk Oum. Modelops: Cloud-based lifecycle management for reliable and trusted ai. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 113–120. IEEE, 2019.

[115] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *ICML*, pages 2137–2146, 2018.

[116] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *NeurIPS*, pages 125–136, 2019.

[117] Fuyuki Ishikawa and Nobukazu Yoshioka. How do engineers perceive difficulties in engineering of machine-learning systems?: questionnaire survey. In *Proceedings of the Joint 7th International Workshop on Conducting Empirical Studies in Industry and 6th International Workshop on Software Engineering Research and Industrial Practice*, pages 2–9. IEEE, 2019.

[118] ISO. Systems and software engineering — systems and software quality requirements and evaluation. Technical report, Technical Report. ISO/IEC 25010, 2011.

[119] Jörn-Henrik Jacobsen, Jens Behrmann, Nicholas Carlini, and Nicolas Florian Tramer. Exploiting excessive invariance caused by norm-bounded adversarial robustness. *ICLR*, pages 140–155, 2019.

[120] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *IEEE CVPR Workshops*, pages 11–19, 2017.

[121] Charles Jin and Martin Rinard. Manifold regularization for adversarial robustness. *arXiv:2003.04286*, 2020.

[122] Jason Jo and Yoshua Bengio. Measuring the tendency of CNNs to learn surface statistical regularities. *arXiv:1711.11561*, 2017.

[123] Anna Jobin, Marcello Ienca, and Effy Vayena. The global landscape of ai ethics guidelines. *Nature Machine Intelligence*, 1(9):389–399, 2019.

[124] Per John. Software development best practices in a deep learning environment. https://towardsdatascience.com/software-development-best-practices-in-a-deep-\learning-environment-a1769e9859b1, 2019. [Online; accessed 22-03-2021].

[125] Sebastian Junges, Nils Jansen, Ralf Wimmer, Tim Quatmann, Leonore Winterer, Joost-Pieter Katoen, and Bernd Becker. Finite-state controllers of POMDPs via parameter synthesis. In *UAI*, 2018.

[126] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1):99–134, 1998.

[127] Davinder Kaur, Suleyman Uslu, and Arjan Durresi. Requirements for trustworthy artificial intelligence–a review. In *International Conference on Network-Based Information Systems*, pages 105–115. Springer, 2020.

[128] Rick Kazman, S Jeromy Carrière, and Steven G Woods. Toward a discipline of scenario-based architectural engineering. *Annals of Software Engineering*, 9(1-2): 5–33, 2000.

[129] Michael J Kearns, Yishay Mansour, and Andrew Y Ng. Approximate planning in large POMDPs via reusable trajectories. In *NeurIPS*, pages 1001–1007, 2000.

[130] Alex Kendall and Yarin Gal. What uncertainties do we need in bayesian deep learning for computer vision? In *NeurIPS*, pages 5574–5584, 2017.

[131] Foutse Khomh, Bram Adams, Jinghui Cheng, Marios Fokaefs, and Giuliano Antoniol. Software engineering for machine-learning applications: The road ahead. *IEEE Software*, 35(5):81–84, 2018.

[132] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *ICLR*, 2015.

[133] Barbara Kitchenham and Stuart Charters. Guidelines for performing systematic literature reviews in software engineering. 2007.

[134] Barbara A Kitchenham. Systematic review in software engineering: where we are and where we should be going. In *International Workshop on Evidential Assessment of Software Technologies*, pages 1–2, 2012.

[135] Barbara A Kitchenham and Shari L Pfleeger. Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer, 2008.

[136] Barbara A Kitchenham and Shari Lawrence Pfleeger. Principles of survey research part 2: designing a survey. *ACM SIGSOFT Software Engineering Notes*, 27(1):18–20, 2002.

[137] Aleksander Kołcz and Choon Hui Teo. Feature weighting for improved classifier robustness. In *CEAS'09: Sixth Conference on Email and Anti-spam*, 2009.

[138] Bettina Könighofer, Roderick Bloem, Sebastian Junges, Nils Jansen, and Alex Serban. Safe reinforcement learning using probabilistic shields. In *International Conference on Concurrency Theory: 31st CONCUR 2020: Vienna, Austria (Virtual Conference)*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2020.

[139] Felix Kreuk, Assi Barak, Shir Aviv-Reuven, Moran Baruch, Benny Pinkas, and Joseph Keshet. Adversarial examples on discrete sequences for beating whole-binary malware detection. *arXiv:1802.04528*, 2018.

[140] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[141] Philippe Kruchten. What do software architects really do? *Journal of Systems and Software*, 81(12):2413–2416, 2008.

[142] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv:1611.01236*, 2016.

[143] Hiroshi Kuwajima, Hirotoshi Yasuoka, and Toshihiro Nakae. Engineering problems in machine learning systems. *Machine Learning*, 109(5):1103–1126, 2020.

[144] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[145] Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *JMLR*, 10(Jan), 2009.

[146] James Le. 10 best practices for deep learning. https://nanonets.com/blog/10-best-practices-deep-learning/, 2019. [Online; accessed 22-03-2021].

[147] Hyeungill Lee, Sungyeob Han, and Jungwoo Lee. Generative adversarial trainer: Defense to adversarial perturbations with gan. *arXiv:1705.03387*, 2017.

[148] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(1):1334–1373, 2016.

[149] Ke Li and Jitendra Malik. Learning to optimize. *arXiv:1606.01885*, 2016.

[150] Ke Li and Jitendra Malik. Learning to optimize neural nets. *arXiv:1703.00441*, 2017.

[151] Li Erran Li, Eric Chen, Jeremy Hermann, Pusheng Zhang, and Luming Wang. Scaling machine learning as a service. In *International Conference on Predictive Applications and APIs*, pages 14–29, 2017.

[152] Shasha Li, Ajaya Neupane, Sujoy Paul, Chengyu Song, Srikanth V. Krishnamurthy, Amit K. Roy Chowdhury, and Ananthram Swami. Adversarial perturbations against real-time video classification systems. *arXiv:1807.00458*, 2018.

[153] Michael L. Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via GLTL. *arXiv preprint 1704.04341*, 2017.

[154] Hanyan Liu, Samuel Eksmo, Johan Risberg, and Regina Hebig. Emerging and changing tasks in the development process for machine learning systems. In *International Conference on Software and System Processes*, pages 125–134, 2020.

[155] Hsueh-Ti Derek Liu, Michael Tao, Chun-Liang Li, Derek Nowrouzezahrai, and Alec Jacobson. Beyond pixel norm-balls: Parametric adversaries using an analytically differentiable renderer. *ICLR*, 2018.

[156] Qiang Liu, Pan Li, Wentao Zhao, Wei Cai, Shui Yu, and Victor CM Leung. A survey on security threats and defensive techniques of machine learning: A data driven view. *IEEE Access*, 6:12103–12117, 2018.

[157] Daniel Lowd and Christopher Meek. Adversarial learning. In *KDD*, pages 641–647. ACM, 2005.

[158] Jiajun Lu, Theerasit Issaranon, and David A. Forsyth. Safetynet: Detecting and rejecting adversarial examples robustly. In *ICCV*, pages 446–454. IEEE, 2017.

[159] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):2522–5839, 2020.

[160] Lucy Ellen Lwakatare, Aiswarya Raj, Jan Bosch, Helena Holmström Olsson, and Ivica Crnkovic. A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In *International Conference on Agile Software Development*, pages 227–243. Springer, 2019.

[161] Omid Madani, Steve Hanks, and Anne Condon. On the undecidability of probabilistic planning and infinite-horizon partially observable Markov decision problems. In *AAAI*, pages 541–548, 1999.

[162] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

[163] Sara Mahdavi-Hezavehi, Paris Avgeriou, and Danny Weyns. A classification framework of uncertainty in architecture-based self-adaptive systems with multiple quality requirements. In *Managing Trade-Offs in Adaptable Software Architectures*, pages 45–77. Elsevier, 2017.

[164] Chengzhi Mao, Ziyuan Zhong, Junfeng Yang, Carl Vondrick, and Baishakhi Ray. Metric learning for adversarial robustness. In *NeurIPS*, pages 480–491, 2019.

[165] David Maplesden, Ewan Tempero, John Hosking, and John C Grundy. Performance analysis for object-oriented software: A systematic mapping. *IEEE Transactions on Software Engineering*, 41(7):691–710, 2015.

[166] ISO May. Systems and software engineering–architecture description. Technical report, Technical Report. ISO/IEC/IEEE 42010, 2011.

[167] Richard McElreath. *Statistical rethinking: A Bayesian course with examples in R and Stan.* CRC press, 2020.

[168] Gary McGraw, Richie Bonett, Harold Figueroa, and Victor Shepardson. Security engineering for machine learning. *Computer*, 52(8):54–57, 2019.

[169] Gary McGraw, Harold Figueroa, Victor Shepardson, and Richie Bonett. An architectural risk analysis of ml systems: Toward more secure ml. *Berryville Institute of Machine Learning, Clarke County, VA.*, 23, 2020.

[170] Indika Meedeniya, Aldeida Aleti, and Lars Grunske. Architecture-driven reliability optimization with uncertain model parameters. *Journal of Systems and Software*, 85 (10):2340–2355, 2012.

[171] V.M. Megler. Managing machine learning projects. https://d1.awsstatic.com/whitepapers/aws-managing-ml-projects.pdf, 2019. [Online; accessed 22-03-2021].

[172] Dongyu Meng and Hao Chen. Magnet: a two-pronged defense against adversarial examples. In *ACM CCS*, pages 135–147. ACM, 2017.

[173] Pradeep Menon. Demystifying data lake architecture. https://www.datasciencecentral.com/profiles/blogs/demystifying-data-lake-architecture, 2020. [Online; accessed 22-03-2021].

[174] Pascal Mettes, Elise van der Pol, and Cees Snoek. Hyperspherical prototype networks. In *NeurIPS*, pages 1487–1497, 2019.

[175] Jan Hendrik Metzen, Tim Genewein, Volker Fischer, and Bastian Bischoff. On detecting adversarial perturbations. *ICLR*, 2017.

[176] Nicolas Meuleau, Leonid Peshkin, Kee-Eung Kim, and Leslie Pack Kaelbling. Learning finite-state controllers for partially observable environments. In *UAI*, pages 427–436, 1999.

[177] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NeurIPS*, pages 3111–3119, 2013.

[178] Matthew Mirman, Timon Gehr, and Martin Vechev. Differentiable abstract interpretation for provably robust neural networks. *ICML*, pages 3578–3586, 2018.

[179] Tom M. Mitchell et al. Machine learning. 1997. *Burr Ridge, IL: McGraw Hill*, 45(37): 870–877, 1997.

[180] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing with virtual adversarial training. *arXiv:1507.00677*, 2015.

[181] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[182] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *CVPR*, pages 1765–1773. IEEE, 2017.

[183] Aamir Mustafa, Salman Khan, Munawar Hayat, Roland Goecke, Jianbing Shen, and Ling Shao. Adversarial defense by restricting the hidden space of deep neural networks. In *ICCV*, pages 3385–3394, 2019.

[184] Nina Narodytska and Shiva Prasad Kasiviswanathan. Simple black-box adversarial perturbations on deep neural networks. In *CVPR Workshops*, volume 2, 2017.

[185] Elizamary Nascimento, Anh Nguyen-Duc, Ingrid Sundbø, and Tayana Conte. Software engineering for artificial intelligence and machine learning software: A systematic literature review. *arXiv:2011.03751*, 2020.

[186] National Science and Technology Council (US). Select Committee on Artificial Intelligence. The national artificial intelligence research and development strategic plan: 2019 update. https://www.nitrd.gov/news/National-AI-RD-Strategy-2019.aspx, 2019. [Online; accessed 22-03-2021].

[187] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *CVPR*, pages 427–436. IEEE, 2015.

[188] Phuoc Nguyen and Dat Tran. Repulsive-svdd classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 277–288. Springer, 2015.

[189] Yasuharu Nishi, Satoshi Masuda, Hideto Ogawa, and Keiji Uetsuki. A test architecture for machine learning product. In *International Conference on Software Testing, Verification and Validation Workshops*, pages 273–278. IEEE, 2018.

[190] Gethin Norman, David Parker, and Xueyi Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, 2017.

[191] Linda Northrop, Ipek Ozkaya, George Fairbanks, and Michael Keeling. Designing the software systems of the future. *ACM SIGSOFT Software Engineering Notes*, 43(4): 28–30, 2019.

[192] Lawrence A Palinkas, Sarah M Horwitz, Carla A Green, et al. Purposeful sampling for qualitative data collection and analysis in mixed method implementation research. *Administration and policy in mental health and mental health services research*, 42(5): 533–544, 2015.

[193] Tianyu Pang, Kun Xu, Yinpeng Dong, Chao Du, Ning Chen, and Jun Zhu. Rethinking softmax cross-entropy loss for adversarial robustness. *ICLR*, 2019.

[194] Tianyu Pang, Kun Xu, and Jun Zhu. Mixup inference: Better exploiting mixup to defend adversarial attacks. *ICLR*, 2019.

[195] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[196] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv:1803.04765*, 2018.

[197] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, et al. Technical report on the cleverhans v2. 1.0 adversarial examples library. *arXiv:1610.00768*, 2016.

[198] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. Transferability in machine learning: from phenomena to black-box attacks using adversarial samples. *arXiv:1605.07277*, 2016.

[199] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *ASIACCS*, pages 506–519. ACM, 2017.

[200] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael P. Wellman. Sok: Security and privacy in machine learning. In *EuroS&P*, pages 399–414. IEEE, 2018.

[201] Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *ICLR*, 2018.

[202] Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *ICLR*, 2014.

[203] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier, 2014.

[204] Diego Perez-Palacin and Raffaela Mirandola. Uncertainties in the modeling of self-adaptive systems: A taxonomy and an example of availability evaluation. In *ACM/SPEC International Conference on Performance Engineering*, pages 3–14, 2014.

[205] Joelle Pineau, Geoff Gordon, and Sebastian Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *IJCAI*, pages 1025–1032, 2003.

[206] Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research (a report from the NeurIPS 2019 reproducibility program). *arXiv:2003.12206*, 2020.

[207] Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE Computer Society, 1977. doi: 10.1109/SFCS.1977.32.

[208] Neoklis Polyzotis, Martin Zinkevich, Sudip Roy, Eric Breck, and Steven Whang. Data validation for machine learning. *Machine Learning and Systems*, 1:334–347, 2019.

[209] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *CVPR*, pages 4422–4431. IEEE, 2018.

[210] Jennifer Prendki. The curse of big data labeling and three ways to solve it. https://aws.amazon.com/blogs/apn/the-curse-of-big-data-\labeling-and-three-ways-to-solve-it/, 2018. [Online; accessed 22-03-2021].

[211] Alexander Pritzel, Benigno Uria, Sriram Srinivasan, Adrià Puigdomènech Badia, Oriol Vinyals, Demis Hassabis, Daan Wierstra, and Charles Blundell. Neural episodic control. In *ICML*, volume 70 of *Proc. of Machine Learning Research*, pages 2827–2836. PMLR, 2017.

[212] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. *arXiv:2103.00020*, 2021.

[213] Md Saidur Rahman, Emilio Rivera, Foutse Khomh, Yann-Gaël Guéhéneuc, and Bernd Lehnert. Machine learning software engineering in practice: An industrial case study. *arXiv:1906.07154*, 2019.

[214] Adnan Siraj Rakin, Zhezhi He, Boqing Gong, and Deliang Fan. Blind pre-processing: A robust defense method against adversarial examples. *arXiv:1802.01549*, 2018.

[215] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. *ICLR*, 2017.

[216] Lars Reimann and Günter Kniesel-Wünsche. Achieving guidance in applied machine learning through software engineering techniques. In *International Conference on Art, Science, and Engineering of Programming*, pages 7–12, 2020.

[217] Leslie Rice, Eric Wong, and J Zico Kolter. Overfitting in adversarially robust deep learning. *ICML*, pages 8093–8104, 2020.

[218] Gordon Rios. Patterns (and anti-patterns) for developing machine learning systems. https://www.usenix.org/legacy/events/sysml08/tech/rios_talk.pdfl, 2019. [Online; accessed 22-03-2021].

[219] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data-ai integration perspective. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[220] Bernd Rohrmann. Verbal qualifiers for rating scales: Sociolinguistic considerations and psychometric data. *University of Melbourne, Australia*, 2007.

[221] Ishai Rosenberg, Asaf Shabtai, Lior Rokach, and Yuval Elovici. Generic black-box end-to-end attack against rnns and other api calls based malware classifiers. *arXiv:1707.05970*, 2017.

[222] Roshanak Roshandel, Nenad Medvidovic, and Leana Golubchik. A bayesian model for predicting reliability of software systems at the architectural level. In *International Conference on the Quality of Software Architectures*, pages 108–126. Springer, 2007.

[223] Alkis Polyzotis Martin A. Zinkevich Steven Whang Sudip Roy. Data management challenges in production machine learning. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/45a9dcf23dbdfa24dbced358f825636c58518afa.pdf, 2017. [Online; accessed 22-03-2021].

[224] Benjamin IP. Rubinstein, Blaine Nelson, Ling Huang, Anthony D. Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and JD. Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *SIGCOMM*, pages 1–14. ACM, 2009.

[225] Sara Sabour, Yanshuai Cao, Fartash Faghri, and David J. Fleet. Adversarial manipulation of deep representations. *arXiv:1511.05122*, 2015.

[226] Hadi Salman, Andrew Ilyas, Logan Engstrom, Ashish Kapoor, and Aleksander Madry. Do adversarially robust imagenet models transfer better? *NeurIPS*, 33, 2020.

[227] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, pages 1842–1850, 2016.

[228] Adam Santoro, Ryan Faulkner, David Raposo, Jack W. Rae, Mike Chrzanowski, Theophane Weber, Daan Wierstra, Oriol Vinyals, Razvan Pascanu, and Timothy P. Lillicrap. Relational recurrent neural networks. In *NeurIPS*, pages 7310–7321, 2018.

[229] Carlton Sapp. Preparing and architecting for machine learning. https://www.gartner.com, 2017. [Online; accessed 22-03-2021].

[230] Danilo Sato, Arif Wider, and Christoph Windheuser. Continuous delivery for machine learning. https://martinfowler.com/articles/cd4ml.html, 2019. [Online; accessed 22-03-2021].

[231] Leonard J Savage. *The foundations of statistics*. Courier Corporation, 1972.

[232] Max Scheerer, Jonas Klamroth, Ralf Reussner, and Bernhard Beckert. Towards classes of architectural dependability assurance for machine-learning-based systems. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 31–37, 2020.

[233] PB Schiilkop, Chris Burgest, and Vladimir Vapnik. Extracting support data for a given task. In *Proceedings, First International Conference on Knowledge Discovery & Data Mining. AAAI Press*, pages 252–257, 1995.

[234] Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *ICNN*, pages 407–412. IEEE, 1993.

[235] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. *NeurIPS*, 2018.

[236] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. Hidden technical debt in machine learning systems. In *NeurIPS*, pages 2503–2511, 2015.

[237] Bruno Sena, Ana Paula Allian, and Elisa Yumi Nakagawa. Characterizing big data software architectures: a systematic mapping study. In *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*, pages 1–10, 2017.

[238] Bruno Sena, Lina Garcés, Ana Paula Allian, and Elisa Yumi Nakagawa. Investigating the applicability of architectural patterns in big data systems. In *Conference on Pattern Languages of Programs*, pages 1–15, 2018.

[239] Alex Serban and Joost Visser. Adapting software architectures to machine learning challenges. In *IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 2022.

[240] Alex Serban, Erik Poll, and Joost Visser. Deep repulsive prototypes for adversarial robustness. *arXiv:1903.08428*, 2019.

[241] Alex Serban, Erik Poll, and Joost Visser. A standard driven software architecture for fully autonomous vehicles. *Journal of Automotive Software Engineering*, 1:20–33, 2020.

[242] Alex Serban, Erik Poll, and Joost Visser. Adversarial examples on object recognition: a comprehensive survey. *ACM CSUR*, 53(3):1–38, 2020.

[243] Alex Serban, Erik Poll, and Joost Visser. Learning to learn from mistakes: Robust optimization for adversarial noise. In *International Conference on Artificial Neural Networks*, pages 467–478. Springer, 2020.

[244] Alex Serban, Erik Poll, and Joost Visser. Towards using probabilistic models to design software systems with inherent uncertainty. In *European Conference on Software Architecture*, pages 89–97. Springer, 2020.

[245] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. Adoption and effects of software engineering best practices in machine learning. *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 1–12, 2020.

[246] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. *Adoption and Effects of Software Engineering Best Practices in Machine Learning - Supplementary Material*. Zenodo, Jul 2020. doi: 10.5281/zenodo.3946453.

[247] Alex Serban, Koen van der Blom, Holger Hoos, and Joost Visser. Practices for engineering trustworthy machine learning applications. In *IEEE Workshop on AI Engineering WAIN'21@ICSE'21*. IEEE, 2021.

[248] Alexandru Constantin Serban. Designing safety critical software systems to manage inherent uncertainty. In *IEEE International Conference on Software Architectue (ICSA)*, pages 246–249. IEEE, 2019.

[249] Alexandru Constantin Serban and Erik Poll. Adversarial examples - a complete characterisation of the phenomenon. *arXiv:1810.01185*, 2018.

[250] Alexandru Constantin Serban, Erik Poll, and Joost Visser. A security analysis of the etsi its vehicular communications. In *International Conference on Computer Safety, Reliability, and Security*, pages 365–373. Springer, 2018.

[251] Alexandru Constantin Serban, Erik Poll, and Joost Visser. A standard driven software architecture for fully autonomous vehicles. In *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pages 120–127. IEEE, 2018.

[252] Alexandru Constantin Serban, Erik Poll, and Joost Visser. Tactical safety reasoning. a case for autonomous vehicles. In *IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2018.

[253] Roman Seyffarth. Machine learning: Moving from experiments to production. https://blog.codecentric.de/en/2019/03/machine-learning-experiments-production/, 2019. [Online; accessed 22-03-2021].

[254] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *NeurIPS*, 32, 2019.

[255] Mojtaba Shahin, Muhammad Ali Babar, and Liming Zhu. Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. *IEEE Access*, 5:3909–3943, 2017.

[256] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[257] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *SIGSAC*, pages 1528–1540. ACM, 2016.

[258] David Silver and Joel Veness. Monte-carlo planning in large POMDPs. In *NeurIPS*, pages 2164–2172, 2010.

[259] Leslie N Smith. Cyclical learning rates for training neural networks. In *IEEE WACV*, pages 464–472. IEEE, 2017.

[260] Leslie N Smith. A useful taxonomy for adversarial robustness of neural networks. *arXiv:1910.10679*, 2019.

[261] Trey Smith and Reid Simmons. Heuristic search value iteration for POMDPs. In *UAI*, pages 520–527, 2004.

[262] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017.

[263] Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. PixelDefend: Leveraging generative models to understand and defend against adversarial examples. *ICLR*, 2018.

[264] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy?–a comprehensive study on the robustness of 18 deep image classification models. In *ECCV*, pages 644–661, 2018.

[265] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5): 828–841, 2019.

[266] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NeurIPS*, pages 1057–1063, 2000.

[267] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv:1312.6199*, 2013.

[268] Nisha Talagala. Operational machine learning. https://www.kdnuggets.com/2018/04/operational-machine-learning-successful-mlops.htmls, 2018. [Online; accessed 22-03-2021].

[269] Antony Tang, Ann Nicholson, Yan Jin, and Jun Han. Using bayesian belief networks for change impact analysis in architecture design. *Journal of Systems and Software*, 80(1):127–148, 2007.

[270] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1013–1020. IEEE, 2018.

[271] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[272] Simen Thys, Wiebe Van Ranst, and Toon Goedemé. Fooling automated surveillance cameras: adversarial patches to attack person detection. *CVPR Workshops*, 2019.

[273] Luís Torgo, Rita P Ribeiro, Bernhard Pfahringer, and Paula Branco. Smote for regression. In *Portuguese conference on artificial intelligence*, pages 378–389. Springer, 2013.

[274] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv:1705.07204*, 2017.

[275] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *arXiv:2002.08347*, 2020.

[276] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *ICLR*, 2018.

[277] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *ICLR*, 2019.

[278] Leslie G. Valiant. A theory of the learnable. In *ACM Symp. on Theory of Computing*. ACM, 1984.

[279] Tom van der Weide, Dimitris Papadopoulos, Oleg Smirnov, Michal Zielinski, and Tim van Kasteren. Versioning for end-to-end machine learning pipelines. In *Proceedings of the 1st Workshop on Data Management for End-to-End Machine Learning*, pages 1–9, 2017.

[280] Nikos Vlassis, Michael L. Littman, and David Barber. On the computational complexity of stochastic controller optimization in POMDPs. *ACM Trans. on Computation Theory*, 4(4):12:1–12:8, 2012.

[281] Erwin Walraven and Matthijs Spaan. Accelerated vector pruning for optimal POMDP solvers. In *AAAI*, pages 3672–3678, 2017.

[282] Zhiyuan Wan, Xin Xia, David Lo, and Gail C Murphy. How does machine learning change software development practices? *IEEE Transactions on Software Engineering*, 2019.

[283] Heng Wang, Bin Wang, Bingbing Liu, Xiaoli Meng, and Guanghong Yang. Pedestrian recognition and tracking using 3d lidar for autonomous vehicle. *Robotics and Autonomous Systems*, 88:71–78, 2017.

[284] Jigang Wang, Predrag Neskovic, and Leon N Cooper. Pattern classification via single spheres. In *International Conference on Discovery Science*, pages 241–252. Springer, 2005.

[285] Yue Wang, Swarat Chaudhuri, and Lydia E. Kavraki. Bounded policy synthesis for pomdps with safe-reachability objectives. In *AAMAS*, pages 238–246. Int'l Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM, 2018.

[286] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. Studying software engineering patterns for designing machine learning systems. In *10th International Workshop on Empirical Software Engineering in Practice (IWESEP)*, pages 49–495. IEEE, 2019.

[287] Hironori Washizaki, Hiromu Uchida, Foutse Khomh, and Yann-Gaël Guéhéneuc. Machine learning architecture and design patterns. http://www.washi.cs.waseda.ac.jp/wp-content/uploads/2019/12/IEEE_Software_19__ML_Patterns.pdf, 2019. [Online; accessed 22-03-2021].

[288] Xingxing Wei, Siyuan Liang, Xiaochun Cao, and Jun Zhu. Transferable adversarial attacks for image and video object detection. *arXiv:1811.12641*, 2018.

[289] Danny Weyns. Software engineering of self-adaptive systems. In *Handbook of Software Engineering*, pages 399–443. Springer, 2019.

[290] Danny Weyns, Nelly Bencomo, Radu Calinescu, Javier Cámara, Carlo Ghezzi, Vincenzo Grassi, Lars Grunske, Paola Inverardi, Jean-Marc Jezequel, Sam Malek, et al. Perpetual assurances for self-adaptive systems. In *Software Engineering for Self-Adaptive Systems III. Assurances*, pages 31–63. Springer, 2017.

[291] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *ICML*, pages 3751–3760, 2017.

[292] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Solving deep memory POMDPs with recurrent policy gradients. In *ICANN*, pages 697–706. Springer, 2007.

[293] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Minimal counterexamples for linear-time probabilistic verification. *Theoretical Computer Science*, 549:61–100, 2014.

[294] Rüdiger Wirth and Jochen Hipp. Crisp-dm: Towards a standard process model for data mining. In *Proceedings of the 4th international conference on the practical applications of knowledge discovery and data mining*, pages 29–39. Springer, 2000.

[295] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[296] Eric Wong and J. Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. *ICML*, pages 5286–5295, 2018.

[297] Eric Wong, Frank R. Schmidt, and J. Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. *arXiv:1902.07906*, 2019.

[298] Eric Wong, Leslie Rice, and J Zico Kolter. Fast is better than free: Revisiting adversarial training. *ICLR*, 2020.

[299] Eoin Woods. Operational: The forgotten architectural view. *IEEE Software*, 33(3): 20–23, 2016.

[300] Eoin Woods. Software architecture in a changing world. *IEEE Software*, 33(6):94–97, 2016.

[301] Carole-Jean Wu, David Brooks, Kevin Chen, et al. Machine learning at Facebook: Understanding inference at the edge. In *International Symposium on High Performance Computer Architecture*, pages 331–344. IEEE, 2019.

[302] Cihang Xie, Mingxing Tan, Boqing Gong, Alan Yuille, and Quoc V Le. Smooth adversarial training. *arXiv preprint arXiv:2006.14536*, 2020.

[303] Chengxiang Yin, Jian Tang, Zhiyuan Xu, and Yanzhi Wang. Adversarial meta-learning. *arXiv:1806.03316*, 2018.

[304] Haruki Yokoyama. Machine learning system architectural pattern for improving operational stability. In *International Conference on Software Architecture Companion (ICSA-C)*, pages 267–274. IEEE, 2019.

[305] A Steven Younger, Sepp Hochreiter, and Peter R Conwell. Meta-learning with backpropagation. In *IJCNN*, volume 3. IEEE, 2001.

[306] Dinghuai Zhang, Tianyuan Zhang, Yiping Lu, Zhanxing Zhu, and Bin Dong. You only propagate once: Accelerating adversarial training via maximal principle. *NeurIPS*, 2019.

[307] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *ICML*, pages 7472–7482, 2019.

[308] Hongyu Zhang and Stan Jarzabek. A bayesian network approach to rational architectural design. *International Journal of Software Engineering and Knowledge Engineering*, 15(04):695–717, 2005.

[309] Huan Zhang, Hongge Chen, Zhao Song, Duane Boning, Inderjit S Dhillon, and Cho-Jui Hsieh. The limitations of adversarial training and the blind-spot attack. *ICLR*, 2019.

[310] Ji Zhang and Sanjiv Singh. LOAM: LIDAR odometry and mapping in real-time. In *Robotics: Science and Systems*, 2014.

[311] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[312] Xufan Zhang, Yilin Yang, Yang Feng, and Zhenyu Chen. Software engineering practice in the development of deep learning applications. *arXiv:1910.03156*, 2019.

[313] Martin Zinkevich. Rules of machine learning: Best practices for ML engineering. https://developers.google.com/machine-learning/guides/rules-of-ml, 2019. [Online; accessed 22-03-2021].

[314] Martin Zinkevich. Rules of machine learning: Best practices for ML engineering. https://developers.google.com/machine-learning/guides/rules-of-ml/, 2020. [Online; accessed 22-03-2021].