Safer Learning-enabled Autonomous Systems

by

Seyed Sepehr Sharifi

Thesis submitted to the Faculty of Engineering
In partial fulfillment of the requirements
For the Doctorate of Philosophy degree in
Digital Transformation and Innovation

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

The safety of learning-enabled autonomous systems is critical for their reliable operation in real-world applications. However, ensuring the safety of learned components in these systems is highly challenging due to the complexity of their behaviors and interactions with the system's operational contexts. This thesis presents two contributions addressing key challenges in learning-enabled autonomous system safety.

First, we propose MLCSHE (ML Component Safety Hazard Envelope), a novel method to identify the hazard boundary of learned components in a learning-enabled autonomous system. MLCSHE uses a Cooperative Co-Evolutionary Algorithm (CCEA) to decompose the high-dimensional problem space into manageable subproblems, efficiently exploring safe and unsafe regions. Additionally, we introduce a probabilistic view of the hazard boundary, incorporating a custom fitness function that drives the search. Empirical evaluation on an Autonomous Vehicle case study shows that MLCSHE outperforms traditional methods like genetic algorithms in both effectiveness and efficiency.

Second, we address the challenge of developing real-time safety monitors for learned components. We propose a method based on Deep Learning (DL)-based probabilistic time series forecasting to predict safety violations given the operational context. Through empirical evaluation on autonomous aviation and autonomous driving case studies, we assess the accuracy, latency, and resource usage of several forecasting models. Among them, the Temporal Fusion Transformer (TFT) demonstrates superior performance in predicting imminent safety violations while maintaining acceptable computational overhead.

Together, these contributions advance the state of the art in safety assurance of learning-enabled autonomous systems, facilitating their safer deployment in complex environments. Finally, the thesis discusses the larger practical and policy implications of the proposed methods and provides future directions for research.

Acknowledgements

I am very grateful to Auxon Corporation for their financial support and to Zachary Pierce for his insightful feedback. I would also like to thank Nathan Aschbacher and Frederic Risacher for their constructive feedback on the work from a practical point-of-view. This work was partially supported by funding from the Natural Sciences and Engineering Research Council of Canada (NSERC), through the Canada Research Chairs and Discovery programs, the Science Foundation Ireland grant 13/RC/2094-2, Ontario Graduate Scholarship (OGS), and Mitacs Accelerate Program. This research was enabled in part by support provided by British Columbia Digital Research Infrastructure (`https://www.bc.net`), Compute Ontario (`https://www.computeontario.ca`), and the Digital Research Alliance of Canada (`https://alliancecan.ca`).

I additionally would like to thank Corina Păsăreanu for her feedback in the early stages of the safety monitoring reserach and her pointer to the TinyTaxiNet model.

I am very grateful to Donghwan Shin and Andrea Stocco, who have helped me greatly along the way, especially when faced with challenges during the course of my research.

I am also thankful to the members of my Thesis Advisory Committee, Daniel Amyot and Jason Millar, whose valuable feedback and continued support have improved my thesis significantly.

My heartfelt thanks go to my supervisor, Lionel Briand, for their unwavering support and patience throughout this journey. Their critical insights and constructive criticism have been pivotal in the completion of this work.

Dedication

This dissertation is dedicated to my partner and best friend, Anahita, who has been a unwavering source of support and encouragement during the challenges of graduate school and life. I am truly grateful for having you in my life.

This work is also dedicated to my parents, Hamid and Minoo, who have always loved me unconditionally.

Moreover, this work is dedicated to the many individuals who have shaped the trajectory of my life and from whom I have had the privilege of learning.

Finally, as Steve Jobs famously said, this work is dedicated to the *crazy ones*—those who motivated the author to make the hubristic choice of trying to make a meaningful difference, however small, in the world.

Table of Contents

List of Tables viii

List of Figures ix

Glossary of Acronyms xi

1 Introduction 1

1.1 Motivation . . . 1
1.2 Thesis Objectives . . . 2
1.3 Research Methodology . . . 3
1.3.1 DSR's Conceptual Framework . . . 3
1.3.2 Relevance . . . 4
1.3.3 Research Process . . . 5
1.3.4 Evaluation . . . 5
1.4 Contributions . . . 7
1.5 Publications . . . 8
1.6 Thesis Structure . . . 8

2 Hazard Boundary Identification 10

2.1 Overview . . . 10
2.2 Background . . . 12
2.3 Problem and Challenges . . . 14
2.3.1 Problem Definition . . . 14
2.3.2 Challenges . . . 16
2.4 Related Work . . . 17
2.4.1 Search-based Methods . . . 17

2.4.2 Sampling-based Methods . . . 17
2.4.3 Formal Methods . . . 18
2.4.4 Remark on Differences in Objectives . . . 19
2.5 Our Approach . . . 20
2.5.1 Representations . . . 20
2.5.2 Fitness Function . . . 22
2.5.3 MLC Systemic Hazard Envelope (MLCSHE) Algorithm . . . 25
2.6 Evaluation . . . 30
2.6.1 Evaluation Subjects . . . 31
2.6.2 RQ_1: Effectiveness . . . 33
2.6.3 RQ_2: Efficiency . . . 37
2.6.4 Discussion . . . 38
2.6.5 Data Availability . . . 41
2.7 Conclusion . . . 41

3 Safety Monitoring **43**
3.1 Overview . . . 43
3.2 Background . . . 44
3.2.1 Time Series Forecasting . . . 44
3.2.2 DL-based Forecasting Architectures . . . 45
3.3 Problem and Challenges . . . 46
3.3.1 Problem Definition . . . 46
3.3.2 Challenges . . . 48
3.4 Related Work . . . 48
3.4.1 Black-box Methods . . . 49
3.4.2 White-box Methods . . . 49
3.4.3 Limitations of Existing Methods . . . 50
3.5 Temporal Forecasting of Safety Metrics . . . 50
3.6 Empirical Evaluation . . . 53
3.6.1 Evaluation Subject . . . 54
3.6.2 Models Under Evaluation . . . 58
3.6.3 RQ_1: Safety Metric Forecast Accuracy . . . 61
3.6.4 RQ_2: Safety Violation Prediction Accuracy . . . 67

3.6.5 RQ_3: Prediction Accuracy Sensitivity Analysis 71
3.6.6 RQ_4: Latency and Memory Overhead 78
3.6.7 Discussion 81
3.6.8 Threats to Validity 83
3.6.9 Data Availability 85
3.7 Conclusion 86

4 Discussion **90**
4.1 Industry Considerations 90
4.2 Regulatory and Societal Considerations 92

5 Conclusion **96**
5.1 Summary of Contributions 96
5.2 Future Work 97

References **101**

List of Tables

1.1 Seven DSR guidelines [48] 4

2.1 *DBS* values for different search methods at different values of t_b and d_{th}. . 35

2.2 Statistical comparison of DBS values for different search methods at different values of t_b and d_{th}. Comparisons with no results are specified as N/A. Such cases happen when A or B have no samples to compare. 37

3.1 Hyperparameters of the models under evaluation, for ***cte*** and ***he*** safety requirements of the ACT case study, and the ***cte*** requirement of the ADS case study 60

3.2 q-Risk values for different models and quantiles, for the ***cte*** and ***he*** safety requirements of the ACT case study and the ***cte*** safety requirement of the ADS case study, respectively. 63

3.3 Statistical comparison of q-Risk values for different DL-based forecasters at different quantiles q. 65

3.4 Various safety violation prediction accuracy metric values for different models and quantiles, for ***cte*** and ***he*** (rows highlighted in gray) safety requirements. 87

3.5 Various safety violation prediction accuracy metric values for different models and quantiles, for the ADS case study. 88

3.6 Statistical comparison of F_3 score values for different DL-based forecasters at quantiles q$\geq$ 0.5, for ***cte*** and ***he*** safety requirements of the ACT case study and the ***cte*** safety requirement of the ADS case study, respectively. . 89

5.1 Thesis objective-contribution traceability matrix 97

List of Figures

1.1 The DSRM process model (based on the process proposed by Peffers et al. [104]). 6

2.1 An abstract coevolutionary algorithm (CCEA). 13

2.2 An illustration of safe and unsafe regions and the corresponding hazard boundary. 15

2.3 A possible application of *DeepJanus* to the systemic hazard boundary detection problem. Connected dots are a safe and unsafe pair. 18

2.4 The scenario domain model for the running example. The model is based on the concepts provided in the CARLA World domain model [21, 33]. The scenario parameters are shown on the figure in bold font, i.e., the weather preset, the attributes of Mission and the number of actors such as Vehicles and Pedestrians. 23

2.5 The relationship between d_{th} (distinctiveness threshold) and *DBS* (distinct boundary solutions) along with their confidence intervals (shown as error bars) for MLCSHE, GA, and RS for different t_b (boundary closeness threshold) values. 36

2.6 Plots of *DBS* vs. % *simulation budget* for MLCSHE, GA, and RS, with $d_{th} \in \{0.1, 0.2, 0.3\}$ and $t_b \in \{0.03, 0.05, 0.15\}$. 39

3.1 The overall process for training the temporal safety metric forecaster for safety monitoring. 52

3.2 System-in-the-loop simulation of the ACT system and the X-Plane simulator (a), and the ADS system and the Udacity simulator (b). 55

3.3 Average q-Risk values and their corresponding 95% confidence interval ($CI_{0.95}$) for all models over all reported quantiles, for the ***cte*** and ***he*** safety requirements of the autonomous taxiing (ACT) case study and the ***cte*** safety requirement of the autonomous driving (ADS) case study, respectively. Note that the x-axis is *not* drawn to scale, in favor of a more readable presentation. 64

3.4 False Negatives (FN) and False Positives (FP) at different prediction quantiles (q), for ***cte*** and ***he*** safety requirements of the ACT case study and the ***cte*** safety requirement of the ADS case study, respectively. Note that the x-axis is *not* drawn to scale, in favor of a more readable presentation. . . . 69

3.5 Safety metric prediction accuracy metrics for various window configurations of DeepAR, MQCNN, Seq2Seq and TFT, for the *cte* and *he* safety requirements, respectively. 73

3.6 Precision score measurements for DeepAR, MQCNN, Seq2Seq, and TFT, in various window configurations, for the *cte* (a) and *he* (b) safety requirements. 75

3.7 Recall score measurements for DeepAR, MQCNN, Seq2Seq, and TFT, in various window configurations, for the *cte* (a) and *he* (b) safety requirements. 76

3.8 F_3 score measurements for DeepAR, MQCNN, Seq2Seq, and TFT, in various window configurations, for the *cte* (a) and *he* (b) safety requirements. . . . 77

3.9 The plot for a) model memory usage, b) peak inference memory usage, and c) average inference latency for DeepAR, MQCNN, Seq2Seq, and TFT models at different window configurations, for the *cte* safety requirement of the ACT case study, which is similar to the results for the *he* safety requirement. The above results at forecast horizon of 3 s and $cm = 1$ are also similar to the results for the ADS case study. 80

Glossary of Acronyms

ACT	Autonomous Centerline Tracking
ADAS	Advanced Driver Assistance System
ADS	Autonomous Driving System
AMS	Adaptive Multi-Splitting
ANSI	American National Standards Institute
ARIMA	AutoRegressive Integrated Moving Average
AV	Autonomous Vehicle
CCEA	Cooperative Co-Evolutionary Algorithm
CI	Confidence Interval
CNN	Convolutional Neural Network
CPS	Cyber Physical System
CPU	Central Processing Unit
CTE	Cross-Track Error
DBS	Distinct Boundary Solutions
DOI	Document Object Identifier
DL	Deep Learning
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
DSR	Design Science Research
DSRM	Design Science Research Methodology
EA	Evolutionary Algorithm

FN	False Negative
FP	False Positive
FTA	Fault Tree Analysis
GA	Genetic Algorithm
GLM	Generalized Linear Model
GPS	Global Positioning System
GPU	Graphic Processing Unit
GRU	Gated Recurrent Unit
HE	Heading Error
IEC	International Electrotechnical Commission
ISO	International Organization for Standardization
KPI	Key Performance Indicator
LHS	Latin Hypercube Sampling
LLM	Large Language Model
LSTM	Long Short-Term Memory
ML	Machine Learning
MLAS	Machine Learning Autonomous System
MLC	Machine Learning Component
MLCO	Machine Learning Component Output
MLCSHE	Machine Learning Component Hazard Envelope
NBS	Neural Bridge Sampling
ODD	Operational Design Domain
OOD	Out-Of-Distribution
PAF	Performance Asssessment Function
RAM	Rapid Access Memory
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network

RQ	Research Question
RS	Random Search
RTA	Run Time Assurance
TFT	Temporal Fusion Transformer
TO	Thesis Objective
UAS	Unmanned Aircraft System
VAE	Variational Auto-Encoder
XAI	eXplainable Artificial Intelligence

Chapter 1

Introduction

1.1 Motivation

Autonomous systems are increasingly empowered by being embedded with learned components for various tasks, such as perception, localization, prediction, planning and control. These components are inherently different from conventional software components and pose new challenges and safety risks that are not manageable by traditional software engineering practices. The main reason for this difference is that these components' logic is not captured by source code or specifications but their behavior is rather learned through training. Therefore, ensuring the safety and reliability of such systems through conventional software engineering practices is inadequate. These risks are particularly acute when learning-enabled autonomous systems [1] are employed in safety-critical applications, e.g., autonomous driving [16], autonomous aviation [64], medical diagnosis [154] or disease prediction [155], as failures could directly jeopardize human safety. These systems have already contributed — if not directly led — to fatalities in the case of Autonomous Vehicles (AVs) [11, 96].

Recent efforts have focused on making learned components more reliable, robust and accurate through novel testing methods [50, 151]. However, even a system with reliable components is still prone to accidents [73]. For example, some accidents are caused as a result of unsafe component interactions [2, 73]. Thus, the impact of learned components on safety can only be studied in the context of the system they are integrated into and in a specific operational context [14, 73].

The inherent specificity of learned components favors the use of safety monitors (also known as Run Time Assurance or RTA mechanisms) [128]. Safety monitors, at run time, check the inputs and outputs of a component that cannot be fully trusted, e.g., a learned component, and will block its outputs from being propagated to the rest of the system if they are potentially hazardous. In such cases, systems usually fall back on a trustworthy

[1] In this thesis, we sometimes refer to learned components as Machine Learning Components (MLCs), and to learning-enabled autonomous systems as ML-enabled Autonomous Systems (MLASs), as they are interchangeable in our view.

but less efficient component [6], or take any other pre-designed fallback mechanisms, such as stopping the AV on the shoulder of the road. Runtime safety monitors observe the system, its operational context, and the inputs and outputs of a component that cannot be fully trusted, such as a learned component, predicting if its outputs may lead the system toward a safety requirement violation. For example, misclassification of an AV's object detection component might not lead to any hazards under a certain system context (henceforth called a *scenario*), e.g., when an AV misidentifies an animal crossing the road as a pedestrian and stops. Thus, identifying the combinations of system contexts (i.e., scenarios) and learned components' behaviors (i.e., inputs and outputs) that will transition the system to a hazard state is an essential step in developing safety monitors to be able to ensure the safety of the learning-enabled system.

However, finding the hazardous combinations of the learned components' behaviors and operational scenarios before system deployment is challenging. First, the problem space of scenarios and learned components' behaviors is vast and high-dimensional, making it difficult for conventional search methods like Genetic Algorithms (GA) to be effective. Second, determining safety violations requires running the system within its operational environment, which involves computationally expensive simulations. The combination of a large problem space and high computational cost exacerbates the challenge. Last, environmental parameters, such as the random trajectory of pedestrians in simulators like CARLA [33] for AVs, cannot always be controlled, and the non-linear behavior of ML models like Deep Neural Networks (DNNs) means that similar inputs can lead to vastly different outputs. As a result, we cannot assume uniform safety outcomes across a region of the problem space, making it difficult to establish clear boundaries between safe and unsafe regions.

Monitoring for hazardous combinations of operational scenarios and learned component outputs during system operation presents additional challenges. First, system integrators often lack access to the training or test data and detailed internal information of third-party learned components. Second, safety monitors must track not only the outputs of these components but also the operational context, including static parameters like weather and dynamic ones like nearby vehicle trajectories. Last, in safety-critical situations, the monitor must predict safety violations with enough lead time for corrective actions to be effective, necessitating efficient monitors with low latency that can operate within the constraints of onboard computing resources.

The safety-critical nature of the rapidly proliferating learning-enabled autonomous systems, coupled with the numerous challenges in identifying and monitoring hazardous combinations of operational scenarios and learned component outputs (*hazard conditions*), underscores the urgent need for research.

1.2 Thesis Objectives

The long-term vision of this research is provide accessible ways of designing and developing safer autonomous systems with learned components. Thus, given the motivation above (section 1.1), the overall goal of this thesis is as follows:

" Identify and monitor the hazard conditions for the learned components of an autonomous system based on system-level safety requirements."

To address the above goal we further refine them to the following two main Thesis Objectives (TO):

TO_1 (Hazard Boundary Identification): Identify the combinations of system contexts and learned components' behaviors that lead the system to a hazard state, i.e., the hazard boundary of learned components.

TO_2 (Safety Monitoring): Monitor the system during operation (at runtime) and predict if the behavior of learned components can lead to system safety violations (i.e., hazard) given its operational context.

1.3 Research Methodology

To address reach the objectives set out in section 1.2 in this thesis, we use a research methodology that is heavily influenced by the *Design Science Research* (DSR) approach in *Information Science* (IS), initially presented by Hevner et al. [48]. DSR integrates both social and design aspects within information science, as well as in other engineering and computer science disciplines.

This section begins by outlining the conceptual framework of DSR, followed by its specific application to this thesis.

1.3.1 DSR's Conceptual Framework

The DSR methodology is aimed at the IS field with the goal of designing and evaluating artifacts to address specific problems. It deals with the complexity inherent in IS research, encompassing both social (natural) science and design science elements. Hevner et al. identified three primary cycles in a DSR-based approach [48]: First, the research should address significant business needs while producing outcomes relevant to the domain (*Relevance Cycle*). Second, the research should be grounded by solid theoretical foundations from the existing knowledge base while its findings contribute to this base (*Rigor Cycle*). Finally, the research should iteratively design, develop and evaluate artifacts, whereby the evaluation results inform the design phase (*Design Cycle*).

Based of the above cycles, Hevner et al. provide guidelines, which are summarized in Table 1.1 [48]. The research described in this thesis has adhered to these guidelines to enhance the potential for producing meaningful and practical results by carrying out the research.

In this thesis, we present multiple research and artifact contributions (G1 and G4 in Table 1.1) which we discuss in section 1.4. We discuss the relevance of the problem

Table 1.1: Seven DSR guidelines [48]

Guideline	Description
G1: Design an artifact	DSR should produce a practical artifact, which can be a construct, model, method, or instantiation.
G2: Problem relevance	The aim of DSR is to create technology-based solutions to significant and relevant business issues.
G3: Design evaluation	The utility, quality, and effectiveness of a design artifact must be rigorously demonstrated through well-conducted evaluation methods.
G4: Research contributions	Effective DSR must offer clear and verifiable contributions in terms of the design artifact, design foundations, and/or design methodologies.
G5: Research rigor	DSR relies on the application of rigorous methods in both the design artifact creation and evaluation.
G6: Design as a search process	The search for an effective artifact requires utilizing available resources to achieve desired outcomes while adhering to constraints in the problem environment.
G7: Communication of research	DSR must be effectively communicated to both technology- and management-oriented audiences.

addressed by this research (G2 in Table 1.1) in subsection 1.3.2. We discuss the design-based research process (G6 in Table 1.1), that we used for this research, in subsection 1.3.3. We outline the rigorous evaluation methods to evaluate the proposed solutions of this research (G3 and G5 in Table 1.1), in subsection 1.3.4. Finally, we list the publications that have resulted from this research (G7 in Table 1.1) in section 1.5.

1.3.2 Relevance

Part of the problem relevance was motivated in section 1.1, according to guideline G2 in Table 1.1. Further evidence of relevance was provided through collaboration with industrial and research partners:

- Over the course of this research, we collaborated with a North American startup developing automated tools to test and analyze complex AI-enabled systems. Furthermore, we collaborated with a large technology company which had also developed its own automated driving solution. Both have considered using the artifacts of this research in their solutions to test and monitor AI-enabled autonomous systems.
- We had many meetings and discussions with various Canadian and US companies involved in various domains such as aerospace, robotics, software safety verification

and validation, information technology and AI consulting, where the relevance of the problems targeted by our research was recognized as a major concern of the field.

- Recent legislation such as the European Union (EU) AI Act [27] and standardization efforts such the ones published or underway by International Standardization Committee (ISO) and International Electrotechnical Committee (IEC)'s Joint Technical Commission (JTC) ISO/IEC JTC 1/SC 42 [59], where one of their main areas of focus is safety and trustworthiness of AI-enabled (autonomous) systems, highlight the ongoing interest from the regulators and the need for the industry to better identify and manage safety risks of such systems, which is in line with the main objectives of this thesis.

- Events such as the accident involving a Cruise AV dragging a pedestrian leading to the revocation of Cruise's autonomous driving license in California [40] and multiple crashes involving Tesla's '*Full Self-Driving*' software which triggered a mass recall of about two million vehicles [56], further highlights the need for safer and more reliable learning-enabled autonomous systems.

1.3.3 Research Process

To ensure research rigor and approach design as a search process (guidelines G5 and G6 in Table 1.1), a research methodology, with a problem centered research starting point, was adapted from Design Science Research Methodology (DSRM) [104].

Figure 1.1, the DSRM process model, illustrates the research steps and feedback. The process starts with the problem definition stage, which is motivated by problems faced by industry and society. We then refine the problem into the high-level solution objectives which guide the design and development of the solution. We implement the solution and evaluate it which might trigger a re-design of the solution (a *design iteration*). After satisfaction with the evaluated solution, we communicate the evaluation results and the solution to the academic and industrial community, where their feedback could lead us to revise the solution objectives or the solution (*community feedback*). Through frequent communication with industry and academia, while the solution is being defined and developed, we have tried to minimize large objective redefinition or solution redesign efforts.

The problem was motivated in section 1.1, while section 1.2 describes the objectives of the solution. We have provided a detailed problem statement, outlined solution objectives, developed solutions to address TO_1 and TO_2, and have rigorously evaluated them in chapter 2 and chapter 3, respectively. The results of this research were communicated to the scientific community via various publications and presentations, as listed in section 1.5.

1.3.4 Evaluation

The evaluation of the artifacts, as pointed out by guideline G3 of Table 1.1, is performed primarily through rigorous empirical evaluations using various case studies.

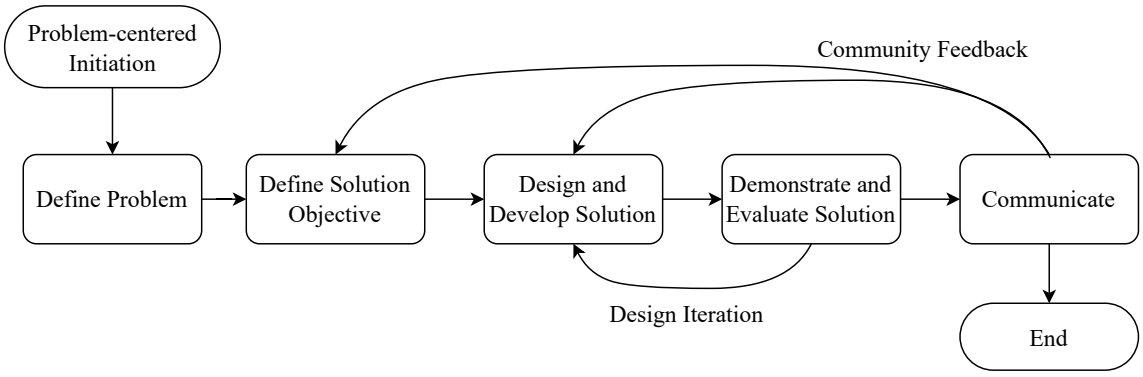


Figure 1.1: The DSRM process model (based on the process proposed by Peffers et al. [104]).

We have evaluated our proposed methods and tools addressing TO_1 and TO_2 by empirically evaluating them using case studies in autonomous driving and autonomous aviation. Each case study involves a realistic learning-enabled autonomous system and a high-fidelity simulator to generate realistic execution data of such systems in various scenarios. The details for each case study, including their detailed setup, can be found in chapter 2 and chapter 3. Note that although a single and shared running example and case study could have been used to answer both TO_1 and TO_2, we did not do so for two reasons. First, due to the exploratory nature of the research, i.e., we could not know, in advance, all of the data required for the safety monitoring method (chapter 3), while we were designing the experiments for the hazard boundary identification method (chapter 2). After identifying the required data to develop the safety monitor, we realized that data recorded from the simulations done in chapter 2 were missing the granular (per second samples) safety metric values required for the safety monitoring method proposed in chapter 3. As noted in chapter 2, the experiments took more than 75 days of parallel simulations which were prohibitively costly for us to re-run and record the necessary data. Second, by the time we were developing our safety monitoring methods, we had discovered newer case studies could generate data much faster than the one used in chapter 2, thus allowing us to save time in generating necessary data, while also increasing the diversity of the case studies addressed by this thesis.

During the experimental evaluations, we have used clearly defined research questions for the empirical evaluations, followed by a rigorous experiment methodology to answer each of them. We also have relied on applicable statistical methods in order to draw statistically sound conclusions from our experiment results. We have provided a detailed discussion of the threats to the validity of each of our experimental evaluations in their respective chapters, i.e, chapter 2 and chapter 3. Throughout our study, we have communicated with industry to receive feedback on the relevance of the research questions and the selected evaluation metrics, as well as with academia to receive feedback on the validity of the study design and execution. Finally, in chapter 4, we have discussed the potential impacts of our results in industry and society.

1.4 Contributions

The research contributions of this thesis include:

1. *Hazard Boundary Identification (TO1)*:
   (a) MLCSHE, a dedicated and tailored cooperative coevolutionary search approach to approximate the hazard boundary of an ML component, in a probabilistic way, taking into account the combination of scenarios and MLC behaviors.
   (b) An application of MLCSHE to a complex Autonomous Vehicle (AV) case study involving an industry-strength simulator and an Autonomous Driving System with deep learning components. Our implementation of MLCSHE as well as other case study artefacts are provided in our replication package (see subsection 2.6.5).
   (c) An empirical evaluation of the effectiveness and efficiency of MLCSHE through large scale experiments using CARLA, a high-fidelity open-source driving simulator, and Pylot, a high-performance open-source AV composed of multiple components.
   (d) A comparison of MLCSHE against baseline methods, namely random search (RS) and a vanilla genetic algorithm (GA).
2. *Safety Monitoring (TO2)*:
   (a) A safety monitoring method that leverages probabilistic time series forecasting of a safety metric to identify learned component behavior and system context that lead to system safety violations at runtime.
   (b) An application of the safety monitoring method to a widely used autonomous aviation case study, i.e., Autonomous Centerline Tracking (ACT), including a dataset generated from system-in-the-loop simulations. We have additionally applied our proposed method to another case study in the automated driving domain using a dataset available from the literature.
   (c) A large-scale empirical evaluation with state-of-the-art DL-based probabilistic forecasting models targeting safety metric and safety violation prediction accuracy, inference latency, and runtime resource usage.
   (d) A Python-based implementation of the models as well as detailed hyperparameter tuning and data generation instructions are provided (subsection 3.6.9).

Contributions 1(a)–1(d) address TO_1, whereas contributions 2(a)–2(d) address TO_2. This thesis also includes a discussion on potential practical and societal considerations of the proposed methods and tools in this thesis. While the author of this thesis is the main author and contributor to all of the above contributions, contributions 1(a)–1(d) benefited from the co-supervision of Donghwan Shin in terms of refining the idea, code contributions, empirical evaluation design and writing. The same contributions also benefited from

Nathan Aschbacher where he provided help with idea refinement and empirical evaluation design feedback, from an industry application point of view. Similarly, the work done related to contributions 2(a)–2(d) benefited from Andrea Stocco's guidance in terms of idea, empirical evaluation design, writing. It goes without saying that all the contributions above were supervised, guided, and reviewed by the supervisor of this thesis, Prof. Lionel Briand.

1.5 Publications

In addition to this thesis, the research results have been communicated to the public via the following publications:

1. **Sepehr Sharifi**, Donghwan Shin, Lionel C. Briand, Nathan Aschbacher, "Identifying the Hazard Boundary of ML-Enabled Autonomous Systems Using Cooperative Coevolutionary Search," in *IEEE Transactions on Software Engineering*, vol. 49, no. 12, pp. 5120-5138, Dec. 2023, doi: 10.1109/TSE.2023.3327575.
    - **Sepehr Sharifi**, Donghwan Shin, Lionel C. Briand, Nathan Aschbacher, "Journal-First Presentation: Identifying the Hazard Boundary of ML-Enabled Autonomous Systems Using Cooperative Coevolutionary Search," in *IEEE/ACM 46th International Conference on Software Engineering*, Journal-first track, Lisbon, Portugal, ACM, 2024.
2. **Sepehr Sharifi**, Andrea Stocco, and Lionel C. Briand, "System Safety Monitoring of Learned Components Using Temporal Metric Forecasting," in arXiv preprints, 2024, arXiv:2405.13254 (*Under Revision*).

The detailed contributions of each publication is listed in section 1.4, where contributions 1(a)–1(d) correspond to publication 1 and its presentation at the *International Conference on Software Engineering*; and contributions 2(a)–2(d) correspond to publication 2. We have provided a detailed statement of contributions of each co-author of the publications in section 1.4[2].

1.6 Thesis Structure

The rest of this thesis is structured as follows:

- chapter 2 addresses TO_1 by providing:
    - a background on Cooperative CoEvolutionary Algorithms (CCEAs), a search-based method based on which we propose our hazard boundary identification algorithm (section 2.2),

[2]The co-authors affirm their agreement with the statement of contributions, as outlined in a signed attestation submitted to the Department of Engineering.

  - a formal definition of the problem of identifying the ML-component hazard boundary for an ML-enabled autonomous system (section 2.3),
  - a review of the related work and their shortcomings in addressing the defined problem (section 2.4),
  - a cooperative coevolutionary algorithm to search for the hazard boundary (section 2.5), and
  - a thorough empirical evaluation of the proposed method while comparing it with two other comparable methods (section 2.6).

- chapter 3 addresses TO_2 by providing:
  - a background on time series forecasting (section 3.2),
  - a formal definition of the safety monitoring problem which is recast as a time series forecasting problem (section 3.3),
  - a review of the related safety monitoring literature and their shortcomings (section 3.4),
  - a safety monitoring solution based on probabilistic forecasting of the safety metric (section 3.5),
  - and empirical evaluation of the proposed method using state-of-the-art Deep Learning (DL)-based probabilistic forecasting models (section 3.6).

- chapter 4 discusses the broader considerations related to the outcomes of this research on industry and society.

- Finally, chapter 5 concludes and enumerates future work items.

Chapter 2

Hazard Boundary Identification

This chapter addresses TO_1, i.e., identifying the hazard boundary of a learned component embedded in an autonomous system. The contents of this chapter have been published in the Journal of *Transactions on Software Engineering (TSE)* [122].

2.1 Overview

As discussed in chapter 1, identifying the combinations of system contexts (i.e., scenarios) and ML component's behaviors (i.e., inputs and outputs) that will transition the system to a hazard state is an essential step in developing safety monitors to be able to ensure the safety of the ML-enabled system.

However, there are several challenges involved with identifying the hazard boundary. First, the problem space of scenarios and ML component behaviors is very large and high-dimensional and is thus a challenge for more conventional search metaheuristics such as Genetic Algorithms (GA). Second, the violation of a given safety requirement can only be determined if the system is executed within its operational environment, which involves computationally intensive simulations. The high computational cost, in addition to the large problem space, renders the problem even more challenging. Last but not least, while safety can only be evaluated by executing the system within an environment, there are many environmental parameters that cannot be controlled even via a high-fidelity simulator; for example, the trajectory of pedestrians in CARLA [33], a well-known AV simulator, is random. Furthermore, two similar ML-enabled Autonomous System (MLAS) inputs may generate largely different outputs due to the non-linear behavior of ML models, such as Deep Neural Networks (DNNs). Therefore, we *cannot* assume that all combinations of scenarios and ML component behaviors within a region of the problem space have a uniform safety outcome, i.e., the region is deterministically safe or unsafe. Consequently, it is difficult to define hard boundaries between safe and unsafe regions.

To address the aforementioned challenges, we propose MLCSHE (ML Component Safety Hazard Envelope), a novel Cooperative Co-Evolutionary Algorithm (CCEA)-based approach that efficiently searches the problem space by decomposing it into two sub-spaces

(one for scenarios and one for ML component behaviors) and parallelizing the search of subspaces while taking the joint contribution of both scenarios and ML component behaviors to the autonomous system safety into account. Moreover, instead of naively assuming that the hazard boundary is a clear line that exists between the safe and unsafe regions, we take a probabilistic view of the problem domain, i.e., at any point within the scenario and ML component behavior space, there is a probability of being safe. Based on this probabilistic lens, we present a novel fitness function that effectively guides the search towards the "probabilistic" hazard boundary based on the probability of finding safe scenario-behavior pairs within a given region.

Contributions. The contributions of this work are summarized as follows:

- MLCSHE, a dedicated and tailored cooperative coevolutionary search approach to approximate the hazard boundary of an ML component, in a probabilistic way, taking into account the combination of scenarios and MLC behaviors.
- An application of MLCSHE to a complex Autonomous Vehicle (AV) case study involving an industry-strength simulator and an Autonomous Driving System with deep learning components. Our implementation of MLCSHE as well as other case study artefacts are provided in our replication package (see Section 2.6.5).
- An empirical evaluation of the effectiveness and efficiency of MLCSHE through large scale experiments using CARLA, a high-fidelity open-source driving simulator, and Pylot, a high-performance open-source AV composed of multiple components.
- A comparison of MLCSHE against baseline methods namely random search (RS) and vanilla genetic algorithm (GA).

Key Findings. The key findings of our empirical evaluation are summarized as follows:

- For reasonable boundary closeness thresholds given a search budget, MLCSHE is significantly more effective than RS and GA at detecting distinct boundary regions. This implies that a cooperative co-evolutionary algorithm makes the search for distinct boundary regions more effective than GA and RS.
- For reasonable boundary closeness thresholds given a search budget, MLCSHE finds significantly more diverse regions that overlap with the hazard boundary at a faster rate than GA and RS.

Chapter Structure. The rest of this chapter is structured as follows. Section 2.2 provides background materials on CCEA. Section 2.3 defines the problem of MLC hazard boundary identification and details its challenges. Section 2.4 discusses related work. Section 2.5 presents MLCSHE in detail. Section 2.6 provides the empirical evaluation of MLCSHE and discusses the results. Section 2.7 concludes and suggests future directions for research and improvement.

2.2 Background

In this section, an overview of Evolutionary Algorithms (EAs) is provided. Then we focus on a specific family of EAs, Cooperative Co-Evolutionary Algorithms (CCEAs), which happens to be particularly useful in our context. Finally, the key decision points involved in designing a CCEA, namely collaborator selection and individual fitness assessment are discussed.

EAs are a family of algorithms designed based on the principles of evolutionary computation [79]. EAs are inspired by the concepts related to biological evolution and have been applied to various optimization problems for which standard mathematical optimization is not applicable [79]. EAs use concepts such as *individual*, *population*, *fitness*, *selection* and *mutation* to formalize an optimization problem. Individuals usually represent solutions to the targeted problem and are members of a population whose fitness is evaluated (usually by a *fitness function*). Desirable individuals, i.e., those with the highest fitness values, are more likely to be selected to act as the parents of the next generation. Using methods such as *crossover* (replacing some parts of an individual with another one) and *mutation* (adding randomness to some parts of an individual), individuals of the next generation population, i.e., next iteration of the search, are created.

For many problems, the search space is high-dimensional such that a conventional EA would not be able to solve it within a reasonable timeframe [83]. To address this, Cooperative Coevolutionary Algorithms (CCEAs), originally proposed by Potter and De Jong [106] in 1994, decompose the original problem into lower-dimensional subproblems, each of which can be solved in a separately evolving population as in conventional EAs described previously. Since individuals from each subproblem population must join together to form a *complete solution* to the original problem, the fitness of an individual can only be evaluated based on the *joint fitness* of the complete solution created by joining the individual with representative individuals, called *collaborators*, from other populations. By carefully selecting collaborators and assessing individuals' fitness, CCEAs are known to be effective at solving even *non-separable* problems [101, 149] where the fitness of an individual of a subproblem population depends on the fitness of individuals of other populations. Furthermore, the decomposition of the original problem naturally allows parallelism to increase search performance [83].

Figure 2.1 depicts the process of an abstract CCEA. Each population is initialized, either with randomly selected or guessed values (usually provided by domain experts). Individuals of each population collaborate with individuals of the other population(s) to form complete solutions. Then, these complete solutions are evaluated via joint fitness assessment functions. The joint assessments are then aggregated to provide evaluations of individual fitness values. If the *stopping_condition* is reached (true), then the fittest individuals are returned. Otherwise, the individuals go through breeding (selection, crossover and mutation) to create the next generation of the populations and go through evaluations again.

Designing a CCEA includes two important decisions in the following aspects: *collaborator selection* and *individual fitness assessment*.

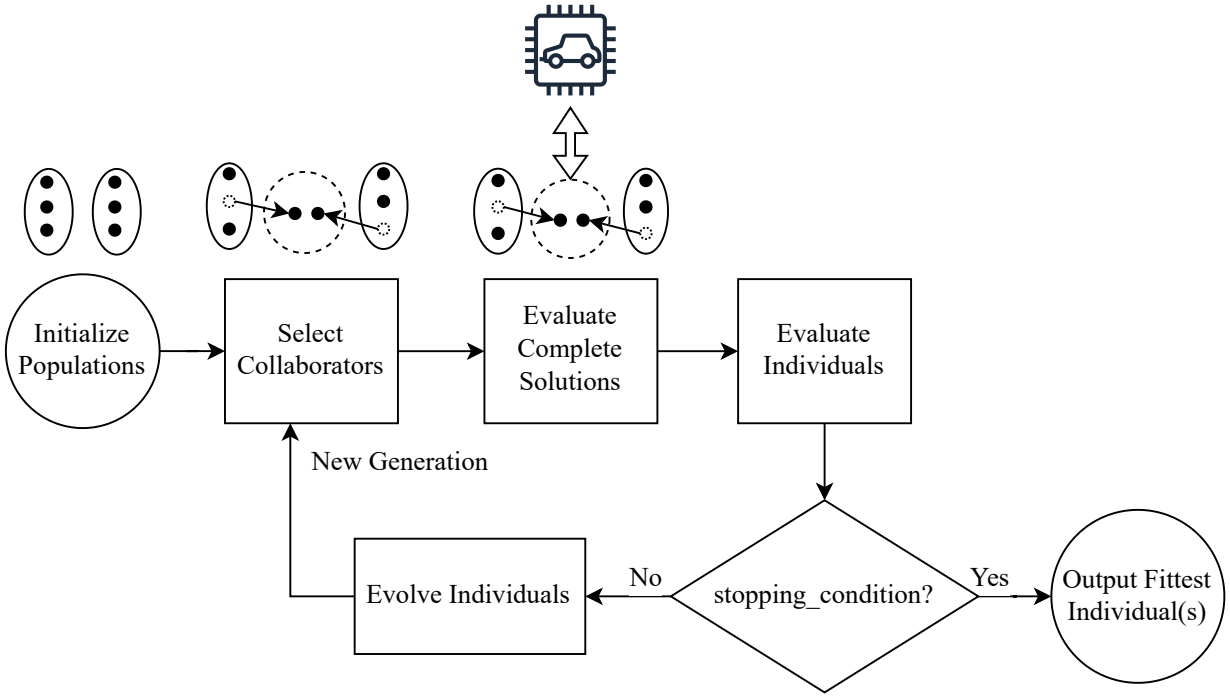


Figure 2.1: An abstract coevolutionary algorithm (CCEA).

Collaborator Selection. One of the most important factors affecting the performance of a CCEA is its *collaborator selection* strategy. To assess the fitness of the individual, the algorithm has to form one or multiple complete solutions with different collaborators. However, ideally, to get closer to the global optimum, all individuals of all other populations should be used as collaborators [100], which is usually infeasible due to resource constraints. Therefore, a strategy to efficiently select collaborators is required. Various strategies have been proposed in the literature, such as *single best*, *tournament-based*, and *random* [83]. These strategies affect the algorithm via controlling the selection pressure and the pool size of the collaborators.

Some studies have proposed *archive-based* collaborator selection to effectively reduce the number of collaborators to join in individual fitness assessments while maintaining the amount of information contained in the populations [83]. The idea is to carefully select a *population archive* which is a subset of a population to be used as collaborators. For example, Panait et al. [101], have proposed *iCCEA*, which aims to minimize the size of the population archives by considering only the collaborators that are *informative* and *distinct*. A collaborator in an archive is informative if adding it to the archive changes the fitness ranking of the population's individuals. If there are multiple collaborators that can change the ranking of the same individuals, the collaborator that changes the ranking the most will be kept in the archive. A collaborator in an archive is distinct if its (Euclidean) distance from other collaborators in the archive is higher than a pre-defined threshold. As a result, a population archive keeps only a minimum number of collaborators while attempting not to lose information in terms of collaborations between subproblem solutions. However, though the population archive selected by iCCEA is minimal in size, the algorithm proposed by

the authors to update the population archive in each generation has a high time complexity ($O(n^3)$ where n in the number of individuals in the archive) and this severely impacts the performance of the algorithm. Thus, simpler population archive selection methods, e.g., *elitist*, *random* and *best+random*, that are much faster, are also widely used in practice.

Individual Fitness Assessment. The *collaborator selection* strategy of a CCEA affects its *individual fitness assessment* strategies as well. The only *objective* fitness assessment that can be done on the individuals is based on their joint fitness assessments with collaborators. Thus, all algorithms perform some form of aggregation on joint fitness assessments related to an individual to determine its fitness value. *Best*, *worst* or *average* joint fitness values are usually used for individual fitness assessments.

2.3 Problem and Challenges

In this section, we provide a precise problem definition regarding the identification of the boundaries of hazard envelopes, focusing on the behavior of a Machine Learning Component (MLC). While we use an AV as an example, it can be easily generalised to any ML-based Autonomous System (MLAS).

2.3.1 Problem Definition

Consider an AV as an ML-enabled Autonomous System (MLAS) including an ML component (MLC), namely an image-based object detection component using DNNs. The AV continuously observes its surrounding environments—such as roads, traffic signs, buildings, and other moving vehicles via sensors (e.g., camera) and generates driving commands (e.g., steer left and decrease speed) to best satisfy given functional and safety requirements (e.g., reach a given destination point without colliding with other vehicles). During testing of the AV, the environment is often simulated by a high-fidelity driving simulator due to the high cost and risk of real-world testing. Inside of the AV, whenever new sensor data (e.g., an image taken from the camera) is collected, it passes through the object detection component to identify the positions of surrounding objects, if any, from the (fused) sensor data (e.g., in the form of bounding boxes in the given image), which will then be used to determine proper driving commands. Under a certain driving scenario, the AV might violate requirements such as "*the AV shall keep a minimum distance of 1.5 m from any vehicle in front.*" In such cases, the MLC could internally contribute to the violation. Therefore, identifying the boundaries of the hazard envelopes of the AV in terms of the combination of driving scenarios and MLC behaviors is important. Figure 2.2 provides a simplified illustration of an ML component's hazard envelope, defined in terms of scenarios and ML component behaviors, where a *safe* region leading to no violations is surrounded by an *unsafe* region leading to violations. The goal is to identify, as precisely and completely as possible, the boundaries between safe and unsafe regions, illustrated by the dashed line in Figure 2.2.

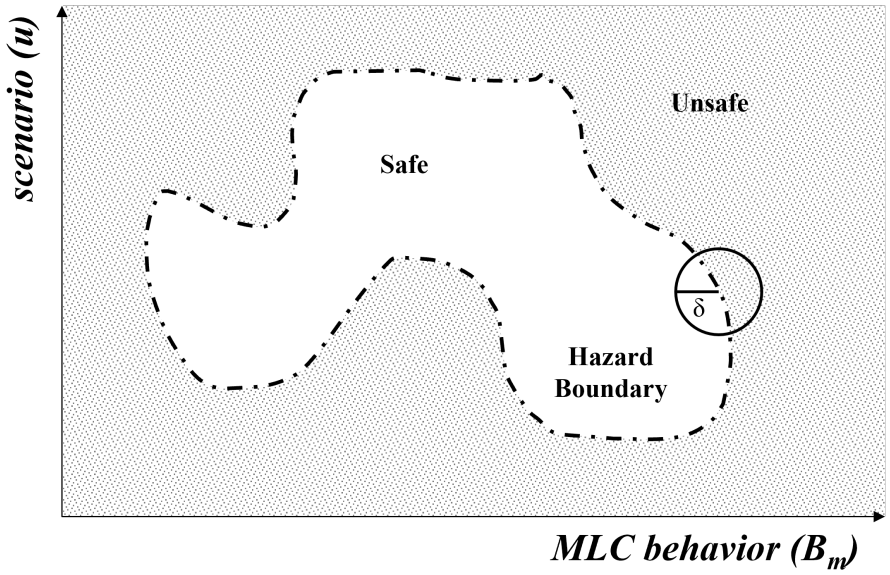


Figure 2.2: An illustration of safe and unsafe regions and the corresponding hazard boundary.

More specifically, let s be the AV including the MLC m for image-based object detection, operating in a simulated driving environment. For a given scenario u, which consists of all the static and dynamic entities of the environment such as road shape, weather, and other vehicles, the simulation result for s and m, denoted by $\Pi_{u,s,m}$, is a sequence $\langle e_1, e_2, \ldots, e_T \rangle$ where T is the duration of the execution and e_t for $t = 1, \ldots, T$ is the snapshot (state) of the environment at time step t. For each time $t \in \{1, \ldots, T\}$, s takes an image $in_{s,t}$ taken from the camera by observing e_t and generates a pre-processed (e.g., gray-scaled) image $in_{m,t}$ for m. Then, m produces the object information $out_{m,t}$ (in the form of bounding boxes) by processing $in_{m,t}$ and s produces driving commands $out_{s,t}$ by processing $out_{m,t}$[1]. The environment snapshot e_{t+1} for the next time step $t+1$ is updated based on $out_{s,t}$, and the whole process repeats until t reaches T. The behavior of m, denoted by B_m, is defined as the sequence of input/output pairs such that, for an input/output pair $(in_m, out_m) \in B_m$, out_m is the output produced by m by processing in_m. For a safety requirement r (e.g., do not collide with other vehicles), we can measure the degree of the safety violation (e.g., the distance to the other vehicles) of s for u and B_m in terms of r, denoted by $f(r, \Pi_{u,s,m})$, by analyzing $\Pi_{u,s,m} = \langle e_1, e_2, \ldots, e_T \rangle$ against r. If $f(r, \Pi_{u,s,m}) > \epsilon$ for a small threshold ϵ predefined for r, we say that r for s is violated by (the combination of) u and B_m. This means that we can decide the violation of r (with ϵ) given u and B_m.

Given the above context, let (u, B_m) be a point in a space referred to as the *input space*, that is defined by (the combination of) possible scenarios and MLC behaviors. For each point (u, B_m) in the input space, we can decide its output (i.e., unsafe or safe) by checking whether it leads to the violation of r or not. The identification of the boundaries

[1]Note that there can be other components in s that interact with m. For example, $out_{m,t}$ can be one of the (possibly many) factors that determine $out_{s,t}$. We only assume that m is one of the (possibly many) components required for s to operate; specifically, $in_{s,t}$ can affect $in_{m,t}$ and $out_{m,t}$ can affect $out_{s,t}$.

of hazard envelopes attempts to find as many (u, B_m) points as possible that are close to the boundaries between safe and unsafe regions in the space. Notice that we intentionally left the precise definition of safe and unsafe regions unclear since it is one of the challenges we address next.

2.3.2 Challenges

The problem of hazard boundary identification for an MLC in the MLAS under analysis, entails multiple major challenges.

As discussed in Section 2.3.1, both a scenario u and an MLC behavior B_m collaboratively determine the violation or satisfaction of a safety requirement r. As a result, there are too many possible scenarios and MLC behaviors for the input space to be exhaustively explored without resorting to limiting assumptions that can bias the results [97]. One might argue that unsafe regions of the input space could be analytically identified using methods based on expert knowledge, such as FTA [54] and HAZOP [130], to provide clear insights into how hazards can occur. However, such methods are not sufficient to address all possible ways hazards can arise due to complex interactions between MLAS components and the opacity of ML components.

Second, the satisfaction or violation of r can only be determined if the system is operated within its surrounding environment. During testing, in addition to the first challenge above, this requires running a high-fidelity simulator which is generally very resource-intensive. The high cost of simulation highlights the need for an efficient and effective method to search as much of the input space as possible while focusing on the regions close to the boundary.

Lastly, recall it is unrealistic to consider a region 100% safe or unsafe. This is explained by two main reasons. First, simulators do not often enable full control of all relevant parameters in the environment, thus randomly configuring some of them. For example, the movement of pedestrians is random in CARLA [21], a high-fidelity simulator. Second, two inputs that are close in the input space may generate different MLC outputs that are handled differently by the rest of the system, e.g., due to the non-linear behavior of other DNNs using the MLC outputs as their input, resulting in different safety results (i.e., safe or unsafe). As a result, we cannot assume a uniform and consistent safety outcome for a region, making it difficult to define hard boundaries between safe and unsafe regions. Rather, hazard envelope boundaries (i.e., the dashed line in Figure 2.2) should be probabilistic as they encompass regions with a given probability threshold of violating a selected requirement.

To address the above challenges, we propose a novel method using Cooperative Co-Evolutionary Algorithm (CCEA) that efficiently address our objectives as an optimization problem, within a large input space, by decomposing such problem into lower-dimensional subproblems. Further, to recast our problem into a coevolutionary search problem, we define a special fitness function that can assess how far a candidate solution (i.e., a combination of u and B_m) is from the boundary of a "probabilistic" unsafe region. See Section 2.5 for details of our method.

2.4 Related Work

This section discusses existing studies related to the problem of hazard envelope boundary identification. Depending on the methods used, we found three categories: search-based methods, sampling-based methods, and formal methods.

2.4.1 Search-based Methods

Search-based methods employ metaheuristics (search algorithms) and convert the boundary identification problem into a search problem guided by a fitness function that evaluates how close a system input (e.g., test scenario) is from the boundary. Fitness assessment for individual system inputs often involves simulation executions to check whether safety requirements are violated.

Although there are many search-based methods for testing MLCs [115, 151], the problem of boundary identification has received very little attention. Only recently, Riccio and Tonella [116] proposed DeepJanus, the first search-based method to identify the *frontier of behavior* (frontier) of MLCs, i.e., a set of similar input pairs that trigger different behaviors (e.g., safe and unsafe) of the system. The discovered frontier can allow developers to approximate a safe operating envelope for the MLC (by interpolating the pairs). Also, the overlap of the estimated safe operating envelope with the *validity domain* of the MLC, which is the domain where the MLC is expected to behave according to its requirements [116], can facilitate the evaluation of the MLC's quality. Therefore, for example, DeepJanus can be useful in distinguishing between the performance of two MLCs that perform the same task. However, it cannot solve the issue of identifying the hazard boundary, as the impact of MLCs on safety can only be assessed when evaluating the entire system in a given environmental context. Furthermore, as illustrated in Figure 2.3, the interpolated frontier of behavior and the hazard boundary of an MLC are not necessarily the same. More precisely, a member of the frontier (i.e., a pair of safe and unsafe inputs) does not necessarily lie in proximity to the hazard boundary since the violations can occur in probabilistic safe regions (i.e., regions where the proportion of safe inputs is above a certain threshold) as argued in Section 2.3.2. Therefore, we need a novel method to identify the hazard boundary of an MLC within a system, considering the probabilistic nature of (un)safe inputs.

2.4.2 Sampling-based Methods

Unlike search-based methods, which are guided by fitness functions, sampling-based methods use repeated random samplings (e.g., Monte Carlo methods) or statistical metrics to identify certain system inputs that lead to safety violations. For example, Meltz and Guterman [89] proposed SmARTest, which uses Monte Carlo methods to identify a *scenario domain* (i.e., set of system inputs) that lead to safety requirement violations determined by measuring *Performance Assessment Functions* (PAFs) defined based on the requirements' Key Performance Indicators (KPIs). Sinha et al. [126] proposed Neural Bridge Sampling

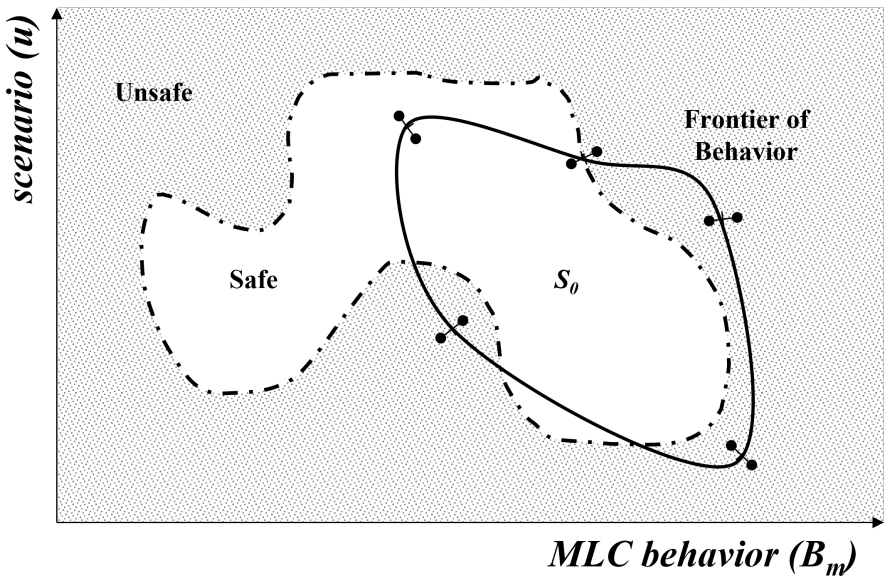


Figure 2.3: A possible application of *DeepJanus* to the systemic hazard boundary detection problem. Connected dots are a safe and unsafe pair.

(NBS), a method to measure the probability of rare events, such as accidents, using Monte Carlo methods. NBS decomposes the probability of a rare event into chained conditional probabilities, which are tractable to compute using standard Monte Carlo methods. This provides a better estimate than the naive Monte Carlo or Adaptive Multi-Splitting (AMS) methods.

Sampling-based methods can efficiently identify safe and unsafe inputs from the system's input space. However, they only consider system inputs (i.e., scenarios) and not the effect of different MLC behaviors for the same scenario. As discussed in Section 2.3.2, both the scenario and the MLC behavior must be taken into account to determine the conditions when MLC behavior leads to safety violations.

2.4.3 Formal Methods

Formal methods rely on formal representations of the input space, the system (including the MLC), and the output space. Examples of such representations include hybrid system or dynamical system formalisms [5]. Tools like SMT solvers [66] and Mixed Integer Linear Programming (MILP) [78] can be used to analyze whether the system containing the MLC can reach an unsafe region given its input space [50]. This is known as *reachability analysis*. Reachability analysis is used to identify the MLC's *barrier certificate*, which is an invariant function that constrains the state space of the system and ensures the satisfaction of a safety property [107] while considering the closed-loop behavior of the system. Barrier certificates can be seen as an over-approximation of the hazard boundary of the MLC.

Ivanov et al. [61] proposed Verisig, which can be applied to Cyber-Physical Systems (CPSs) with DNN-based feed-forward controllers with ReLU activation functions. Verisig

transforms the ReLU DNN into a hybrid system representation and combines it with the rest of the system. This recasts the problem as a hybrid system verification problem. Given a set of system inputs, the outputs can be approximated using Flow* [24], a nonlinear system reachability analyzer. Tuncali et al. [136] proposed another method to identify barrier certificates of DNN-based, feed-forward controllers, which is not limited to architectures with ReLU activation functions. This method first identifies candidate barrier certificates using simulations, then evaluates their suitability using the dReal [37], an SMT solver for nonlinear formulas in real numbers. Tran et al. [134] proposed NNV, a method to perform closed-loop reachability analysis of control systems with Deep Reinforcement Learning (DRL) controllers. These controllers have a feed-forward architecture with ReLU/Saturation activation functions. NNV calculates a low-error over-approximation of the output region, which are reached by the system given its inputs.

Although the aforementioned methods provide guarantees for the hazard boundary and cover all possible trajectories of the system, they suffer from practicality and scalability issues. For example, over-approximation of the hazard boundary might incorrectly reduce (or even remove in the worst case) the safe operating envelope of the system by incorrectly considering some safe behaviors unsafe, thus limiting the practicality of the methods [135]. Furthermore, reachability analysis can only be applied to feed-forward controllers with specific activation functions. Thus, it cannot be used for practical MLCs that perform perception, obstacle tracking, or prediction tasks with different DNN architectures (e.g., recurrent neural networks). Also, reachability analysis has not yet been applied to closed-loop, industrial Cyber-Physical Systems (CPS) with feedback DNN controllers [61,134,136]. In such a context, scalability is very likely to become an acute problem.

2.4.4 Remark on Differences in Objectives

A common goal underlying all the above-mentioned methods is to identify the hazard boundary of a *given* MLC embedded within its containing system (MLAS). It could be useful when the MLC under test is fixed, but as soon as the MLC changes (e.g., via retraining), the previously identified hazard boundary would be invalid, and the whole safety verification exercise would have to be repeated. On the other hand, in our research, we aim to identify the combinations of conditions and MLC behaviors, *without* referring to a specific MLC implementation, that could potentially lead to hazards. Once characterized, such situations could then, independently of a specific MLC implementation, be used to monitor the operation of the system and MLC and warn the user in case it is operating near to the hazard boundary.

In the following section, we propose a novel method that addresses the challenges discussed in Section 2.3.2, and is applicable to various types of MLCs, such as perception, planning, and control, without making any assumptions about their architecture.

2.5 Our Approach

In this section, we provide a solution to the problem described in Section 2.3, i.e., the hazard boundary identification of an MLC in the MLAS under analysis. Our key idea is to recast the problem as a cooperative co-evolutionary search problem where scenarios and MLC behaviors co-evolve as two separate populations but contribute together to find complete solutions (i.e., the combinations of scenarios and MLC behaviors) close to the boundary. Then, we use CCEAs, the algorithms that are well known to be effective at solving search problems such as the one described in Section 2.2.

In the following subsections, we first describe how scenarios and MLC behaviors can be represented as two separate populations in a search problem (Section 2.5.1). We then define a novel fitness function of the search problem to assess how close a complete solution is from the boundary (Section 2.5.2). Finally, we present our novel method based on CCEAs using the representation and the fitness function (Section 2.5.3).

2.5.1 Representations

We consider two populations, one for scenarios and another one for MLC behaviors. This is to consider *all possible* MLC behaviors that can lead to the boundary regions when combined with certain scenarios. Note that MLCSHE does not aim to test a particular MLC in the system under analysis, but rather to monitor the behavior of current and future implementations of the MLC using the resulting boundary information. Therefore, it is important to manipulate MLC behaviors and scenarios to find all boundary regions. However, the individuals of the MLC population, subjected to evolutionary operators, are only represented as *MLC-outputs* ($\boldsymbol{out}_m$). This is due to the initial *MLC-input* ($\boldsymbol{in}_m$) being (indirectly) determined by the scenario, whereas the next MLC-inputs are affected by previous MLC-outputs. Therefore, $\boldsymbol{in}_m$ is recorded in an *archive* of complete solutions (i.e., A_c in Algorithm 1; see Section 2.5.3 for details) but not included in the representation of the behavior of the MLC that can be manipulated by the search. Recording the $\boldsymbol{in}_m$ and $\boldsymbol{out}_m$ sequences, along with their corresponding scenarios ($\boldsymbol{u}$), is indeed crucial as it records unsafe behaviors of an MLC (its input and output sequences) given a set of environmental conditions (its *scenarios*). This information enables the design of safety monitors that will prevent the MLC from contributing to a systemic hazard via leveraging the recorded information.

MLC behaviors

One of the two populations considered for the search is the set of MLC behaviors. The behavior of an MLC can be expressed as a sequence of input and output tuples. However, as discussed above, the inputs of an MLC are indirectly controlled by the environmental input to the system (i.e., scenario parameters) and the components of the system that process that input before it is passed on to the MLC. Thus, the parameters that we can directly manipulate during the search are the outputs of the MLC. We represent an individual in

the population of the MLC behaviors as a sequence of MLC outputs where the t-th element of the sequence denotes an MLC output at time step t.

The output of an MLC depends on the task performed by the MLC. For instance, in the case of a steering angle estimator, the output is a single real value. Whereas, in the case of an object classifier, the output is a vector of probabilities (real values between 0 and 1), where each element corresponds to a label. Finally, similar to our running example, in the case of obstacle detection, the outputs in an ML component (MLC) are detected obstacles, i.e., their bounding box[2], their label (such as pedestrian, vehicle, lamp post, etc.), and their timestamp. Therefore, an *mlco* (MLC Output) for a simulation duration T can be defined as a sequence of the trajectories of detected obstacles during T in the case of obstacle detection.

Specifically, given the maximum number of detectable objects n and the simulation duration T, an *mlco* can be defined as a sequence of n trajectories $\langle trj_1, \ldots, trj_n \rangle$ where trj_i represents the trajectory of the i-th object (in terms of the bounding boxes) for T. By allowing the search algorithm to manipulate individual trajectories, an arbitrary *mlco* can be generated for obstacle detection.

However, allowing the search algorithm to generate all the bounding boxes for individual time steps will likely yield an unrealistic trajectory randomly moving around without a consistent direction, which we observed during our initial trials. Therefore, it is better to allow the search algorithm to generate only the *start* and *end* bounding boxes, and then generate the remaining bounding boxes for intermediate time steps using linear interpolation between the start and end boxes. Specifically, trj_i can be defined as a triple $(class_i, start_i, end_i)$ where $class_i$ is the class of the i-th object (e.g., car, bicycle, pedestrian), $start_i$ is the position and the size of the bounding box of the i-th object at time step $t = t_{start}$, and end_i is the position and the size of the bounding box of the i-th object at time step $t = t_{end}$. For example, *start* or *end* can be defined as a quintuple $(t, x_{min}, x_{max}, y_{min}, y_{max})$, which are time and bounding box parameters for the beginning or the end of the trajectory, respectively. Then, for a given trajectory $trj = (class, start, end)$, we can easily generate the positions and sizes of bounding boxes for intermediate time steps (i.e., $1 < t < T$) based on *start* and *end* (using linear interpolation) whenever needed for a simulation.

We want to note that MLCSHE does not aim to test a given MLC implementation in the system under analysis, but to monitor the behavior of any current or future implementation of the MLC using the resulting boundary information (see Sections 2.1 and 2.3.1). For a given implementation, there may indeed be "unfeasible-in-practice" MLC behavior for a given scenario, which is, however, hard to determine beforehand. But then this is part of the problem space that will never be reached at run-time and is not an issue.

[2]A bounding box specifies the area on the image processed by the obstacle detector that contains the detected obstacle. It can be expressed as $(x_{min}, x_{max}, y_{min}, y_{max})$ corresponding to a specific 2D box.

Scenarios

A scenario can be represented as a heterogeneous vector of real and integer values. For the case of an AV, a scenario consists of the vehicle itself, the weather, the road and other static (e.g., lamp posts and other obstacles) and dynamic objects (e.g., pedestrians and other cars) [44]. Each of them have many attributes of various types, namely float (e.g., speed) and enumerated types (e.g., line pattern) which can be encoded as integer values.

The size of a scenario individual is determined by the simulator. Furthermore, a finer-grained level of simulation control implies a larger scenario size as more parameters have to be manipulated by the search algorithm. For instance, one can manipulate all weather-related parameters separately (10 parameters in the case of CARLA [21]) or manipulate them using the weather preset parameter (1 parameter) which sets the value of all granular weather parameters according to high-level modalities, e.g., rainy sunset, clear noon.

Figure 2.4 is the scenario domain model for our running example. A Scenario consists of one or more Vehicles (including the ego vehicle), zero or more Pedestrians and, Mission and Weather. The attributes of the domain model that act as the parameters for a scenario representation are written in bold font in Figure 2.4. Therefore, a scenario can be defined by the time_of_day, weather preset, map of the town, start_point of the ego vehicle, its target_destination and target_velocity, the number of Pedestrians, and the number and position of other Vehicles with respect to the ego vehicle (e.g., in front, on the opposite lane).

Operational Design Domain (ODD). The Operational Design Domain or ODD defines an operational envelope of the AV, i.e., a set of bounds on the environmental parameters of the system. For instance, highway driving is an ODD for AVs which determines the type of the road, the average speed of the surrounding vehicles, and the (lack of) pedestrians in the vicinity [28]. However, within an ODD, many scenarios can still be defined, e.g., weather, the number of cars, the length and shape of the road. Therefore, a search can be done within an ODD, which sets the values or the bounds of some parameters, such as the target speed of the ego vehicle. The parameter bounds or values set by the ODD will remain static for the duration of the search, e.g., a target speed of 90 km/hr in a highway driving ODD, or the angle of the sunlight during a daytime driving ODD.

2.5.2 Fitness Function

This section presents our proposed fitness function in detail. Our aim is to design a fitness function that can effectively guide the search towards the boundary of unsafe regions. However, as mentioned in Section 2.3, we cannot assume that a region is either 100% unsafe or safe. To address this, we first define the notion of safe and unsafe inputs, followed by *probabilistic* unsafe regions.

Definition 1 (Safe and Unsafe Inputs)**.** An input is *unsafe* if and only if it leads to the violation of a given requirement. Otherwise, the input is *safe*.

Recall that an input is a combination of a scenario and an MLC behavior in our context.

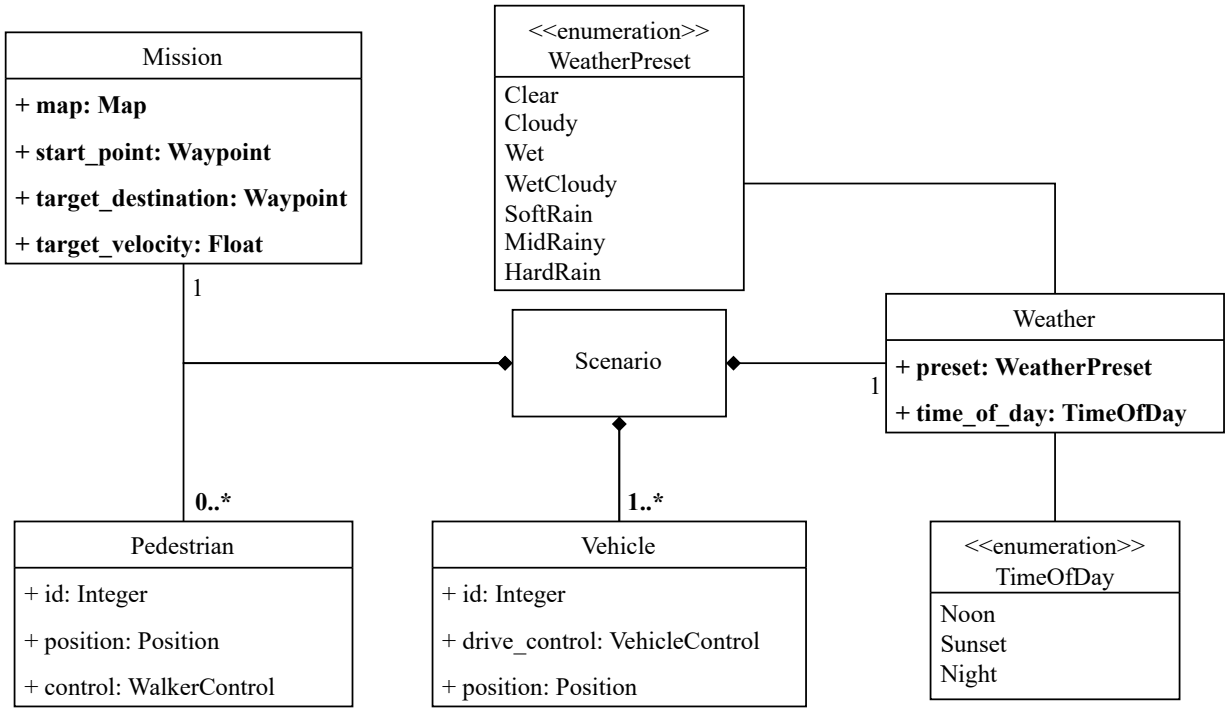


Figure 2.4: The scenario domain model for the running example. The model is based on the concepts provided in the CARLA World domain model [21, 33]. The scenario parameters are shown on the figure in bold font, i.e., the weather preset, the attributes of Mission and the number of actors such as Vehicles and Pedestrians.

Definition 2 (Probabilistic Unsafe Region)**.** Let X be a set of all possible inputs, representing the input space. Given a threshold probability p_{th}, a region $G \subseteq X$ is p_{th}-*unsafe* when the proportion of unsafe inputs in G is higher than p_{th}.

For example, if we randomly draw an input from a 5%-unsafe region, we have more than 5% chance of leading to a safety violation. The value of p_{th} should be determined by a domain expert within a specific application context.

Notice that the shape of a probabilistic unsafe region is unknown, as is its boundary. Nevertheless, we can *approximate* how far an arbitrary input is from the boundary by sampling its neighborhood. Specifically, for an input $x \in X$, let p_x be the proportion of unsafe inputs in the neighborhood of x. If $p_x \leq p_{th}$, it implies that x is not likely to be located in a p_{th}-unsafe region. Otherwise, if $p_x > p_{th}$, it implies that x is likely in a p_{th}-unsafe region. Therefore, if p_x is close to p_{th}, it implies that x is close to the boundary of a p_{th}-unsafe region. To leverage this idea, we define the notion of neighborhood as follows:

Definition 3 (Neighborhood)**.** For an input $x_i \in X$ and a non-negative real number $\delta \in \mathbb{R}^+$, a *neighborhood* of x_i with the radius of δ, denoted by $N(x_i, \delta)$, is defined as follows:

$$N(x_i, \delta) = \{x_j \in X |\ dist(x_i, x_j) \leq \delta\} \tag{2.1}$$

where $dist(x_i, x_j)$ indicates the distance between x_i and x_j.

Notice that various distance functions *dist* can be adopted depending on the nature of complete solutions. For example, if a complete solution can be represented as a heterogeneous vector composed of numerical, ordinal, and categorical values, *heterogeneous distance metrics* [144] are good candidates to measure the distance between two complete solutions. In Figure 2.2, a neighborhood with a radius of δ is visualised as a circle between safe and unsafe regions.

Based on Definition 3, let $p_{x,\delta}$ be the proportion of unsafe inputs in $N(x,\delta)$. Then, as discussed above, we can use the difference between $p_{x,\delta}$ and p_{th} to approximate the distance between x and the boundary of a p_{th}-unsafe region. However, we cannot compute the exact value of $p_{x,\delta}$ since $N(x,\delta)$ has too many complete solutions to exhaustively evaluate. Nevertheless, we can compute an *estimate* of $p_{x_i,\delta}$, denoted by $\hat{p}_{x_i,\delta}$, and its confidence interval since the consecutive trials of checking whether an input $x_j \in N(x_i,\delta)$ is safe or not are assumed to be independent and can be treated as Bernoulli Experiments.

Specifically, the probability distribution of $p_{x,\delta}$ can be modelled as a Binomial distribution, and we can compute $\hat{p}_{x,\delta}$ as follows:

$$\hat{p}_{x,\delta} = \frac{unsafe(N(x,\delta))}{evaluated(N(x,\delta))} \tag{2.2}$$

where $evaluated(N(x,\delta))$ is the number of inputs evaluated (sampled) in $N(x,\delta)$ and $unsafe(N(x,\delta))$ is the number of unsafe inputs among those evaluated. Furthermore, using the Wilson Confidence Intervals [19], we can compute the confidence interval of $p_{x,\delta}$, denoted by $CI(p_{x,\delta})$, as follows:

$$\begin{aligned} CI(p_{x,\delta}) &= \frac{1}{1+\gamma}\left(\hat{p}_{x,\delta} + \frac{\gamma}{2}\right) \\ &\pm \frac{z}{1+\gamma}\sqrt{\frac{\hat{p}_{x,\delta}(1-\hat{p}_{x,\delta})}{evaluated(N(x,\delta))} + \frac{\gamma}{4 \times evaluated(N(x,\delta))}} \end{aligned} \tag{2.3}$$

where $\gamma = \frac{z^2}{evaluated(N(x,\delta))}$ and z is determined by the standard normal distribution for a given confidence level (e.g., for a 95% confidence level, $z = 1.96$).

Based on $CI(p_{x,\delta})$, we can assess the maximum difference[3] between $p_{x,\delta}$ and p_{th} as follows:

$$diff(p_{x,\delta}, p_{th}) = \max\left(|UL(p_{x,\delta}) - p_{th}|, |LL(p_{x,\delta}) - p_{th}|\right) \tag{2.4}$$

where $UL(p_{x,\delta})$ and $LL(p_{x,\delta})$ are the upper and lower limits of $CI(p_{x,\delta})$, respectively. Using $diff(p_{x,\delta}, p_{th})$, we define our fitness function as follows.

Definition 4 (Boundary-Seeking Fitness Function)**.** For an input x, a neighborhood radius δ, and a threshold probability p_{th}, the *fitness value* of x given δ and p_{th}, denoted by $fitness(x,\delta,p_{th})$, is defined as follows:

$$fitness(x,\delta,p_{th}) = \frac{diff(p_{x,\delta}, p_{th})}{\max(p_{th}, (1-p_{th}))} \tag{2.5}$$

[3]We consider the maximum difference to be conservative.

where the denominator is a normalisation factor, making the range of the fitness value between 0 and 1.

In other words, we compute the fitness value of an input x using the difference between $p_{x,\delta}$ (i.e., the proportion of unsafe inputs in the neighborhood of x with the radius of δ) and p_{th} (i.e., the probability threshold).

Note that the fitness function is meant to be minimized and decreases as the difference between p_{th} and $p_{x,\delta}$ decreases. The fitness function also takes the number of observations (i.e., evaluated inputs) within $N(x, \delta)$ into account, as the size of $CI(p_{x,\delta})$ (i.e., the confidence interval of $p_{x,\delta}$) decreases when the number of observations in the neighborhood increases, thereby also decreasing the value of the fitness function. A sparsely populated neighborhood therefore tends to yield high fitness values, which is what we would expect as $p_{x,\delta}$ in such neighborhoods comes with much uncertainty.

To better illustrate how the boundary-seeking fitness function distinguishes between inputs based on their proximity to the boundary, let us consider an input space X and two inputs $x_1 \in X$ and $x_2 \in X$ where $CI(p_{x_1,\delta}) = 0.1 \pm 0.05$ and $CI(p_{x_2,\delta}) = 0.5 \pm 0.1$ for a small δ. This means that the proportions of unsafe inputs around x_1 and x_2 are estimated as 0.1 ± 0.05 and 0.5 ± 0.1, respectively. If we consider the boundary of a 5%-unsafe region (i.e., $p_{th} = 0.05$), we can say that x_1 is closer to the boundary than x_2 since the proportion of unsafe inputs around x_1 is up to 15% while that around x_2 is up to 60%. This is exactly captured by the fitness function since $\mathit{diff}(p_{x_1,\delta}, 0.05) = 0.1$ and $\mathit{diff}(p_{x_2,\delta}, 0.05) = 0.55$, thus yielding $\mathit{fitness}(x_1, \delta, 0.05) = \frac{0.1}{0.95} = 0.105$ and $\mathit{fitness}(x_2, \delta, 0.05) = \frac{0.55}{0.95} = 0.579$, showing that x_1 is closer to the boundary than x_2.

2.5.3 MLC Systemic Hazard Envelope (MLCSHE) Algorithm

Based on the representations of scenarios and MLC behaviors described in Section 2.5.1 and the boundary-seeking fitness function described in Section 2.5.2, this section proposes a novel algorithm, *MLC Systemic Hazard Envelope (MLCSHE)* [/mɪlʃ/], based on CCEA as described at the beginning of Section 2.5.

Algorithm 1 shows the pseudocode of MLCSHE. It takes as input a population size n, a minimum number of joint fitness assessments per individual k, a threshold probability p_{th} to define a probabilistic unsafe region, a threshold distance d_a to ensure the diversity of individuals and complete solutions in archives, a maximum population archive size l, a distance threshold d_{th} to filter the complete solutions that are distinct enough, and a boundary fitness threshold t_b to filter complete solutions close enough to the boundary; it returns an archive A_b of distinct complete solutions, with the pairwise distance of more than d_{th}, whose fitness values are less than t_b (i.e., close to the boundary of a p_{th}-unsafe region), while k and l are parameters to control the algorithm's search behavior (detailed below). MLCSHE in essence is a CCEA that uses population archives as described in Section 2.2. However, it is different from other similar methods as its goal is to return a set of complete solutions satisfying certain properties (i.e., close to the boundary) rather than returning a single-best complete solution.

Algorithm 1: MLC Hazard Envelope Search algorithm (MLCSHE)

Input : Population Size n
Minimum Number of Fitness Assessments per Individual k
Threshold Probability p_{th}
Distance Threshold for Population Archives d_a
Maximum Size of Population Archive l
Distance Threshold for Post-processing d_{th}
Boundary Fitness Threshold for Post-processing t_b

Output: Archive of Distinct Boundary Complete Solutions A_b

1 Population of MLC Output Sequences $P_O \leftarrow initPopulation_O(n)$
2 Population of Scenarios $P_S \leftarrow initPopulation_S(n)$
3 Archive of MLC Output Sequences $A_O \leftarrow P_O$
4 Archive of Scenarios $A_S \leftarrow P_S$
5 Archive of Complete Solutions $A_c \leftarrow \emptyset$
6 **while** *not*(*stopping_condition*) **do**
7 $\quad P_O, P_S, A_c \leftarrow \textbf{\textit{assessFitness}}(P_O, P_S, A_O, A_S, k, A_c, \delta, p_{th})$
8 $\quad A_O \leftarrow \textbf{\textit{updatePopulationArchive}}(P_O, l, d_a)$
9 $\quad A_S \leftarrow \textbf{\textit{updatePopulationArchive}}(P_S, l, d_a)$
10 $\quad P_O \leftarrow Breed(P_O) \cup A_O$
11 $\quad P_S \leftarrow Breed(P_S) \cup A_S$
12 Archive of Complete Solutions $A_b \leftarrow postProcess(A_c, d_{th}, t_b)$
13 **return** A_b

The algorithm first randomly initializes the population of MLC Output sequences P_O (line 1), the population of scenarios P_S (line 2), and their population archives, A_O (line 3) and A_S (line 4), respectively. The algorithm also initializes the archive of complete solutions A_c as an empty set (line 5). The algorithm then co-evolves P_O and P_S using A_O and A_S, until the *stopping_condition* is met (line 6), such that it guides them towards the complete solutions that are close to the boundary of a p_{th}-unsafe region (lines 6–11). During the co-evolution, the algorithm repeats the following three steps:

1. assess the fitness values of individuals in both P_O and P_S and update A_c to include complete solutions with their joint fitness values evaluated by the simulator (using function ***assessFitness*** at line 7, described in detail in Algorithm 2);

2. update A_O and A_S based on the individual fitness values, d, and l (using function ***updatePopulationArchive*** at lines 8–9, described in detail in Algorithm 3); and

3. evolve P_O and P_S (using the function *breed* detailed at the end of Section 2.5.3), and merging them with A_O and A_S, respectively, to make up the next generation of P_O and P_S (lines 10–11).

After the co-evolution, the algorithm creates a set of complete solutions A_b from A_c such that the distance between two arbitrary, complete solutions in A_b is at least d_{th} and the fitness value of every complete solution in A_b is less than t_b (using function *postProcess* at line 12). The algorithm ends by returning A_b (line 13).

Fitness Assessment

The function ***assessFitness*** is to first calculate the joint fitness values of complete solutions, generated by joining the individuals in P_O and P_S (with higher priorities to the individuals in A_O and A_S, respectively) such that each individual is joined at least k times, using the simulator. In other words, the fitness of each individual is assessed based on at least k collaborators to avoid inaccurately estimating the individual fitness (see Section 2.2 for more details about collaborators). To reduce the number of computationally intensive simulations, complete solutions that are the same as the ones in A_C (i.e., generated in the previous generations) are not simulated again. Then, the function assesses the fitness value of each individual using the joint fitness values of the complete solutions that contain the individual.

Specifically, Algorithm 2 shows the pseudocode of ***assessFitness***. It takes as input the population of MLC output sequences P_O, the population of scenarios P_S, the population archive of MLC output sequences A_O, the population archive of scenarios A_S, the minimum number of fitness assessments per individual k, the archive of previously evaluated complete solutions A_C, the neighborhood radius δ, and the threshold probability p_{th}; it then returns P_O and P_S updated to include individual fitness values, and A_C updated to include newly generated complete solutions and their joint fitness values.

The algorithm begins by initializing a set of populations PS as $\{P_S, P_O\}$ (line 1). It also initializes a set of complete solutions C by selecting and collaborating individuals from P_S and P_O using the *collaborate* function (line 2). This function first makes every individual of P_S and P_O collaborate with every individual of A_O and A_S, respectively, and if the number of collaborations for each individual is less than k (i.e., when the size of A_O and A_S are less than k, where $k \geq 1$), randomly selected individuals of $P_O \setminus A_O$ and $P_S \setminus A_S$ are used in addition to A_O and A_S, respectively, to ensure a minimum of k collaborations for each individual [4]. Then, for each complete solution $c \in C$ (line 3), if $c \notin A_C$, i.e., c has not been previously evaluated (line 4), the algorithm evaluates c using the high-fidelity simulator to identify if c is unsafe (line 5) and adds c with its evaluated result into A_c (line 6). Once A_c is updated using C, for each complete solution $c \in A_c$ (line 7), the algorithm computes its joint fitness value (i.e., the boundary-seeking fitness value) using A_c, δ, and p_{th} by calculating the proportion of unsafe complete solutions in the neighborhood of c and its difference from the threshold probability as described in Section 2.5.2 (line 8). Although computing the neighborhood of c requires many distance computations, we can significantly reduce the computations by reusing the distances among the complete solutions that were originally in the input A_c. For each individual $P \in PS$ and for each $i \in P$ (lines 9–10), the algorithm sets the minimum (i.e., the best since we aim to minimize fitness values) joint fitness value of the complete solutions involving i as the individual fitness of i (line 11). An elitist individual fitness assessment strategy (i.e., selecting the best fitness value), is consistent with reported experimental studies [79, 83] as well as our preliminary evaluation results on a widely used benchmark problem known

[4]Note that a high value of k might add significant computational cost to the search since it tends to exponentially increase the number of joint fitness evaluations per individual.

Algorithm 2: ***assessFitness***

Input : Population of MLC Output Sequences P_O
Population of Scenarios P_S
Archive of MLC Output Sequences A_O
Archive of Scenarios A_S
Minimum Number of Fitness Assessments per Individual k
Archive of Complete Solutions A_c
Complete Solutions Pairwise Distance Matrix D_c
Neighborhood Radius δ
Threshold Probability p_{th}

Output: Updated Population of MLC Output Sequences P_O
Updated Population of Scenarios P_S
Updated Archive of Complete Solutions A_c

```
1  Set of Populations PS ← {P_O, P_S}
2  Set of Complete Solutions C ← collaborate(P_O, P_S, A_O, A_S, k)
3  foreach Complete Solution c ∈ C do
4      if c ∉ A_c then
5          c.isUnsafe ← simulate(c)
6          A_c ← A_c ∪ {c}
7  foreach Complete Solution c ∈ A_c do
8      c.fitness ← computeBoundaryFitness(c, A_c, δ, p_th, D_c)
9  foreach Population P ∈ PS do
10     foreach Individual i ∈ P do
11         i.fitness ← assessIndividualFitness(i, A_c)
12 return P_O, P_S, A_c
```

as the *MTQ* (*Maximum of Two Quadratics*) [20]. The algorithm ends by returning the updated P_O, P_S, and A_c (line 12).

Update Population Archive

The function ***updatePopulationArchive*** updates the population archives A_O and A_S, for the next generation. They play a key role in guiding the search algorithm since every other individual has to form a complete solution with them, whose joint fitness will be assessed afterwards.

There are many ways to update the population archive such as the ones proposed in *iCCEA* and *pCCEA* [83]. However, as mentioned in Section 2.2, they can be inefficient due to additional fitness evaluations for updating population archives, making them impractical for our problem involving computationally expensive simulations for fitness evaluation. Instead, we can consider more efficient archive update strategies as follows: selecting individuals with the best fitness values (*Best*), selecting the best individual plus random individuals (*Best+Random*), or randomly selecting individuals (*Random*) [83]. Our preliminary evaluation results on the *MTQ* problem showed that both *Best* and *Best+Random* work similarly well for updating population archives in MLCSHE. To ensure the diversity of individuals in each population archive and maximize exploration, we choose *Best+Random* with a similarity threshold (i.e., the distance threshold d_{th}) that filters out individuals deemed too similar to be included in a population archive. The pseudocode for updating a population archive is provided in Algorithm 3.

The algorithm takes as input a target population P, a maximum size of a population archive l, and a threshold distance (i.e., the minimum distance between two arbitrary individuals in a population archive) d_{th}; it returns a population archive A_P of P such that $|A_P| \leq l$ and $d(i, j) \geq d_{th}$, based on the distance function d as described in Section 2.5.2, for all $i, j \in A_P$ if $i \neq j$.

Algorithm 3: ***updatePopulationArchive***

Input : Population P
Maximum Size of Population Archive l
Threshold Distance d_{th}
Output: Population Archive A_P

```
1 Archive A_P ← {popBestFitnessIndividual(P)}
2 while |A_P| < l and |P| > 0 do
3     Individual i ← randomPop(P)
4     if isDistinct(i, A_P, d_th) then
5         A_P ← A_P ∪ {i}
6 return A_P
```

The algorithm starts by initializing a population archive A_P for P using the individual with the best fitness value among all the individuals in P (using function *popBestFitnessIndividual* at line 1). While $|A_P| < l$ or $|P| > 0$, the algorithm iteratively pops a random

individual i from P (line 3) and add i into A_P (line 5) if i is distinct from all individuals in A_P based on the distance threshold of d_{th} (line 4). The algorithm ends by returning A_P (line 6).

Evolution

As illustrated in Algorithm 1, after updating A_O and A_S, P_O and P_S undergo evolution to generate their next generation using a *breed* operation, which entails three main steps:

1. *Selection.* MLCSHE selects the candidate individuals for breeding via the standard *tournament selection* technique, i.e., the most widely used selection technique for evolutionary algorithms [142]. It is simple yet effective since it only requires the rank ordering of individuals in terms of their fitness values.

2. *Crossover.* Selected individuals of each population act as parents to create offspring individuals using a crossover operation [142]. For our problem, the widely used *uniform crossover* technique [79, 142] is used, since there is no preference for specific points in individuals as crossover points.

3. *Mutation.* Finally, offspring individuals are mutated via the introduction of stochastic noise [142]. Since the individuals are heterogeneous vectors with both float and integer values, the standard *Gaussian* and *integer randomization* mutation techniques are applied to individual elements, respectively [79]. Through these mutations, all valid individuals can be considered during the search, making it possible to find the global optimum. There is a chance a mutated individual might be invalid. For example, the x-coordinates (x_{min} and x_{max}) of a bounding box used to define a detected obstacle can be out of the camera frame width bounds (from 0 to 800 pixels). Such invalid cases are handled by a simple *repair* function in our implementation, which is available in the replication package (see Section 2.6.5).

We want to note that there are hyperparameter values for selection, crossover, and mutation (e.g., the tournament size, crossover and mutation rates) that could affect breeding performance. More details on tuning hyperparameter values in our evaluation is provided in Section 2.6.2.

2.6 Evaluation

In this section, we report on the empirical evaluation of MLCSHE when applied to an open-source MLAS. Specifically, we provide answers for the following research questions:

RQ$_1$ (Effectiveness) How *effectively* can MLCSHE find the MLAS hazard envelop boundary compared to baseline boundary search approaches?

RQ$_2$ (Efficiency) How *efficiently* can MLCSHE find the MLAS hazard envelop boundary compared to baseline boundary search approaches?

To answer RQ_1, we investigate *how many* complete solutions (i.e., combinations of scenarios and MLC behaviors) that are close to the boundaries are found by different boundary search approaches, including MLCSHE, given a same time budget. To answer RQ_2, we investigate *how quickly* complete solutions that are close to the boundaries are found by different boundary search approaches. The results of RQ_1 and RQ_2 may depend on the distance threshold between complete solutions and the boundaries[5] (d_{th} and t_b in algorithm 1). Thus, we also consider the effects of the distance thresholds while answering RQ_1 and RQ_2.

2.6.1 Evaluation Subjects

We use Pylot [39], one of the highest ranking component-based AV on the CARLA Autonomous Driving Leaderboard [1], at the time of the evaluation. The leaderboard evaluates AV according to 11 metrics that are designed to assess safe driving performance such as collision, red light infractions and route completion. Pylot's high performance on the leaderboard makes it a good candidate to consider as an evaluation case study. Furthermore, Pylot is one of the only high-ranking AV that is open-source, has been deployed on a real-life vehicle [39].

We also use CARLA [33], a high-fidelity open-source AV simulator. CARLA allows us to control various static and dynamic elements in driving environments. Based on the controllable elements, following a previous study using CARLA [43], we consider the following seven scenario elements: road curve and length, start and end points on maps, the density of pedestrians, time of day, and weather condition. The detailed explanation for the scenario elements and their values (ranges) are available in the supporting material (see Section 2.6.5).

For the ML component under test in Pylot, we target a DNN-based obstacle detection module. The obstacle detection module takes digital images of the front-facing camera and detects obstacles in the images in terms of their location and size (captured as a bounding box), their type, and the uncertainty associated with the predicted label. The output of the obstacle detection module is then used by an obstacle prediction module that predicts the trajectories of the detected obstacles for future timestamps, followed by planning and control modules that generate driving commands considering the obstacles' predicted trajectories. Thus, we let the boundary search methods manipulate the parameters that define the output sequence of the target MLC (i.e., the object detection module) during the execution of a simulation. Specifically, for each obstacle, there are 11 parameters: the label of the detected obstacles (pedestrian or vehicle), the start and end time the obstacles are detected, and the 2D coordinates (i.e., x_{min}, x_{max}, y_{min} and y_{max}) that define the start and end bounding boxes of the trajectories on the input image. Additional details regarding the scenario and MLC output are available in the supporting material (see Section 2.6.5).

[5]Recall that the fitness of a complete solution is defined based on its distance from the hazard boundary. Throughout the rest of the section boundary distance threshold and fitness threshold are used interchangeably.

The above-mentioned categorical (e.g., road curve, weather condition, and obstacle's label) and numeric (e.g., obstacle's position) parameters defining scenarios and MLC outputs are used to define a distance function *dist* which measures the distance between two complete solutions as discussed in Section 2.5.2. Given that these parameters are heterogeneous, we use a *heterogeneous distance metric*. Specifically, *dist* is defined as the average of the normalized Hamming distance [139] of categorical values and the normalized City Block distance [139] of numeric values, where the latter are normalized by their maximum range of values that each parameter can take. For instance, the y-coordinates of the MLC outputs range from 0 to 600 due to the height of the camera frame, and thus they are divided by 600 to be normalized. As we have many pairwise distance calculations during the search, we opted for these distance metrics since they are computationally efficient compared to alternatives. Given its definition above, *dist* ranges between 0 and 1. If $dist = 1$ between two complete solutions, it means all the categorical values of the two are different, and the differences in all the numeric values of the two are the maximum.

Among various AV safety requirements used in the literature [44, 80, 114], considering the capability of CARLA and the major functionality of our target MLC (i.e., the object detection module), we focus on the following safety requirement: *"AV should have a distance no less than d_{min} from the vehicle in front."* To detect safety violations, if any, during the simulation of a complete solution (i.e., the combination of a scenario and an MLC behavior), we measure the distance between the ego vehicle and the vehicle in front for each simulation time step. If the distance is less than d_{min} at any time, the violation is detected and the complete solution is marked as unsafe. Since our driving scenarios often involve junctions with traffic lights where the vehicles should completely stop for a while, d_{min} should be small enough to avoid false alarms (i.e., incorrectly triggering safety violations) even when the vehicles completely stop. Based on the rule-of-thumb that the driver should be able to see the rear tire of the vehicle in front and a small part of the asphalt when stopping behind a stopped vehicle, we set d_{min} to 1.5 m.

Due to the execution time of individual simulations in CARLA, which is around five minutes on average, the total computing time for the evaluation is more than *1800 hours* (*75 days*). To address this issue, we conduct our evaluation on two machines, M1 and M2. Machine M1 is a desktop computer with 2.6 GHz Intel i7-10750H CPU, NVidia GeForce RTX 2070 with Max-Q Design GPU (with 8 GB memory), and 32 GB RAM, running Ubuntu 20.04. Machine M2 is a g4dn.2xlarge node configured as NVIDIA GPU-Optimized AMI (version 22.06.0) in Amazon Elastic Cloud (EC2) with eight virtual cores, NVIDIA T4 GPU (with 16GB memory), and 32 GB RAM, running Ubuntu 20.04. Specifically, we use M1 for Random Search (RS) and standard Genetic Algorithm (GA), while MLCSHE is run on M2. Note that since we keep the number of simulations, as opposed to the execution time, constant over all the experiments, the experiments on M1 and M2 are comparable (see Section 2.6.2 for details).

2.6.2 RQ_1: Effectiveness

Methodology

To answer RQ_1, we execute MLCSHE and other comparable methods to generate sets of complete solutions that are close to the boundary and measure their boundary search effectiveness in terms of *Distinct Boundary Solutions (DBS)* capturing the number of distinct complete solutions close to the boundary. Specifically, given a distinctiveness (distance) threshold d_{th} (for the distinctiveness of complete solutions) and a boundary closeness (fitness) threshold t_b (for the closeness to the boundary), let C_V be the set of complete solutions generated by a boundary-seeking method V, satisfying the following conditions[6]: (1) the pairwise distance between two arbitrary complete solutions in C_V is more than d_{th} and (2) the fitness value of every complete solution in C_V is less than t_b. Then, *DBS* of V is defined as $DBS(V) = |C_V|$ (i.e., the size of C_V). Recall that both distance and fitness values are normalized ($d_{th}, t_b \in [0, 1]$).

To better understand how *DBS* varies depending on different d_{th} and t_b thresholds, we vary d_{th} and t_b. Specifically, we set d_{th} to 0.1, 0.2, and 0.3 because it is unrealistic to think that two arbitrary complete solutions are distinctive only if their pairwise distances are more than 30% of the maximum possible distance[7]. We set t_b to 0.01, 0.03, 0.5, 0.1, 0.15, and 0.2 because we are not interested in complete solutions with fitness values above 0.2 (i.e., far from the boundary, such that the normalized difference between the probability threshold p_{th} and the proportion of unsafe inputs near the complete solutions is above 0.2).

For the other methods to compare with MLCSHE, as discussed in section 2.4, we could not find any other work that has been proposed to address the problem targeted by this chapter. Note that DeepJanus is incomparable to MLCSHE, as discussed in Section 2.4, because:

1. its goal is to study an MLC's safety under various conditions, which is different than the goal of this research effort, i.e., finding the conditions under which an MLC's behavior can impact the safety of the system;

2. the boundary identified by DeepJanus consists of safe-unsafe pairs that can exist in probabilistic safe or unsafe regions.

Thus, we compare MLCSHE against two baseline methods, namely *Random Search (RS)* and *standard Genetic Algorithm (GA)* [79]. RS randomly generates complete solutions, and GA evolves complete solutions without considering two separate populations of scenarios and MLC behaviors. In all the methods (including MLCSHE), the fitness function is the same as defined in Section 2.5.2. The results of RS will show how difficult the search problem is. Furthermore, the comparison between MLCSHE and GA will show how effective our CCEA-based method is compared to a standard search method.

[6] C_V is computed via the post-processing function *postProcess* shown in Algorithm 1.

[7] We actually confirmed that, when $d_{th} > 0.3$, even MLCSHE yields insufficient DBS for safety monitoring. The results for $d_{th} > 0.3$ are also included in our replication package [121].

For all methods, we set the total number of simulations as the search budget to 1,300 (i.e., around 2.5 days to run with two parallel simulations per run), which was a large enough number to see the convergence of the effectiveness metrics on our preliminary evaluation. Since most of the execution cost is dedicated to running simulations, the computation budget of the experiments is mainly determined by the number of simulations. Thus, we use the total number of simulations as the search budget. Note that, for MLCSHE and GA, the actual number of simulations could be slightly more than the predefined total number since population-based method check if the search budget is exhausted only after the completion of one generation. In addition to the search budget, to ensure the comparability, we set the same *boundary threshold probability* (i.e., p_{th}) and the same maximum number of obstacle trajectories per *mlco* to 0.1 and 2, respectively, for all the methods. This makes a complete solution to have 7 ($scenario$) + 2×11 ($mlco$) = 29 dimensions.

MLCSHE and GA have additional hyperparameters. For GA, we used recommended values in [91]; the population size, the mutation rate, and the crossover rate are set to 60, 0.01, and 0.85, respectively. However, since there are no suggested values for CCEAs, for which there is much less experience, we decided to tune them on two benchmark problems, namely MTQ and Onemax, that are widely used in evaluating CCEAs [83]. As a result, we used the following hyperparameters for MLCSHE: population size = 10, maximum population archive size = 3, mutation rate = 1.0, and crossover rate = 0.5. The reason for the high mutation rate is to compensate for the individuals in the population archives that are directly passed to the next generation without mutation and crossover in CCEAs. Similarly, regarding the distinctiveness threshold for population archives (d_a) in MLCSHE, we set it to 0.4 based on the two benchmark results.

To account for the randomness of the search-based methods, we repeat the experiments for each method 10 times. To evaluate the statistical significance of the difference in effectiveness metrics of different search methods, we use the Mann-Whitney U test [87]. To measure the effect size of the differences, we measure Vargha and Delaney's $\hat{A}_{AB}$, where $0 \leq \hat{A}_{AB} \leq 1$ [138]. Typically, the value of $\hat{A}_{AB}$ indicates a small, medium, and large difference (effect size) between populations A and B when it is higher than 0.56, 0.64, and 0.71, respectively.

Results

Table 2.1 reports the *DBS* achieved by MLCSHE, RS and GA over 10 runs at various distinctiveness (distance) threshold (d_{th}) and boundary closeness (fitness) threshold (t_b) values.

Overall, for all three boundary-seeking methods, *DBS* values increase as boundary closeness threshold (t_b) values increase. This is expected since increasing the value of t_b results in more boundary solutions to consider. Further, *DBS* values drop rapidly as distinctiveness threshold (d_{th}) values increase. This is also expected since identifying complete solutions that are distinct enough with respect to higher d_{th} values becomes quickly more challenging. However, when $t_b = 0.01$, the *DBS* equals to zero for all methods

Table 2.1: *DBS* values for different search methods at different values of t_b and d_{th}.

		Average $DBS \pm 0.5 \times CI_{0.95}$					
		$t_b = 0.01$	$t_b = 0.03$	$t_b = 0.05$	$t_b = 0.10$	$t_b = 0.15$	$t_b = 0.20$
$d_{th} = 0.1$	RS	0.0	0.0	0.0	0.5 ± 0.4	5.1 ± 2.0	19.0 ± 8.8
	GA	0.0	12.0 ± 6.1	37.2 ± 8.9	66.7 ± 11.4	82.8 ± 10.8	94.1 ± 15.4
	MLCSHE	0.0	0.0	39.6 ± 17.2	166.8 ± 33.2	247.5 ± 29.2	315.4 ± 35.1
$d_{th} = 0.2$	RS	0.0	0.0	0.0	0.5 ± 0.4	4.6 ± 1.5	17.0 ± 7.3
	GA	0.0	3.8 ± 1.6	9.2 ± 2.5	18.8 ± 3.3	26.0 ± 2.6	29.8 ± 2.8
	MLCSHE	0.0	0.0	19.4 ± 6.8	57.5 ± 6.7	82.0 ± 5.2	101.6 ± 6.5
$d_{th} = 0.3$	RS	0.0	0.0	0.0	0.4 ± 0.3	2.6 ± 0.6	6.7 ± 2.1
	GA	0.0	1.5 ± 0.7	3.7 ± 0.9	8.0 ± 1.0	11.2 ± 1.2	12.3 ± 1.1
	MLCSHE	0.0	0.0	7.9 ± 1.8	17.9 ± 1.1	23.1 ± 1.7	25.6 ± 2.2

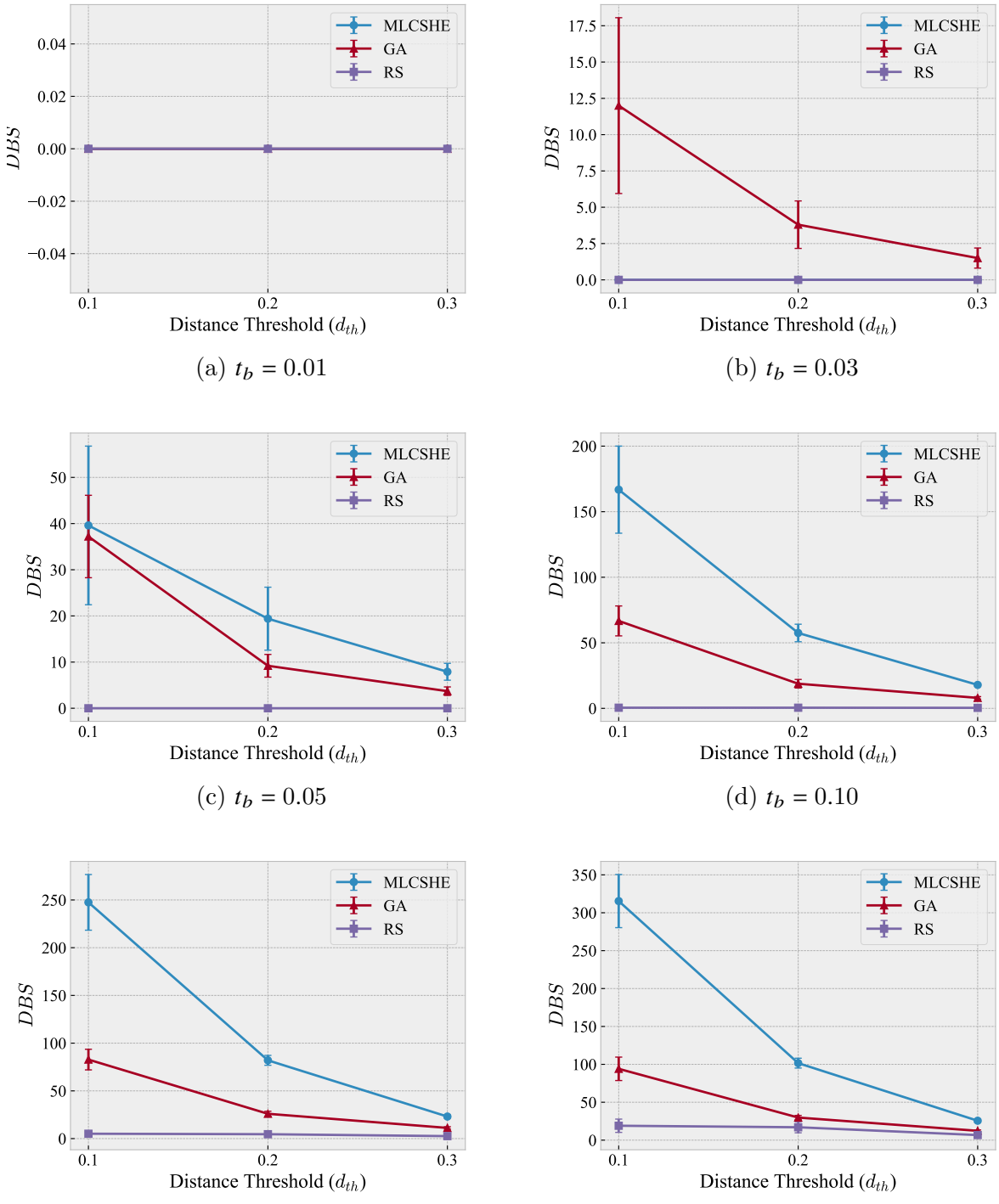
regardless of d_{th}, meaning that none of the methods found complete solutions closer to the boundary than 0.01. This is simply because the provided search budget (1,300 simulations) is not enough to decrease the fitness values of complete solutions below 0.01 by reducing the size of confidence intervals (i.e., Equation 2.3).

Figure 2.5 depicts how the *DBS* values of the different methods vary with increasing d_{th} for different t_b values. In each plot, the x-axis is d_{th} and the y-axis is the average *DBS* over 10 repeats. The *DBS* values for MLCSHE, GA, and RS are marked with circles, triangles, and squares, respectively. The 95% confidence intervals for the average *DBS* values are also shown as error bars.

First, RS achieves extremely low *DBS* values when compared to the other two methods in all cases. This implies that the problem of identifying MLAS boundaries is sufficiently challenging for RS not to be able to satisfactorily address it.

Regarding MLCSHE and GA, we can see different patterns depending on different t_b values. When $t_b = 0.01$, due to the limited search budget provided as already discussed above, $DBS = 0$ for all the methods, meaning that none of the methods find distinct complete solutions near the boundary. When $t_b = 0.03$, MLCSHE once again does not find any complete solutions near the boundary, whereas GA finds some (e.g., 12.0 ± 6.1 for $d_{th} = 0.1$), which are likely not applicable for safety monitoring as a very focused and limited part of the hazard boundary is covered by these solutions. However, when $t_b \geq 0.05$, MLCSHE finds more complete solutions near the boundary (e.g., 39.6 ± 17.2 for $d_{th} = 0.1$ and $t_b = 0.05$) than GA. Furthermore, for the same d_{th} value, the gap between MLCSHE and GA increases significantly as t_b increases.

A plausible explanation for these results is that, although GA is better than MLCSHE in terms of exploiting a specific region, leading to higher *DBS* values when the boundary closeness threshold is very low compared to the provided simulation budget (i.e., when $t_b = 0.03$), MLCSHE successfully uses a cooperative co-evolutionary algorithm which decomposes a high-dimensional problem into two lower-dimensional sub-problems, making the search more effective than GA in terms of identifying distinctive (diverse) complete solutions near the boundary for higher t_b values. Furthermore, MLCSHE takes advantage of population archives that not only carry information regarding the highest-performing



(a) $t_b = 0.01$

(b) $t_b = 0.03$

(c) $t_b = 0.05$

(d) $t_b = 0.10$

(e) $t_b = 0.15$

(f) $t_b = 0.20$

Figure 2.5: The relationship between d_{th} (distinctiveness threshold) and DBS (distinct boundary solutions) along with their confidence intervals (shown as error bars) for MLCSHE, GA, and RS for different t_b (boundary closeness threshold) values.

Table 2.2: Statistical comparison of DBS values for different search methods at different values of t_b and d_{th}. Comparisons with no results are specified as N/A. Such cases happen when A or B have no samples to compare.

	Comparison		*DBS*											
	A	B	$t_b = 0.01$		$t_b = 0.03$		$t_b = 0.05$		$t_b = 0.10$		$t_b = 0.15$		$t_b = 0.20$	
			p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$
$d_{th} =$ 0.1	*MLCSHE*	*RS*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	1.46×10^{-4}	1.00	1.78×10^{-4}	1.00	1.83×10^{-4}	1.00
	MLCSHE	*GA*	*N/A*	*N/A*	*N/A*	*N/A*	1.00	0.51	4.40×10^{-4}	0.97	1.83×10^{-4}	1.00	1.83×10^{-4}	1.00
	RS	*GA*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	1.46×10^{-4}	0.00	1.78×10^{-4}	0.00	1.83×10^{-4}	0.00
$d_{th} =$ 0.2	*MLCSHE*	*RS*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	1.44×10^{-4}	1.00	1.73×10^{-4}	1.00	1.77×10^{-4}	1.00
	MLCSHE	*GA*	*N/A*	*N/A*	*N/A*	*N/A*	4.88×10^{-2}	0.77	1.79×10^{-4}	1.00	1.81×10^{-4}	1.00	1.73×10^{-4}	1.00
	RS	*GA*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	1.44×10^{-4}	0.00	1.73×10^{-4}	0.00	6.32×10^{-3}	0.14
$d_{th} =$ 0.3	*MLCSHE*	*RS*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	1.38×10^{-4}	1.00	1.63×10^{-4}	1.00	1.70×10^{-4}	1.00
	MLCSHE	*GA*	*N/A*	*N/A*	*N/A*	*N/A*	3.35×10^{-3}	0.89	1.74×10^{-4}	1.00	1.72×10^{-4}	1.00	1.75×10^{-4}	1.00
	RS	*GA*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	1.36×10^{-4}	0.00	1.62×10^{-4}	0.00	5.45×10^{-3}	0.13

individuals but also enforce diversity among archive members.

Our visual observations are supported by the results of the statistical comparisons provided in Table 2.2. Columns A and B indicate the search methods being compared. Columns p and $\hat{A}_{AB}$ indicate statistical significance and effect size, respectively, when comparing A and B in terms of *DBS* at different t_b and d_{th} values. Comparisons with no results are denoted as N/A; it happens when A or B have no boundary search results to compare. Given a significance level of $\alpha = 0.01$, the differences between MLCSHE and other methods are significant when $t_b \geq 0.05$, except when $t_b = 0.05$ and $d_{th} = 0.1$—that is when the very low threshold makes it infeasible to find many complete solutions that are both *distinct enough* from each other and *close enough* to the hazard boundary—for which the average *DBS* of MLCSHE is only slightly higher than that of GA. Moreover, $\hat{A}_{AB}$ is always greater than 0.71 when A = MLCSHE, indicating that MLCSHE always has a large effect size when compared to other search methods. Therefore, we conclude that, for t_b values that require practical numbers of simulations to find a sufficient number of distinct boundary solutions for safety monitoring, MLCSHE yields better results than GA and RS.

For boundary closeness thresholds that require practical numbers of simulations to find a sufficient number of distinct boundary solutions for safety monitoring, MLCSHE is significantly more effective than GA and RS with high effect size, meaning that MLCSHE finds significantly more diverse regions near the hazard boundary.

2.6.3 RQ_2: Efficiency

Methodology

To answer RQ_2, we follow the same methodology as for RQ_1, including the hyperparameters and 10 repeats for each method, except for the search (simulation) budget. Specifically, we measure *DBS* across different methods while varying the simulation budget from 10% (130 simulations) to 100% (1300 simulations) in steps of 10%. We then report and analyze how the effectiveness values of different methods vary over time.

Results

Based on the data we collected in our experiment, we analyzed how all different threshold values for d_{th} and t_b affect the relationship between the percentage of simulation budget consumed and the average *DBS* values for 10 runs across MLCSHE, GA, and RS. In Figure 2.6, we selected three t_b values (0.03, 0.05, and 0.15) that, together, are representative of the overall trends. The remaining plots[8] are available in the supporting material (see Section 2.6.5).

On the one hand, Figure 2.6a shows that only GA finds a few boundary solutions when $t_b = 0.03$. Although GA does not reach a plateau for $d_{th} \leq 0.2$, the numbers of distinct boundary solutions found by GA are not enough for safety monitoring as already discussed in Section 2.6.2. On the other hand, Figure 2.6b and Figure 2.6c show that MLCSHE leads to significantly higher *DBS* once the consumed budget is above 10%, except when $t_b = 0.05$ and $d_{th} = 0.1$ for reasons that we already discussed in Section 2.6.2.

We suspect that the results during the first 10% of the simulation budget can be explained by the initial overhead of MLCSHE: since it simulates all possible complete solutions that can be generated by joining the scenario and MLC output populations in the first generation, it could complete only one search generation while GA could complete two or more generations. However, MLCSHE continues to find new distinct complete solutions near the boundary as the budget increases, whereas GA quickly starts to stagnate and reach a plateau. As a result, after only spending 20% of the total budget, MLCSHE always significantly outperforms GA.

Note that, even though we had to set the maximum simulation budget to 1,300 simulations due to the large size of experiments and the unavoidable limitations in computational resources, the *DBS* values of MLCSHE keep increasing until the budget is exhausted for practical boundary closeness thresholds ($t_b \geq 0.05$). This suggests that MLCSHE is able to find considerably more boundary solutions with more simulation budget, when available.

> MLCSHE is significantly more efficient than GA and RS for practical boundary closeness thresholds: MLCSHE finds significantly more diverse regions that overlap with the hazard boundary at a faster rate than GA and RS.

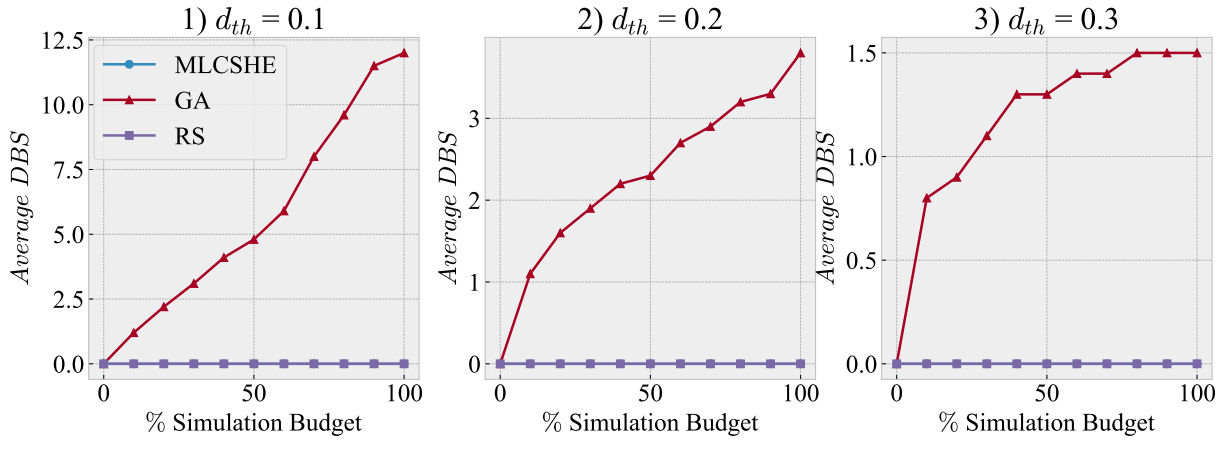
2.6.4 Discussion

Interpretability of Boundary Region

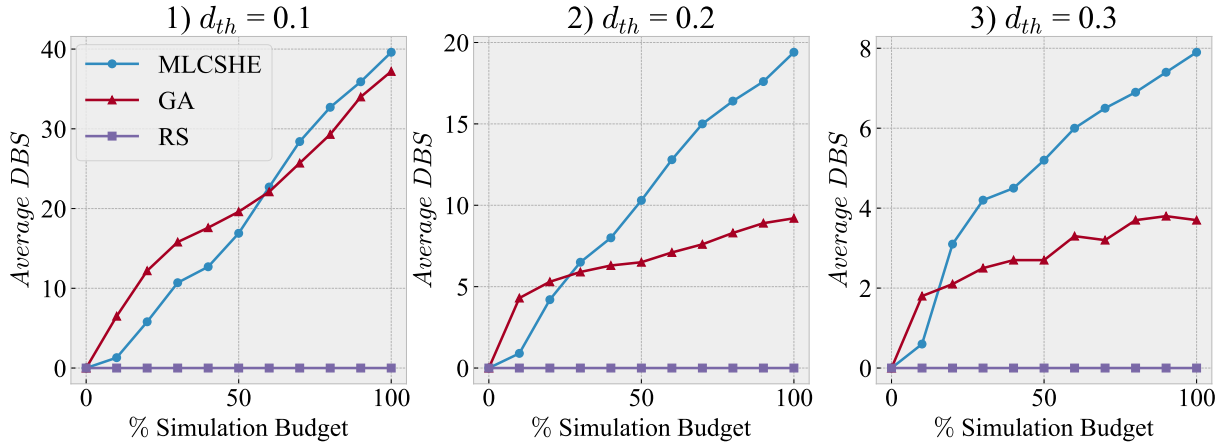
Since the boundary complete solutions found by MLCSHE and other methods are for safety monitoring, one might wonder if we could obtain interesting insights from such boundaries regarding the characteristics leading to high risks of safety violations.

However, the concept of meaningful boundaries is not relevant here as we are not looking for boundaries for a specific MLC implementation but for boundaries that should not be

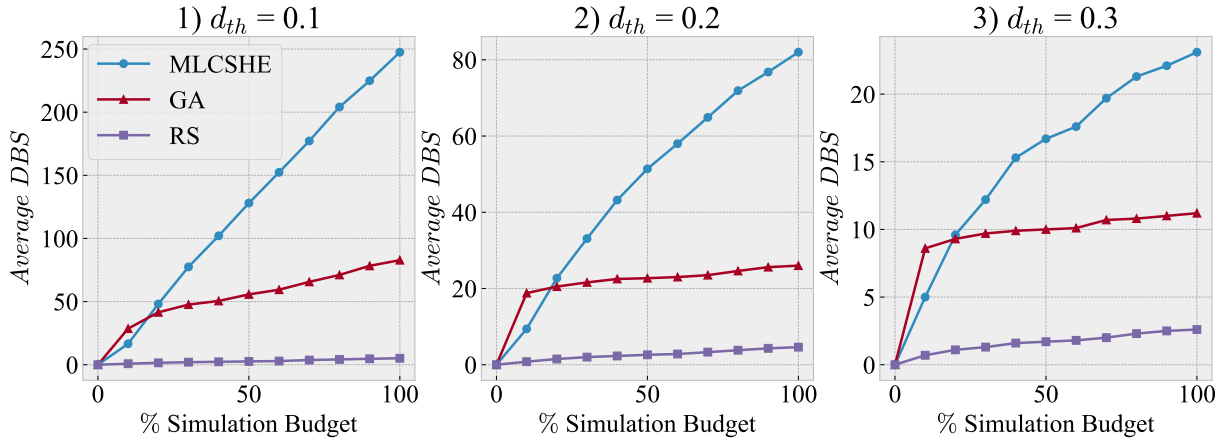
[8]The plots for $t_b = 0.1$ and $t_b = 0.2$ are very similar to Figure 2.6c.



(a) $t_b = 0.03$



(b) $t_b = 0.05$



(c) $t_b = 0.15$

Figure 2.6: Plots of *DBS* vs. % *simulation budget* for MLCSHE, GA, and RS, with $d_{th} \in \{0.1, 0.2, 0.3\}$ and $t_b \in \{0.03, 0.05, 0.15\}$.

approached by any implementation for given scenarios. Incorrect MLC implementations can yield arbitrary outputs that may lead, for certain scenarios, to violations. Why that is the case for certain scenarios and not others is extremely difficult to explain as it requires going into the details of how the system uses these outputs across different scenarios.

Nevertheless, the violations identified by MLCSHE are the result of executing the system (including the MLC) in interaction with the simulation environment. Therefore, they are *real* violations regardless of whether engineers can interpret them.

Threats to Validity

This section discusses potential threats to the validity of our results, namely internal, external, conclusion and construct validity [127, 145, 156].

Internal Validity. Internal validity is concerned with the accuracy of the cause-and-effect relationships established by the experiments.

As mentioned in Section 2.6.2, the actual number of executed simulations is slightly higher than the allocated simulation budget (1,300) for the population-based methods (i.e., MLCSHE and GA). Although the same budget should be used for different methods for a fair comparison, the deviations are so small (less than 5% of the allocated budget) that they cannot significantly impact the results in terms of effectiveness and efficiency.

Another potential threat to internal validity is that the hyperparameter values for GA can affect the results. For example, one might want to intentionally increase the mutation rate of GA to improve the diversity of the complete solutions found by GA. However, it could make GA similar to a Random Search (RS) and could substantially reduce its performance [91], which was also confirmed in our preliminary evaluation results. To mitigate this threat, as mentioned in Section 2.6.2, we relied on the values recommended by Mirjalili [91], which are commonly used in the literature.

External Validity. External validity is concerned with the generalizability of the results.

One notable factor to consider is related to the fact that we have relied only on a specific ADS (Pylot) and simulator (CARLA). However, CARLA is a widely used open-source, high-fidelity simulator, and Pylot was the only component-based AV among those high-ranking in the CARLA leaderboard [1] at the time of our evaluation. Moreover, running the experiments on Pylot and CARLA took more than 75 days of execution, even with parallelization, making it infeasible to consider additional evaluation subjects. Nonetheless, further studies involving other ML-enabled Autonomous Systems in autonomous driving as well as other domains, such as aerospace, agriculture, and manufacturing, are required.

Also resulting from the high cost of running experiments is the fact that we could not evaluate different design choices for MLCSHE using the ADS case study, namely hyperparameters, individual fitness assessment, and archive update strategies. This might impact the generalizability of our results. To account for this factor, we relied on two widely used benchmark problems which are widely used by the literature, as referred to in Section 2.5.3, to tune the hyperparameters of MLCSHE and decide between alternative strategies.

The generalizability of our results is also affected by the fact that a specific ODD, i.e., urban driving, was considered for the evaluation. Changing the ODD to highway driving, for example, changes the lower and upper bounds of the scenario parameters, as well as the complete solutions that will be discovered close to the hazard boundary. However, urban driving is one of the most complex driving ODDs where complicated interactions (and safety violations) between many cars and pedestrians can occur, e.g., at an intersection. Thus, the hazard boundary related to the urban driving ODD is expected to have a more complex shape than a simpler ODD such as highway driving. Furthermore, additional ODDs could not be considered due to time and resource constraints, as described above.

The specific encoding of scenarios and MLC outputs would be another generalizability factor since it determines the search space, which could significantly affect the effectiveness and efficiency of each search method. However, for the large search space problems that are common in practice, we expect MLCSHE to fare increasingly better than GA and RS since MLCSHE is designed to decompose high-dimensional problems into lower-dimensional subproblems.

Conclusion Validity. Conclusion validity is concerned with the conclusions that can be drawn from the collected data and their statistical significance. The experiments could only be repeated 10 times, which is less than the widely accepted rule-of-thumb of 30 repetitions. However, as mentioned in Section 2.6.1, more than 1,800 hours were consumed to run the experiments with 10 repetitions. To account for the statistical error associated with the lower number of repetitions, we report every statistical value with its confidence interval.

Construct validity. Construct validity is concerned with the degree to which the measured variables in the study represent the underlying concept being studied. In our case, the concept of hazard boundary coverage is operationalized by the *DBS* value, which sufficiently captures both concepts of *closeness* to the hazard boundary (via t_b) and *coverage* of the hazard boundary in diverse regions (via d_{th}) at the same time.

2.6.5 Data Availability

The search algorithms (i.e., MLCSHE, GA, RS), the parallel simulation execution module, and the postprocess script are all implemented in Python. The replication package, including the aforementioned implementations, the instructions to set up and configure Pylot and CARLA, the detailed descriptions of the initial conditions used in the experiments, and the detailed results, is available at [121].

2.7 Conclusion

In this chapter, we presented MLCSHE, a cooperative coevolutionary search algorithm to effectively and efficiently approximate the systemic hazard boundary of a machine learning component embedded in an ML-enabled autonomous system, given a system-level safety requirement. We address the challenge of the high-dimensional search space and expensive high-fidelity simulations by using cooperative coevolutionary search, which decomposes

the problem into two smaller subproblems. We rely on a probabilistic fitness function that guides the search towards the boundary of probabilistic unsafe regions. We apply the method to an AV case study, where we run large-scale experiments with parallel simulations to evaluate the effectiveness and efficiency of MLCSHE. The evaluation results show that, for practical boundary closeness thresholds, MLCSHE is significantly more effective and efficient than random search and a standard genetic algorithm in identifying diverse boundary regions.

Over the next chapter (chapter 3), we propose a method that can leverage information found in a hazard boundary, i.e., learned component outputs and scenario combinations to monitor the system at runtime and predict safety violations in the near future.

Chapter 3

Safety Monitoring

This chapter addresses thesis objective TO_2, i.e., monitoring the system for safety violations at runtime using learned component outputs and operational contexts (scenarios) of the system. The contents of this chapter have been submitted (arXiv report is available online [124]).

3.1 Overview

As discussed in section 1.1, runtime monitoring of the operational context and the learned components outputs is crucial in developing effective safety monitors that can identify transitions of the system from safe to hazard states, which can lead to safety requirement violations.

However, monitoring the impact of a learned component on learning-enabled system safety poses several significant challenges. First, many safety-critical learning-enabled autonomous systems are developed by system integrators who are developing the system using various components, including learned components, many of which are developed by third parties. Thus, system integrators often do not have access to the training or test data of the learned component, nor to white-box information such as their architecture or neuron weights. Second, safety monitors should be able to monitor not only the outputs of the learned component over time but also the operational context, which typically includes static parameters such as weather, and may also include dynamic parameters such as the trajectory of other vehicles in proximity to an AV. Third, in a safety-critical context, the safety monitor must predict a safety violation early enough to allow the system or a user sufficient time to mitigate it. As such, efficiency is a key requisite, which translates to the necessity of developing monitors that exhibit a low reaction latency and do not exceed the practical limits of onboard computing units, as opposed to cloud-based alternatives.

Currently, existing methods fail to address all of the above challenges as they monitor for learned component mispredicitons, as opposed to system safety violations [41,45,47,131, 133, 140, 152]. Furthermore, many of the proposed methods rely on internal information sources from the learned component [46, 99, 131].

To address the above challenges, we propose a safety monitoring method based on the idea of predicting the near-future values of a safety metric, i.e., the objective measure used to determine the satisfaction or violation of a safety requirement [8], given the history of learned component outputs and the operational context of the system.

Given the safety-criticality of learning-enabled autonomous systems, where the cost of not predicting safety violations at runtime is very high, instead of relying on single forecast values for each timestep, our method predicts the probability distribution of the safety metric and relies on its tail-end values to conservatively predict safety violations. We leverage Deep Learning (DL) based probabilistic time series forecasters and empirically evaluate state-of-the-art models in terms of prediction accuracy as well as average inference latency and runtime computation resource usage.

Chapter Structure. section 3.2 provides the necessary background on time series forecasting models and the main DL-based architectures. Section 3.3 formally defines the safety metric forecasting problem and details its challenges. Section 3.4 discusses related work. Section 3.5 presents our proposed safety monitoring method in detail. Section 3.6 provides an empirical evaluation of our method and discusses the results. Whereas, section 3.7 concludes the chapter.

3.2 Background

In this section, we discuss the main characteristics of time series forecasting methods as well as the main Deep Learning (DL)-based time series forecasting architectures.

3.2.1 Time Series Forecasting

Time series forecasting aims at predicting the future values of a time series. As described in Januschowski et al. [63], we can distinguish among forecasting methods along a number of dimensions such as *global vs. local*, *probablistic vs. point*, *computational complexity and costs*, and *data-driven vs. model-based*.

Global vs. Local Forecasting. Local methods involve estimating model parameters independently for each time series, while global methods estimate parameters jointly using all available time series [13, 63]. This distinction is concerned with how model parameters are estimated and does not necessarily imply a specific dependency structure between the time series. For instance, a global model can still assume independence between forecasts for different time series for computational efficiency reasons, even though it estimates parameters jointly [63]. While traditional statistical methods often adopt local approaches, global methods have been utilized in both the statistics and machine learning (ML) communities. Recent trends show that deep neural networks are effective as global models and surpass their earlier mixed results when used as local models [13].

Probabilistic vs. Point Forecasting. Forecasting techniques can also be broadly categorized into probabilistic and point forecasting methods. While point forecasts offer a single

best prediction, probabilistic forecasting methods quantify predictive uncertainty, allowing decision-makers to consider this uncertainty when using the forecast [63]. Methods for handling uncertainty include Bayesian approaches and frequentist approaches like model ensembles and bootstrap sampling [13]. The use of Bayesian approaches in estimating parameter and model uncertainty is well studied in ML literature (for introductory and recent work references refer to the study by Januschowski et al. [63]). The predictive uncertainty for a time series is fully described by the predictive distribution, but probabilistic forecasting methods differ in how they enable users to access this distribution, often providing pointwise predictive intervals or Monte Carlo sample paths [13]. Some methods assume a parametric form of the distribution and return its parameters [13, 118]. Modern ML methods handle uncertainty by estimating quantile functions directly [63]. The results of the M4 competition have demonstrated the accuracy of prediction intervals obtained from ML methods, even though they may lack theoretical underpinnings [63,85]. This highlights the effectiveness of ML approaches in handling uncertainty in forecasting.

Data-driven vs. Model-based Forecasting. Methods commonly associated with machine learning, such as deep neural networks, are characterized by their data-driven nature. These approaches excel at capturing intricate patterns from data without relying on strong structural assumptions. However, their flexibility comes at the cost of requiring large amounts of data to effectively tune the multitude of parameters they possess. For instance, recurrent neural networks (RNNs) can discern complex nonlinear patterns from data, as exemplified by their ability to predict time series with oscillating variance amplitudes [63]. Nevertheless, the risk of overfitting arises due to their capacity to memorize patterns, a challenge that regularization techniques like Dropout [129], aim to mitigate. In contrast, statistical models like AutoRegressive Integrated Moving Average (ARIMA) models and Generalized Linear Models (GLMs) are characterized by their parsimonious parameterization and reliance on assumptions to model patterns [18]. These models require less data to accurately estimate their parameters but are inherently more rigid due to the limitations imposed by their structural assumptions [63]. Furthermore, a study by Kolassa [71] shows that simpler models can sometimes outperform complex, correctly specified ones, showcasing the intricacies of model-driven approaches [63]. Furthermore, model-driven approaches require meticulous feature engineering and model specification. Conversely, data-driven models are often preferred for forecasting tasks that involve a large number of time series, from which complex patterns can be extracted [86]. Moreover, DL-based forecasting models can often be trained on large datasets without the need for problem-specific feature engineering [63].

3.2.2 DL-based Forecasting Architectures

DL-based forecasting models can be categorized into two main categories of architectures, namely *iterative* and *sequence-to-sequence*. The iterative architecture generates forecasts step by step, where the model predicts a one-time step based on the previous hidden state and current available information [13]. The process is repeated until the desired forecast horizon is reached. Iterative models can easily be applied to any forecast horizon length. However, since the generated forecast at each time step has an error, the recursive structure

of iterative models can potentially lead to large errors being accumulated over long forecast horizons [75]. RNN models such as long short-term memory networks (LSTMs) and gated recurrent units (GRUs) are commonly employed in iterative architectures [13]. On the other hand, sequence-to-sequence architectures operate by mapping an input sequence to an output sequence, potentially of different lengths. This architecture consists of two main components: an encoder and a decoder. The encoder transforms the input sequence into a fixed-size context vector, which is then used by the decoder to generate the output sequence of a predetermined length. A typical training instance in this approach includes the target and covariate (static or time series features or embeddings [75]) values up to a specific time point t as input, while the neural network generates a set number of target values beyond time t.

3.3 Problem and Challenges

In this section we cast the learned component safety monitoring problem as a safety metric forecasting problem and discuss its challenges.

3.3.1 Problem Definition

Safety-critical systems such as autonomous vehicles (AVs) or Unmanned Aircraft Systems (UASs) use learned components such as Deep Neural Networks (DNNs) to automate and inform perception, localization, and planning tasks. In this chapter, we use as a running example an Autonomous Centerline Tracking (ACT) software, which is used to ensure accurate and safe UAS taxiing on a runway, by detecting and following certain reference points or a designated path without human intervention. The distance between the system position to such reference point or centerline is called Centerline Track Error (cte). The ACT uses a DNN to estimate the cte from camera images and steer the physical system, e.g., a vehicle or an aircraft, towards the centerline where $cte = 0$.

During its operation, the system must satisfy certain safety requirements such as *"the system shall stay within 5 meters of the centerline"*. Although the learned component and the system have to be thoroughly tested and validated before going into operation, during certain challenging or unexpected execution scenarios, the learned component could contribute to the system violating the safety requirement, with potentially life-critical consequences. Therefore, early run-time prediction of a safety violation is an important endeavor and a prerequisite for developing fallback measures and mitigation strategies [128], that include blocking the output of the learned component from being broadcast throughout the rest of the system. To measure the degree of satisfaction or violation of a safety requirement, safety metrics are used. For example, from the above requirement, the safety metric can be defined as the difference between the actual cte (measured by calculating the difference between the system and centerline GPS locations) of the system and a maximum safe cte threshold of 5 m. Note that the safety metric value varies over time and is therefore calculated at each time step.

More concretely, let s be the ACT system including the learned component m for image-based cte estimation, operating in its environment under an operational scenario u. The latter is represented by static and dynamic properties that exist during system operation, e.g., the angle of the sun, cloud cover, runway properties, or the initial position of the aircraft. For each time step t, the system takes an input $in_{s,t}$ from the camera, thus capturing the state of the environment, and provides a pre-processed (e.g., by drivers or information fusion) image $in_{m,t}$ ready to be consumed by m. m produces a real number $out_{m,t}$ which represents the cte estimate. s processes $out_{m,t}$, generates a steering command $out_{s,t}$, and applies it to the system. The state of the environment relative to s changes based on $out_{s,t}$, and the entire process repeats during the operation of s, whereas the next learned component inputs are partially determined by previously learned component outputs.

For a safety requirement r (e.g., *"the system shall not deviate from the centerline more than* 5 m*"*), we can measure the degree of safety violation of s at time t, denoted by $y_{r,t}$, with a continuous function $f_r(t)$ which determines at time t whether r has been violated ($y_{r,t} = f_r(t) \geq 0$) or how close it has come to violating it ($y_{r,t} = f_r(t) < 0$). Note, that the exact definition of f_r is context-dependent and varies based on the system and the safety requirement of interest. For the ACT system and the safety requirement above, we define f_r as denoted in Equation 3.1.

$$y_{r,t} = f_r(t) \coloneqq |cte_{act}| - |cte_{thr}| \tag{3.1}$$

Whereas, cte_{act} and cte_{thr} are the actual and safety violation threshold values of the centerline track error, respectively. Based on the above context, let $u^{(n)}$ be a given set of environmental conditions in the space of all possible conditions, also referred to as *operational scenarios*. Let $o^{(n)}_{m,t-k:t}$ and $y^{(n)}_{t-k:t}$ be the sequence of observed m's outputs and safety metric values from time $t-k$ to t (where k denotes the *lookback* horizon), given $u^{(n)}$.

Given a *hazard forecast horizon* h[1] , we want to predict the sequence of safety metric values from time $t+1$ to $t+h$, i.e., $\hat{y}^{(n)}_{t+1:t+h}$, using a prediction model g, as expressed in Equation 3.2, as accurately as possible.

$$\hat{y}^{(n)}_{t+1:t+h} = g(h, y^{(n)}_{t-k:t}, o^{(n)}_{m,t-k:t}, u^{(n)}) \tag{3.2}$$

As mentioned in section 1.4, aside from the ACT system mentioned above, this chapter additionally targets a second cases study, i.e., an Autonomous Driving System (ADS) which performs the *lane keeping* functionality autonomously, while relying only on image inputs from the camera. The formal problem definition of the problem provided in this section, especially Equation 3.2, equally apply to the ADS case study. We provide, in subsection 3.6.1, complete details for both the ACT and ADS case studies evaluated in this chapter.

[1]Hazard forecast horizon is the number of timesteps in the future [42], over which we want to predict the values of a safety metric such that safety violations (*hazards*) can be predicted and mitigated or avoided.

3.3.2 Challenges

Given the context and the problem definition provided in subsection 3.3.1, we observe multiple challenges. First, the development of learned components is often outsourced to third parties [49,113], which are later integrated into the main autonomous system. Thus, the limited or lack of access to the learned component details inhibits the application of white-box methods for safety monitoring [133]. Such details include the training data and the model's architecture, weights, activation patterns, or gradients during the feed-forward pass.

Second, as mentioned in section 3.1, evaluating the safety of a learning-enabled autonomous system relies both on the *static* operational context data (*scenario*) and *dynamic* time series data related to the behavior of the learned component and the history of safety metric values over time. Thus, the safety monitor should be able to utilize both types of data to provide an accurate forecast of the safety metric values over the hazard forecast horizon.

Third, safety monitors are often developed for safety-critical cyber-physical systems with limited computation capabilities. Thus, it is paramount that the safety monitor introduces low latency and memory overhead to the system. Although many safety monitoring methods that rely on white-box confidence estimation techniques [41,131] are more accurate than their black-box counterparts, their memory and computing overhead makes their adoption in resource-constrained settings impractical [132].

To address the above challenges, henceforth denoted C1 through C3, respectively, in this chapter we evaluate time series forecasting methods, especially the ones based on DL, to forecast the safety metric values of a learning-enabled system over the hazard forecast horizon. DL methods have been shown to provide accurate forecasts while being amenable to multiple data types (static and time series) and a large number of samples [13,63,86]. In section 3.5, we provide further details on the proposed safety metric forecasting solutions, whereas in section 3.6 we discuss the experimental evaluation of different state-of-the-art forecasting models for the ACT case study.

3.4 Related Work

This section discusses existing studies related to the problem of learned component safety monitoring. Some surveys [15,81,92,110] distinguish between Out-Of-Distribution (OOD) detection and uncertainty estimation (quantification) methods. The former focuses on identifying learned component inputs that are not within its training distribution, while the latter estimates the uncertainty associated with the learned component outputs. Since these methods aim, at a high level, at a similar goal, i.e., identifying inputs that lead to uncertain and thus untrustworthy outputs, they should therefore be discussed here. Next, we discuss the main safety monitoring techniques in the literature, primarily categorized based on the type of system information access they assume, namely black-box and white-box approaches.

3.4.1 Black-box Methods

Black-box methods use information such as learned component inputs and outputs, as well as their training and test datasets,[2] to identify the shift in the distribution of inputs observed during operation from the training input distribution, which can lead to mispredictions during operation [157].

For example, Zhang et al. [153] proposed DeepRoad which was mainly designed for testing AV learned components by validating single input images according to their minimum distance from the training set based on the embeddings generated according to VGGNet [125] features. SelfOracle [133] is a black-box failure predictor that uses an autoencoder and time series-based anomaly detection to reconstruct the input images observed by the learned component and to use reconstruction loss to detect OOD inputs. Similar methods that utilize variational autoencoders (VAEs) to measure an anomaly score have also been proposed by other studies [17,45,52]. DeepGuard, proposed by Hussain et al. [52], uses the VAE reconstruction error to prevent roadside collisions with other vehicles, Borg et al. [17] proposed an OOD detector based on VAEs combined with object detection for an automated emergency braking system.

Moreover, some black-box methods quantify the uncertainty of the learned component outputs, to help practitioners identify the learned component inputs leading to unreliable outputs, by estimating probability distributions of the outputs given past system executions. These methods leverage Bayesian networks or their approximations [7, 36, 84], allowing them to incorporate expert domain knowledge in their Bayesian network models. In the context of autonomous aviation systems (similar to our ACT example), Asaadi et al. [7] used a non-parametric Bayesian-based uncertainty quantifier, i.e., Gaussian Process (GP) regressor, trained on a subset of the learned component training data, to estimate the uncertainty in learned component outputs given its inputs.

3.4.2 White-box Methods

Unlike black-box methods, white-box methods take advantage of internal information sources from the learned component, e.g., model confidence [46], neuron activation patterns [147] or gradients [131], comparing their observations at runtime (during the learned component operation) against design-time (during the learned component training).

For example, Lakshminarayanan et al. [72] proposed the use of an ensemble of neural networks (*Deep Ensembles*) to effectively predict the uncertainty of perception component outputs at runtime. Kendall and Gal [68] proposed a Bayesian deep learning framework that captures uncertainties associated with both the learned component inputs, also referred to as *aleatoric* uncertainty, as well as the model itself, also known as *epistemic* uncertainty, for a perception component (image segmentation and depth regression). In

[2]The survey conducted by Riccio et al. [115] categorizes the methods that require access to learned component train and test dataset as *data-box* methods. However, to avoid confusion, we categorize them as black-box methods here as they do not use any internal information from the model itself.

the context of autonomous driving, Grewal et al. [41] evaluate different uncertainty quantification methods for the misbehavior prediction of failures. Hendrycks and Gimpel [46] used the learned component's own confidence, i.e., its softmax probability distributions, to measure uncertainty in learned component outputs. However, since learned components are prone to generating incorrect outputs (misprediction) with high confidence [99], many methods have leveraged other information sources to estimate uncertainty. ThirdEye [131] uses an eXplainable AI (XAI) technique, namely attention maps, to generate a confidence score for the learned component (in this case a DNN) based on input images and gradients of the DNN. The generated confidence score is then used to predict a failure by comparing it with a failure threshold learned from past system executions (simulations).

3.4.3 Limitations of Existing Methods

Although the white-box and black-box methods mentioned above are effective at evaluating the inputs to the learned component, they do not consider the effect of learned component outputs on system safety. Learned component inputs that can lead to inaccurate outputs (i.e., mispredictions) may not lead to system safety violations, depending on the system's operational context. As discussed in subsection 3.3.2, a safety monitoring method must be able to predict the combinations of system context and learned component outputs that can lead to system-level safety violations.

In terms of information requirements, methods such as DeepRoad [153] and the Bayesian method proposed by Asaadi et al. [7], require access to training and test datasets. Furthermore, as mentioned in subsection 3.4.2, to identify safety-violating inputs, white-box methods rely on internal information of the model [46, 131, 147]. As discussed in subsection 3.3.2, system integrators often do not have access to such information, nor training and test datasets, as they are frequently developed by third parties.

Finally, both the black-box and white-box methods discussed in Sections 3.4.1 and 3.4.2, respectively, were not evaluated in terms of their inference latency and computation resource usage at runtime [7, 36, 84, 131, 133, 147, 153].

Different from the described black-box and white approaches, we evaluate time series DL methods for safety monitoring, a previously unexplored topic. We empirically evaluate the effectiveness of such methods when using both the operational context of the system and learned component behavior while being computationally feasible for runtime monitoring. We then predict when system context and learned component behavior together lead to system-level safety violations.

3.5 Temporal Forecasting of Safety Metrics

In this work, as mentioned in section 3.3, we have cast the safety metric prediction problem as a safety metric forecasting problem. As such, we evaluate existing time series forecasting methods for the problem of predicting the value of the safety metric of a learning-enabled autonomous system over a hazard forecast horizon, as introduced in subsection 3.3.1. Note

that the hazard forecast horizon is set by the system developers and safety engineers according to the system properties, its intended mission and its corresponding set of operational contexts, also referred to as its Operational Design Domain (ODD) [28, 122].

The inputs to the forecasting model are both static operational scenario data as well as dynamic (time-dependent) learned component behavior and past safety metric data of learning-enabled autonomous systems. The output of the forecasting model[3] is the safety metric forecasts over the hazard forecast horizon.

As mentioned in section 3.2, model-based time series forecasting models such as fitting ARIMA models are not suitable for forecasting when the dataset is large or highly-dimensional, or contains non-linear relationships (between features and target), as the time required to fit them to data considerably increases and their prediction performance degrades. Classical machine learning (ML) models, especially tree-based methods (e.g., gradient-boosted tree methods), have been used widely in time series forecasting as they provide superior prediction performance to their statistical counterparts [13]. While they can be trained on a large number of samples, these models require substantial feature engineering, thus requiring expert knowledge of the system [13], which consumes significant time and effort.

Recently, deep learning time series forecasting models (DL forecasters) have shown great potential for challenging forecasting problems. As discussed in section 3.2, DL forecasters can be trained on large time series samples without requiring white-box knowledge of the system under test, or specific feature engineering. Thus, addressing challenge C1 described in subsection 3.3.2 by relying only on black-box information related to the system under test. Moreover, some DL forecasters can handle samples that contain both static and dynamic data types with complex and non-linear relationships, addressing challenge C2 stated in subsection 3.3.2. Last, DL forecasters, given real-valued times series inputs, consume limited computation resources and enable low inference latency, addressing challenge C3 discussed in subsection 3.3.2[4].

Finally, due to the safety-critical nature of learning-enabled systems, it is important to account for the uncertainty associated with predicted safety metric values. As discussed in section 3.2, DL-based probabilistic forecasting methods account for such uncertainty by predicting the values of the safety metric probability distribution. Knowing the values at the tail-end of the predicted probability distribution of the safety metric allows us to rely on such values for worst-case metric predictions.

Thus, to address the challenges outlined in section 3.3, we propose training a DL-based probabilistic forecaster, given historical execution data of the system and its safety metric, such that it provides forecasts of the safety metric value over the hazard forecast horizon. Note that the weights of the DL model are selected and remain the same for all the different scenarios and time series data that are used for training, i.e., it is a *global* model. Concretely, the DL-based probabilistic forecaster takes the times series of the safety metric values and learned component outputs, as well as scenario parameters as input, and

[3]Also referred to as *target variable* in time series forecasting literature [13].

[4]We will empirically measure the inference latency as well as the computation resource usage of state-of-the-art DL forecasters in section 3.6.

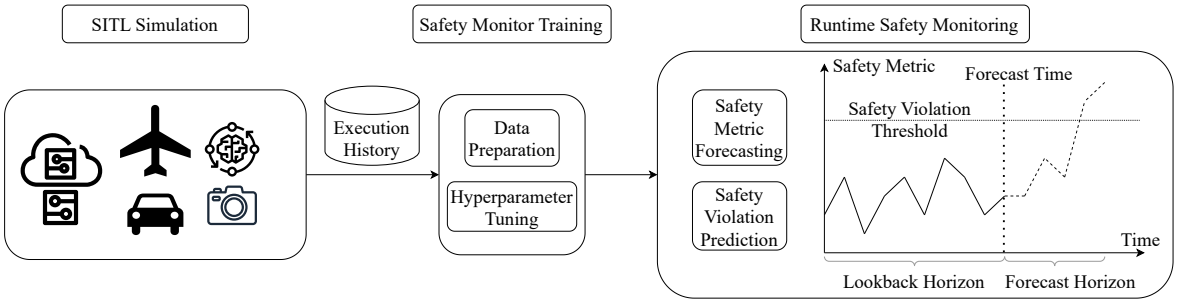


Figure 3.1: The overall process for training the temporal safety metric forecaster for safety monitoring.

returns time series predictions of the safety metric probability distribution. Note that the duration of the input time series is equal to the lookback horizon, while the duration of the predicted time series is equal to the forecast horizon.

Safety Violation Prediction. As discussed earlier in Section 3.3.2, one of the main goals, aside from knowing the value of the safety metric at each timestep, which is crucial for safety-critical decision-making, is to predict safety violations that might occur in the near future, i.e., over the hazard forecast horizon. Therefore, we further describe how a safety metric forecaster can be used to predict safety violations.

Recall that a safety metric forecaster, at timestep t, provides predictions of the safety metric value from timestep $t+1$ to $t+h$. Further recall that we have assumed that the function measuring the safety metric is defined such that non-negative values imply safety violation, similar to Equation 3.1. Thus, if the *maximum* of the predicted safety metric values over the hazard forecast horizon is non-negative, we can say that a safety violation has been predicted.

More specifically, given predicted safety metric values and a hazard forecast horizon h, we can define the safety violation function $v(h)$ as detailed in Equation 3.3.

$$v(h) = sign\big(max\{y_{t+1:t+h}\}\big) \tag{3.3}$$

Note that the sign function $sign(i)$ used in Equation 3.3 returns +1 when $i \geq 0$ and returns -1 otherwise [12, 29]. Based on the definition provided in Equation 3.3, a safety violation is detected over the hazard forecast horizon, i.e., from timestep $t+1$ to $t+h$, if $v(h) = 1$.

Figure 3.1 depicts the overall process for training and deploying the safety monitor. The process starts with data generation using System-in-the-Loop (SITL) simulation where the required data to train the safety monitor, as discussed above, is generated. Then, in the training stage, we preprocess the execution history, tune the hyperparameters of the safety metric prediction model, and train the best model on the complete dataset. Finally, the trained model is deployed during system operation where the future values of the safety metric are predicted, which is in turn used for safety violation prediction.

As surveyed by Benidis et al. [13], various DL models with different architectures have been proposed and applied to time series forecasting. The number and variety of the proposed models make the problem of selecting the appropriate DL model for safety metric and violation prediction an important challenge, which can only be addressed through empirical investigation. To this end, we have empirically evaluated state-of-the-art DL-based time series forecasting models in our specific application context.

3.6 Empirical Evaluation

In this section, we report the empirical evaluation of time series-based safety monitors applied to an ACT and an ADS system. We aim to answer the following research questions:

RQ_1 (Safety Metric Prediction Accuracy) How do different forecasting models score and compare in terms of safety metric prediction accuracy?

RQ_2 (Safety Violation Prediction Accuracy) How do different forecasting models perform and compare in terms of safety violation prediction accuracy?

RQ_3 (Accuracy Sensitivity Analysis) What is the impact of varying lookback and hazard horizon window sizes on safety metrics and safety violation prediction accuracy?

RQ_4 (Resource Overhead Sensitivity Analysis) How do different forecasting models compare in terms of the memory and time overhead of making predictions?

RQ_1 and RQ_2 are motivated by the wide variety of potentially applicable forecasting models, which raises the need for experimental evaluation to determine which one scores best in terms of safety metric forecast and safety violation prediction accuracy, respectively. RQ_2 is particularly relevant in scenarios where a safety monitor does not have a particularly high accuracy in predicting safety metric values, yet its predictions sufficiently contribute to accurate safety violation predictions.

Note that hazard forecast horizon and lookback window sizes are design choices for system developers. Increasing the hazard forecast horizon is expected to decrease safety metric prediction accuracy, as indicated by previous studies [22]. Conversely, increasing the lookback window size is expected to enhance accuracy. However, we also expect such changes to impact models' runtime performance, as they impact the number of model parameters, influencing factors like latency and memory overhead. Given that the forecasters are destined for deployment in resource-constrained safety-critical systems, understanding the consequences of altering window configurations—specified by hazard forecast horizon window size and lookback to forecast window size ratio parameters—on prediction accuracy and runtime performance is crucial for system developers in practice. Consequently, our evaluation further explores the effects of different window configurations on prediction accuracy (RQ_3) and runtime performance (RQ_4).

3.6.1 Evaluation Subject

We evaluate the DL-based forecasting models by applying them to two case studies related to an autonomous centerline tracking system for autonomous taxiing (ACT), and an autonomous driving system (ADS) focused on lane keeping. In this section, we provide for each case study, an overview of the subject system, explain the details of the evaluation dataset and the simulation workflow used to generate it.

ACT Case Study

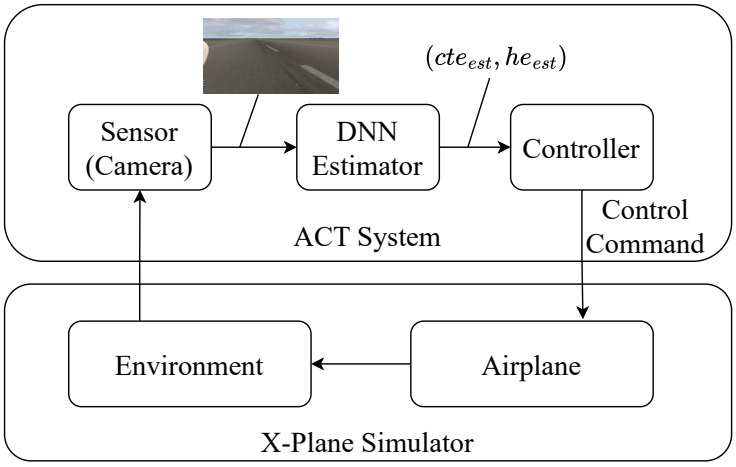
Subject System and Simulation Platform. We used an open-source ACT system [67], similar to previous studies that had ACT-related case studies [8, 26, 103]. As illustrated in Figure 3.2a, the ACT system consists of a camera, a learned component (i.e., a DNN estimator that outputs cross-track error cte and heading error he estimates given an image input), and a proportional controller that generates control commands steering the aircraft.

Previous studies that used ACT as a case study used the TaxiNet model [8, 26, 103], a DNN developed by Boeing for ACT applications, as their learned component. Since we were not granted access to TaxiNet, we relied on the open source version called TinyTaxiNet [67],[5] which has a lower number of deep layers and input vector size. We used X-Plane 11 [146], a high-fidelity flight simulator— which is used for training pilots [146] in all flight phases including taxiing—to control various aircraft and environmental parameters. Based on the simulator's controllable elements, in line with previous studies [67], we considered the following four scenario elements: period of the day, cloud cover, starting cross-track error, and starting heading error. A detailed explanation of the scenario elements and their value ranges, and the TinyTaxiNet model can be found in the supporting material (see subsection 3.6.9).

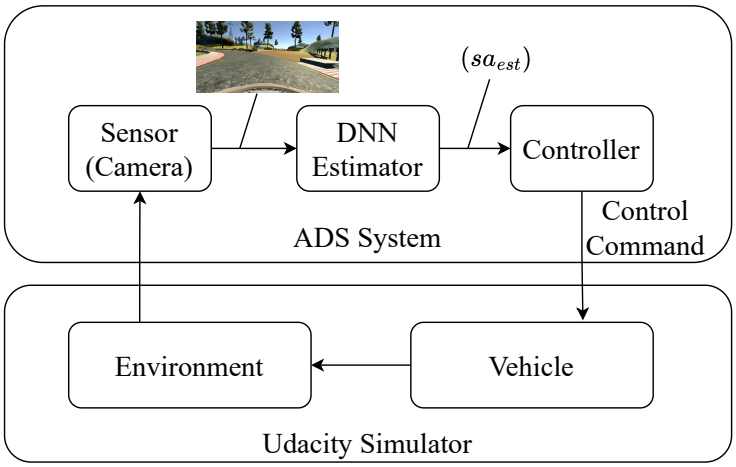
Considering the capability of X-Plane in controlling environmental parameters and the major functionality of our target learned component (i.e., ACT), we focus on the following two safety requirements: 1. *"The aircraft shall have a distance ($|cte_{act}|$) no more than cte_{thr} from the centerline, while taxiing on the runway."* 2. *"The aircraft shall have a heading angle ($|he_{act}|$) no more than he_{thr} from the centerline, while taxiing on the runway."* To compute the safety metric related to the above requirements, we measure the actual distance of the center of the aircraft from the centerline ($|cte_{act}|$), as well as the actual angle between the longitudinal axis of the aircraft and the centerline ($|he_{act}|$), at every time step. Given our definition of the safety metric provided in subsection 3.3.1 (Equation 3.1), negative values of the computed safety metric ($|cte_{act}| < cte_{thr}$) imply that the cte safety requirement is *not* violated, whereas zero and positive values ($|cte_{act}| \geq cte_{thr}$) indicate a safety violation. Note that the same applies to the he safety requirement. Based on the size of the aircraft and its taxiing speed, we set cte_{thr} to 5 m and he_{thr} to 5°, in line with other ACT-related studies [6, 7, 103].

Evaluation Dataset. Given the input and output of the safety metric forecaster, discussed in section 3.5, the dataset used to train and evaluate the DL-based forecasters must

[5] To the best of our knowledge, this is the only open-source DNN model trained for ACT.



(a) ACT.



(b) ADS.

Figure 3.2: System-in-the-loop simulation of the ACT system and the X-Plane simulator (a), and the ADS system and the Udacity simulator (b).

contain time series values of the learned component's output and the safety metric, given a specific scenario. Thus, we generated a dataset based on the autonomous taxiing aircraft case study (section 3.6.1, inspired by the simulation setup proposed by [67]).

Concretely, we used the Latin Hypercube Sampling (LHS) method to generate 1,996 unique scenarios to ensure coverage of the scenario parameter search space [82]. LHS is a sampling method that is used to generate near-random samples from a multidimensional distribution, while ensuring uniform coverage of the simulation input space (i.e., *scenario space*) by stratifying each input dimension [82,120]. Given that the high-fidelity simulations that we require to generate the dataset are computationally expensive, LHS allows us to obtain diverse scenarios with a limited number (1996) of simulations. We executed each scenario using the ACT simulation stack until the aircraft reaches to its destination on the runway (taking an average duration of 200 s), whereby time series data of TinyTaxiNet outputs (cte_{est}, he_{est}) are recorded, as well as the safety metrics (computed based on Equation 3.1, with $cte_{thr} = 5\,\text{m}$ and $he_{thr} = 5°$, as discussed in section 3.6.1). Thanks to our sampling strategy, 52% and 21.9% of the scenarios recorded in the dataset for the cte and he safety requirements, respectively, include safety violations in their test set (the division of the dataset is discussed in section 3.6.3). We therefore generate a large number of diverse cte and he safety violations in the test sets. In many simulations, the ACT system misidentifies an imaginary line, which is a parallel offset of the runway centerline by more than 5 m, as the actual centerline and continues taxiing along it. This can explain why the dataset includes more cte safety violations than he safety violations.

Finally, following best practice, similar to the guidelines provided by previous studies on training and evaluating DL-based time series forecasting models [74, 118, 141], we normalized the time series data, i.e., cte_{est}, he_{est} and the safety metrics values using Z-score normalization, thereby reducing model bias caused by differences in time series magnitudes among various parameters and scenarios [76]. Further details regarding our generated dataset can be found in the supporting material (see subsection 3.6.9).

ADS Case Study

For the ADS case study, we relied on the artifacts available in the paper by Stocco et al. [131], where the authors tested a lane keeping ADS in a driving simulator [131]. The dataset includes not only the time series of the learned component outputs and the scenario parameters but also the time series of the safety metric, which is crucial for our safety monitoring method. Over the rest of this section, we provide an overview of the ADS subject system and the simulation platform used to generate the dataset. We then discuss the details of the raw dataset from the study by Stocco et al. [131], as well as our preprocessing steps leading to the final dataset used by our study.

Subject System and Simulation Platform. The ADS case study involves a lane keeping ADS widely used in previous studies [51, 62, 115, 131, 133, 133]. As depicted in Figure 3.2b, the ADS consists of a camera, a learned component (i.e., a DNN which estimates the required *steering angle* sa_{est} to keep the vehicle within the lane given an image input), and a controller that issues the control commands to the vehicle. The learning component

is based on the NVIDIA Dave-2 model architecture [16], a DNN-based steering angle estimator that is trained with a set of images collected while a human driver is driving a vehicle. The simulator used to evaluate the ADS case study was the Udacity simulator for self-driving cars [137], a driving simulator which has been widely used in the ADS testing literature [51, 62, 62, 131–133]. Udacity provides close-loop tracks to simulate an ADS driving under various scenario conditions. Based on the controllable elements of the Udacity simulator, Stocco et al. [131] considered the following two scenario elements[6]: weather conditions (i.e., clear, fog, rain, and snow), and weather intensity. Since the authors record the time series of cte_{act}, we use the recorded value at the beginning of each episode as our third scenario element, i.e., the starting cross-track error, which indicates the initial position of the vehicle. A detailed explanation of the value ranges for the scenario parameters can be found in the supporting material (subsection 3.6.9).

Considering the main functionality of the target learned component, i.e., lane keeping, the following safety requirement is considered: *"The vehicle shall have a distance ($|cte_{act}|$) no more than cte_{thr} from the centerline, while driving on track".* To compute the safety metric, measurements of the cte_{act} is required, which is contained in the dataset provided by Stocco et al. [131]. Similar to the cte safety requirement for the ACT case study, the safety requirement is violated when $|cte_{act}| \geq cte_{thr}$ and not violated otherwise. Given that the total width of the track set in the simulator, the size and the speed of the vehicle, we set cte_{thr} to 5 m, which is reasonable as it provides the vehicle less than 0.5 m of side clearance from the edge of the track.

Evaluation Dataset. The dataset generated by Stocco et al. [131] contains a time series of the Dave-2 outputs (estimated steering angles), a time series of cte_{act} (which we used to compute the time series of the safety metric, as described above), and the scenario parameters.

Concretely, the authors executed the ADS in the Udacity simulator, i.e., let it drive for one lap around the track under various weather conditions with intensity increments of 10%. Therefore, to cover a diverse range of the scenario space, the authors recorded 31 one-lap simulations (1×clear+10×fog+10×rain+10×snow = 31) which include the time series of cte_{act} measurements [131]. Note that in some scenarios, the ADS drives the car out of the track, in which case the car is reset on the next waypoint on the track. This reset during the execution of the scenario leads to a discontinuity during the execution of the learned component output and safety metric time series. Therefore, we divide the executions at the points where the vehicle has gone out of the track completely into separate episodes, to handle the discontinuity in the time series data. However, this has led to having diverse execution lengths, e.g., from 5 s to more than 100 s. To make the size of the episodes more uniform, we discarded the very short episodes, i.e., less than 15 s, as they do not contain the minimum number of timesteps required to train and test the DL-forecasting models, and divided the larger episodes into shorter chunks. The resulting dataset contains 175 episodes with an average duration of 17.5 s. Note that the size of the ADS dataset is a fraction of (approximately 0.8%) the size of the dataset we generated for the ACT case

[6]Note that we are only mentioning the parts of the study conducted by Stocco et al. [131] that are relevant to our study. We refer the reader to the study itself for comprehensive details on all the evaluations conducted by the authors.

study. As we will see, this will have an impact on our results and conclusions. Despite the ADS dataset size, we observe that 33.7% of the episodes include safety violations in their test set. We therefore observe that the ADS dataset includes, a considerable number of diverse safety violations, with respect to the dataset size, thanks to the sampling strategy used by Stocco et al. [131] to search the scenario space, as discussed above.

Finally, similar to the ACT case study (item 3.6.1), we normalized the time series data, i.e., estimated steering angle and the safety metric values using Z-score normalization. We have included more details about the raw and the preprocessed datasets in our supporting material (subsection 3.6.9).

3.6.2 Models Under Evaluation

In this section, we outline the chosen forecasting models for evaluation, and discuss the hyperparameter tuning process applied to optimize the selected models.

As discussed in section 3.5, we are interested in global univariate probabilistic forecasting models that take in input both dynamic time series and static scenario data and provide probabilistic forecasts of the safety metric. We selected the models for evaluation from the GluonTS library [3], a widely used probabilistic DL-based forecasting Python library, containing the implementations of many state-of-the-art models. From the list of available models in the library, we selected four models, three of which satisfy all the requirements of the safety metric forecasting problem (i.e., a global univariate probabilistic forecasting model capable of processing both static and dynamic inputs), whereas the fourth model acts as a competitive baseline, even though it does not fully satisfy all requirements.

- MQCNN: a sequence-to-sequence model which is the CNN-based variant of Multi Quantile Recurrent Forecaster, using a CNN encoder instead of an RNN [141].
- Temporal Fusion Transformer (TFT): a sequence-to-sequence transformer-based model [74].
- Seq2Seq: a vanilla sequence-to-sequence model with a CNN encoder and an MLP decoder [3].
- DeepAR: an iterative model which utilizes both RNNs and autoregressive techniques to iteratively capture temporal dependencies [118]. Due to DeepAR's architecture, it only takes as input the static scenario parameters and time series of the target variable, i.e., safety metric. Despite DeepAR architecture's lack of ability to process the time series of the learned component output, we have included it in our evaluation as a competitive baseline, due to its high performance in the forecasting benchmarks [86] and wide use in industry [13].

Hyperparameter Tuning. We fine-tuned the hyperparameters of each considered model before answering the research questions, considering the values (fixed or range) retrieved from the original publications. We have tuned the hyperparameters for each safety requirement separately, as they lead to different datasets for the models to be trained and tested on. The relevant hyperparameters and their value ranges for each model are as follows:

- *MQCNN* [141]. *Number of layers in the MLP decoder (or dim)* was selected in the range {2, 4, 8}. *Number of neurons in the hidden layer* was selected in the range {20, 40, 80}. *Number of channels per layer of the CNN encoder* was chosen in the range {20, 40, 80}.

- *TFT* [74]. We set the *dropout rate* to values ranging from 0.1 to 0.3 in steps of 0.1. We took the values of *number of attention heads* and *state size* in the ranges {1, 4} and {80, 160, 320}, respectively. We kept the loss function the same as the original paper, i.e., *quantile (pinball) loss* [74].

- *Seq2Seq*. *Number of layers* was chosen from {2, 4, 8}. *Number of neurons per layer* was selected from {10, 20, 40}.

- *DeepAR* [118]. The *number of RNN layers* was set to 3 as in the original study [118]. The *number of RNN nodes per layer* was selected in the range {40, 100}. We selected the type of RNN nodes in each layer as being one among {LSTM, GRU}. We selected the *dropout rate* in the range {0.1, 0.2, 0.3}. The loss function *negative log likelihood* was used, in line with the original study [118].

- *DL Training*. For all the deep learning models above, we selected the hyper-parameters related to training as follows. We chose *learning rate* in the range {0.0001, 0.001, 0.01}. We selected *max gradient norm* in the range {0.01, 1.0, 100.0}. We evaluated *batch size* in the range {64, 128, 256}. The Adam optimizer [69] was used for training all the models, as per the original paper implementations or that of the GluonTS library.

For other hyperparameters, we relied on the suggested values used in the original studies or the default value in their implementation (additional details on the parameter settings are available in the supporting material in subsection 3.6.9).

We trained each model configuration (defined by a combination of hyperparameters) on the training set (i.e., 70% of the dataset) and evaluated it on the validation dataset (i.e., 10% of the dataset), 5 times, to account for randomness, for example, due to random seeds. Similar to RQ_1, we compared the models based on their q-Risk (Equation 3.4) values at quantiles considered in RQ_1. The details of the evaluation metric, i.e., q-Risk and the quantiles under consideration, are presented in section 3.6.3. Table 3.1 summarizes the hyperparameters for each model, their possible values, and the selected hyperparameters, for ***cte*** (ACT_{cte}) and ***he*** (ACT_{he}) safety requirements of the ACT case study and the ***cte*** (ADS_{cte}) safety requirement of the ADS case study, respectively.

Evaluation Hardware. To train each configuration of the models under investigation on the evaluation dataset and evaluate them, we used the following compute resources: 1x NVIDIA V100 GPU with 32GB HBM2 memory, 16 cores of Intel Silver 4216 Cascade Lake 2.1GHz CPU, and 128GB of RAM.

Table 3.1: Hyperparameters of the models under evaluation, for ***cte*** and ***he*** safety requirements of the ACT case study, and the ***cte*** requirement of the ADS case study

Hyperparameter	Value Range	ACT_{cte}	ACT_{he}	ADS_{cte}
Seq2Seq				
Batch Size	64, 128, 256	128	64	64
Learning Rate	1e-4, 1e-3, 1e-2	1e-3	1e-4	1e-4
Gradient Clipping Value	1e-2, 1.0, 1e+2	1.0	1e-2	1e+2
Number of MLP Decoder Layers	1, 2, 4	2	2	2
Number of Neurons per MLP Layer	20, 80	80	20	80
DeepAR				
Batch Size	64, 128, 256	64	64	64
Learning Rate	1e-4, 1e-3, 1e-2	1e-2	1e-3	1e-3
Gradient Clipping Value	1e-2, 1.0, 1e+2	1e-2	1.0	1.0
RNN Node Type	LSTM, GRU	GRU	GRU	LSTM
Number of RNN Nodes	40, 100	40	40	100
Dropout Rate	0.1, 0.2, 0.3	0.1	0.1	0.1
TFT				
Batch Size	64, 128, 256	256	256	128
Learning Rate	1e-4, 1e-3, 1e-2	1e-3	1e-3	1e-2
Gradient Clipping Value	1e-2, 1.0, 1e+2	1e-2	1.0	1e+2
State Size	40 , 80, 160	160	160	160
Number of Attention Heads	1, 4	4	4	4
Dropout Rate	0.1, 0.2, 0.3	0.1	0.1	0.1
MQCNN				
Batch Size	64, 128, 256	256	64	128
Learning Rate	1e-4, 1e-3, 1e-2	1e-3	1e-4	le-4
Gradient Clipping Value	1e-2, 1.0, 1e+2	1.0	1e-2	1.0
Number of MLP Decoder Layers	1, 2, 4	2	2	2
Number of Neurons per MLP Layer	20, 80	20	20	80
Number of Channels	20, 40	20	20	20

3.6.3 RQ_1: Safety Metric Forecast Accuracy

In this section, first we provide the details of our evaluation methodology to answer RQ_1. Then, we present the results for the autonomous taxiing (ACT) case study, followed by the results of the autonomous driving (ADS) case study. Finally, we draw conclusions from both case studies and present our answer to RQ_1.

Methodology

To answer RQ_1, we divide each dataset (detailed in section 3.6.1) into training, validation, and test datasets, which correspond to 70%, 10% and 20% of the dataset, respectively. The training set is used to train the time series forecasting models, whereas the validation dataset is used for hyper-parameter tuning (subsection 3.6.2). Finally, we generated predictions using the trained models on the test dataset, which is disjoint from the training and validation datasets. To avoid *look-ahead bias* [148], we used time-based splitting [13, 75, 105], such that all the samples in the test dataset occur after the validation dataset, whose samples occur after the training dataset.

The most accurate model resulting from the hyper-parameter tuning phase, in terms of the loss function, was selected and retrained on the union of training and validation datasets. The hyper-parameters for each optimized model selected for evaluation, are listed in Table 3.1. The retrained model was evaluated against the test dataset and its corresponding evaluation metric was computed. To account for randomness (in the training process), we repeated the above process, i.e., training the best model on the joint training and validation set and evaluating it on the test set, 30 times and reported descriptive statistics of the evaluation metric. To evaluate the statistical significance of the difference in accuracy metrics of different DL-based safety metric forecasters, we used the Mann-Whitney U test [87]. To measure the effect size of the differences, we measured Vargha and Delaney's $\hat{A}_{AB}$, where $0 \leq \hat{A}_{AB} \leq 1$ [138]. Generally, the value of $\hat{A}_{AB}$ indicates a small, medium, and large difference (effect size) between populations A and B when it is higher than 0.56, 0.64, and 0.71, respectively.

We investigated RQ_1 while considering, as the window size for the hazard forecast horizon, the minimum reaction time required for a human to take over control of the system in case of a hazard. Considering that each time step in the ACT dataset corresponds to one second and the minimum reaction time for a human with vehicles traveling at 30 mi/h is 3 s [131], we set the minimum hazard forecast horizon to 3 timesteps. Furthermore, the lookback to forecast horizon ratio was set to 3 times, since it has been frequently considered in previous studies [74, 118].

Similarly, for the ADS dataset, we selected the hazard forecast horizon of 3 timesteps equaling to 3 s, which is in line with the minimum reaction time suggested by the original study [131]. However, due to the small size of the dataset, as described in section 3.6.1, we could only select the lookback to a forecast horizon ratio of 1, i.e., 3 s. Note that larger lookbacks to forecast horizon ratios significantly increase the total window size and reduce the number of samples available to train and test the model.

Evaluation Metric. As discussed in section 3.5, a probabilistic forecast is better suited than a point forecast for critical applications, such as predicting a safety violation, as it provides forecast intervals with attached probabilities. Similar to previous studies in other application domains, where the performance of probabilistic forecasting models has been reported [74,118], we report the *q-Risk* metric at multiple quantiles. Equation 3.4 provides the definition of q-Risk. Intuitively, *q-Risk* measures the quantile loss [141] (Equation 3.5) across the entire hazard forecast horizon, normalized over the length of the horizon and over all samples in the test set. Thus, it allows us to compare the safety metric prediction accuracy of models under evaluation at each prediction quantile. Formally, q-Risk is defined as follows:

$$q\text{-Risk} = \frac{2\sum_{y_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{max}} QL\big(y_t,\ \hat{y}(q, t-\tau, \tau),\ q\big)}{\sum_{y_t \in \tilde{\Omega}} \sum_{\tau=1}^{\tau_{max}} |y_t|}, \tag{3.4}$$

where $\tilde{\Omega}$ is the test set, q is the quantile, $\tau = 1, \ldots, \tau_{max}$ is the time step counter of the hazard forecast horizon[7] and QL is the quantile loss function, which is defined in Equation 3.5.

$$QL(y, \hat{y}, q) = q(y - \hat{y})_+ + (1 - q)(\hat{y} - y)_+, \tag{3.5}$$

where $(.)_+ = max(0, .)$.

Given the safety-critical nature of the decisions that need to be made based on the predicted safety metric values, we reported the q-risk at quantiles that correspond to tail-end values of the prediction interval, namely 90% (q=0.05, 0.95), 95% (q=0.025, 0.975), 99% (q=0.005, 0.995), as well as the median (q=0.5) of the prediction distribution.[8]

ACT Case Study Results

Table 3.2 reports the achieved q-Risk values for Seq2Seq, DeepAR, MQCNN, and TFT over 30 repetitions at different quantiles, where the best value for each quantile is written in **bold**.

Overall, we observe that TFT consistently outperforms the other models at all reported quantiles, except in the case of the *he* safety requirement when $q < 0.025$, where TFT and DeepAR both have the lowest q-Risk value. Furthermore, for the *cte* safety requirement, DeepAR is the second best at very high and low ends of the quantile spectrum (specifically, when $q < 0.025$ or $q \geq 0.95$), while yielding the worst accuracy at the median of the forecast probability distribution ($q = 0.5$). Similarly, for the *he* safety requirement, DeepAR is the second best at the ends of the quantile spectrum (except when $q < 0.025$, as mentioned above), while being second to last at the median ($q = 0.5$). We suspect that the fact that DeepAR is an iterative forecasting model, as opposed to the other three models which

[7] $\tau = 1$ and $\tau = \tau_{max}$ correspond to $t+1$ and $t+h$ in Equation 3.2, respectively.

[8] Note that the median of the predicted probability distribution often corresponds to the single value predicted by *point* forecasting methods [13].

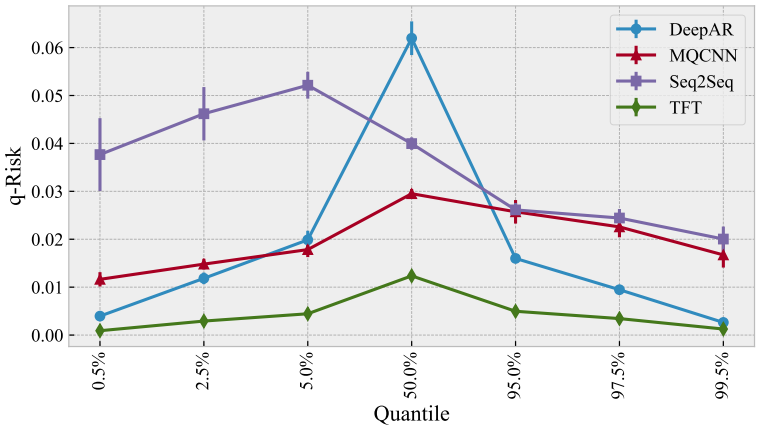
Table 3.2: q-Risk values for different models and quantiles, for the *cte* and *he* safety requirements of the ACT case study and the *cte* safety requirement of the ADS case study, respectively.

Quantile	Seq2Seq	DeepAR	MQCNN	TFT
		ACT_{cte}		
$q = 0.005$	0.038 ± 0.0076	0.004 ± 0.0006	0.012 ± 0.0015	$\mathbf{0.001 \pm 0.0001}$
$q = 0.025$	0.046 ± 0.0056	0.012 ± 0.0012	0.015 ± 0.0011	$\mathbf{0.003 \pm 0.0001}$
$q = 0.05$	0.052 ± 0.0028	0.020 ± 0.0019	0.018 ± 0.0016	$\mathbf{0.004 \pm 0.0002}$
$q = 0.5$	0.040 ± 0.0013	0.062 ± 0.0035	0.030 ± 0.0010	$\mathbf{0.012 \pm 0.0002}$
$q = 0.95$	0.026 ± 0.0014	0.016 ± 0.0008	0.026 ± 0.0025	$\mathbf{0.005 \pm 0.0001}$
$q = 0.975$	0.024 ± 0.0019	0.009 ± 0.0005	0.023 ± 0.0022	$\mathbf{0.003 \pm 0.0001}$
$q = 0.995$	0.020 ± 0.0026	0.003 ± 0.0002	0.017 ± 0.0026	$\mathbf{0.001 \pm 0.0001}$
		ACT_{he}		
$q = 0.005$	0.208 ± 0.0420	$\mathbf{0.041 \pm 0.0022}$	0.092 ± 0.0197	$\mathbf{0.041 \pm 0.0026}$
$q = 0.025$	0.300 ± 0.0363	0.118 ± 0.0040	0.171 ± 0.0161	$\mathbf{0.096 \pm 0.0037}$
$q = 0.05$	0.420 ± 0.0370	0.199 ± 0.0056	0.260 ± 0.0151	$\mathbf{0.140 \pm 0.0041}$
$q = 0.5$	0.841 ± 0.0204	0.732 ± 0.0119	0.705 ± 0.0170	$\mathbf{0.379 \pm 0.0025}$
$q = 0.95$	0.541 ± 0.0390	0.296 ± 0.0092	0.418 ± 0.0372	$\mathbf{0.158 \pm 0.0036}$
$q = 0.975$	0.458 ± 0.0314	0.196 ± 0.0077	0.316 ± 0.0215	$\mathbf{0.112 \pm 0.0034}$
$q = 0.995$	0.330 ± 0.0351	0.086 ± 0.0053	0.180 ± 0.0273	$\mathbf{0.049 \pm 0.0025}$
		ADS_{cte}		
$q = 0.005$	0.019 ± 0.0025	$\mathbf{0.007 \pm 0.0004}$	0.015 ± 0.0021	0.023 ± 0.0040
$q = 0.025$	0.056 ± 0.0064	$\mathbf{0.023 \pm 0.0008}$	0.052 ± 0.0076	0.045 ± 0.0023
$q = 0.05$	0.074 ± 0.0065	$\mathbf{0.042 \pm 0.0011}$	0.070 ± 0.0080	0.069 ± 0.0035
$q = 0.5$	0.222 ± 0.0033	$\mathbf{0.204 \pm 0.0031}$	0.214 ± 0.0031	0.226 ± 0.0066
$q = 0.95$	0.137 ± 0.0073	0.124 ± 0.0049	0.119 ± 0.0068	$\mathbf{0.098 \pm 0.0049}$
$q = 0.975$	0.097 ± 0.0089	0.090 ± 0.0045	0.077 ± 0.0060	$\mathbf{0.065 \pm 0.0039}$
$q = 0.995$	0.045 ± 0.0127	0.048 ± 0.0041	0.028 ± 0.0068	$\mathbf{0.023 \pm 0.0006}$

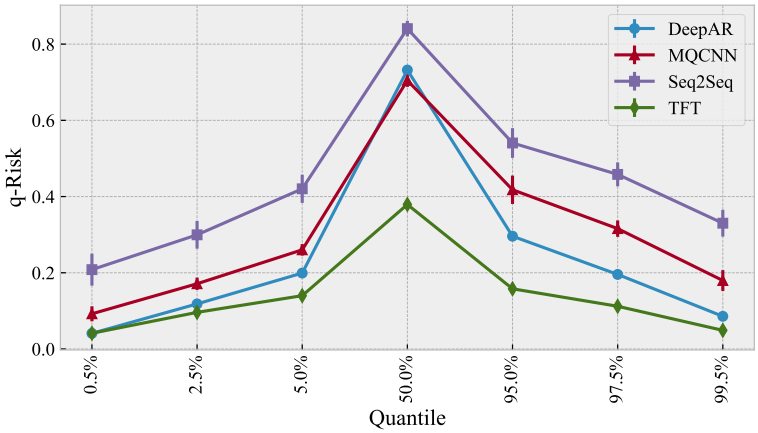
are sequence-to-sequence forecasting models, could explain the large variability in q-Risk values over quantiles. Since iterative forecasting models, such as DeepAR, only predict the target value for the next timestep and use the predicted value to predict the timestep after that (as explained in section 3.2), they are prone to accumulating forecasting errors from previous forecast timesteps. DeepAR's error accumulation is more extreme when predicting at quantiles closer to the median (q = 0.5), where other models also have higher q-Risk values than for other quantiles.

Figure 3.3a, which depicts the q-Risk average values and their 95% confidence interval for different models at all measured quantiles, illustrates the large change in performance (average q-Risk values) of DeepAR.

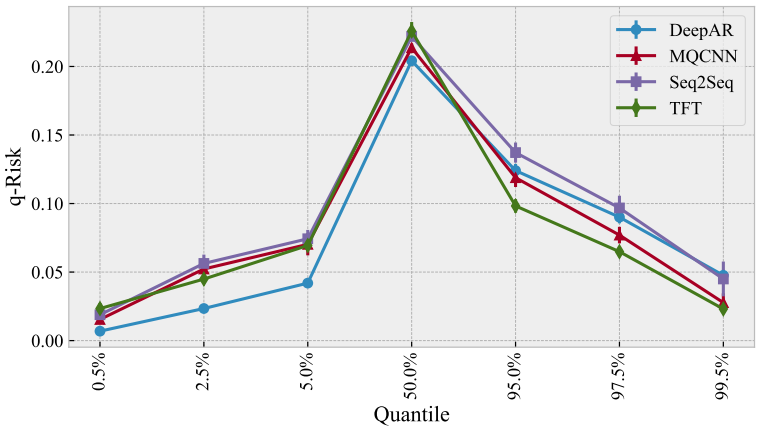
Our visual observations are supported by the statistical comparison results reported in Table 3.3. Columns A and B indicate the DL-forecasting models being compared. Columns p and $\hat{A}_{AB}$ indicate statistical significance and effect size (as described in section 3.6.3), respectively, when comparing A and B in terms of q-Risk at different quantiles q. Given



(a) ACT_{cte}



(b) ACT_{he}



(c) ADS_{cte}

Figure 3.3: Average q-Risk values and their corresponding 95% confidence interval ($\text{CI}_{0.95}$) for all models over all reported quantiles, for the ***cte*** and ***he*** safety requirements of the autonomous taxiing (ACT) case study and the ***cte*** safety requirement of the autonomous driving (ADS) case study, respectively. Note that the x-axis is *not* drawn to scale, in favor of a more readable presentation.

Table 3.3: Statistical comparison of q-Risk values for different DL-based forecasters at different quantiles q.

Comparison		q-Risk											
		$q = 0.005$		$q = 0.025$		$q = 0.05$		$q = 0.5$		$q = 0.95$		$q = 0.975$		$q = 0.995$	
A	B	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$
ACT_{cte}															
Seq2Seq	DeepAR	8.15×10^{-11}	0.99	6.72×10^{-10}	0.96	3.34×10^{-11}	1.00	3.02×10^{-11}	0.00	1.33×10^{-10}	0.98	3.34×10^{-11}	1.00	3.02×10^{-11}	1.00
Seq2Seq	MQCNN	2.00×10^{-5}	0.82	1.56×10^{-8}	0.93	3.02×10^{-11}	1.00	7.39×10^{-11}	0.99	4.04×10^{-1}	0.56	9.33×10^{-2}	0.63	4.51×10^{-2}	0.65
Seq2Seq	TFT	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
DeepAR	MQCNN	2.87×10^{-10}	0.03	1.41×10^{-4}	0.21	7.48×10^{-2}	0.63	3.02×10^{-11}	1.00	3.20×10^{-9}	0.05	3.02×10^{-11}	0.00	3.02×10^{-11}	0.00
DeepAR	TFT	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
MQCNN	TFT	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
ACT_{he}															
Seq2Seq	DeepAR	8.48×10^{-9}	0.93	4.50×10^{-11}	1.00	3.02×10^{-11}	1.00	4.62×10^{-10}	0.97	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
Seq2Seq	MQCNN	2.60×10^{-5}	0.82	1.60×10^{-7}	0.89	1.29×10^{-9}	0.96	9.92×10^{-11}	0.99	4.64×10^{-5}	0.81	1.56×10^{-8}	0.93	1.87×10^{-7}	0.89
Seq2Seq	TFT	8.48×10^{-9}	0.93	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
DeepAR	MQCNN	8.15×10^{-5}	0.20	8.20×10^{-7}	0.13	8.48×10^{-9}	0.07	1.70×10^{-2}	0.68	3.08×10^{-8}	0.08	6.72×10^{-10}	0.04	2.44×10^{-9}	0.05
DeepAR	TFT	8.77×10^{-1}	0.49	4.18×10^{-9}	0.94	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.69×10^{-11}	1.00
MQCNN	TFT	8.66×10^{-5}	0.80	3.16×10^{-10}	0.97	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
ADS_{cte}															
Seq2Seq	DeepAR	4.98×10^{-11}	0.99	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	2.39×10^{-8}	0.92	3.67×10^{-3}	0.72	5.40×10^{-1}	0.55	3.39×10^{-2}	0.34
Seq2Seq	MQCNN	4.36×10^{-2}	0.65	2.12×10^{-1}	0.59	9.05×10^{-2}	0.63	1.25×10^{-4}	0.79	5.56×10^{-4}	0.76	1.17×10^{-3}	0.74	2.50×10^{-3}	0.73
Seq2Seq	TFT	2.06×10^{-1}	0.40	2.07×10^{-2}	0.67	5.79×10^{-1}	0.54	8.53×10^{-1}	0.49	2.44×10^{-9}	0.95	3.96×10^{-8}	0.91	1.58×10^{-1}	0.61
DeepAR	MQCNN	8.99×10^{-11}	0.01	3.02×10^{-11}	0.00	3.02×10^{-11}	0.00	2.01×10^{-4}	0.22	6.35×10^{-2}	0.64	5.56×10^{-4}	0.76	1.75×10^{-5}	0.82
DeepAR	TFT	3.02×10^{-11}	0.00	3.02×10^{-11}	0.00	3.02×10^{-11}	0.00	2.38×10^{-7}	0.11	9.26×10^{-9}	0.93	7.12×10^{-9}	0.94	3.34×10^{-11}	1.00
MQCNN	TFT	1.77×10^{-3}	0.26	6.00×10^{-1}	0.54	1.54×10^{-1}	0.39	1.24×10^{-3}	0.26	2.88×10^{-6}	0.85	1.17×10^{-3}	0.74	7.73×10^{-2}	0.37

a significance level of $\alpha = 0.01$, for the *cte* safety requirement, we observe that the differences between the best model (TFT) in all quantiles, and the other models are significant. Furthermore, for all quantiles, $\hat{A}_{AB}$ is greater than 0.71 when B = TFT, indicating that the difference between TFT and other models is large. For the *he* safety requirement, TFT and DeepAR are equally the best models, when $q < 0.025$. In this case, we observe that TFT is significantly better than the second best model, i.e., MQCNN, with a large difference, though that is not the case for DeepAR.

ADS Case Study Results

The q-Risk values achieved by Seq2Seq, DeepAR, MQCNN, and TFT over 30 repetitions at different quantiles are reported in Table 3.2. Note that the best value for each quantile is written in **bold**.

Overall, we observe that at each quantile, the difference between the most accurate and least accurate models are less than what is observed for the ACT case study results at similar quantiles (compare Figure 3.3c vs Figure 3.3a and Figure 3.3b). We believe that this is due to the sample size of the ADS dataset which is significantly lower than the size of the ACT dataset, as discussed in item 3.6.1, where a model like TFT is expected to suffer the most, as it is a transformer-based model, which has been shown to require significantly more training data than other DL-based models such as CNNs in vision tasks [32]. Nonetheless, we observe that TFT achieves the lowest q-Risk values (most accurate predictions) when $q > 0.5$. Whereas, for $q \leq 0.5$, DeepAR outperforms other models.

Our statistical test results (Table 3.3), confirm that TFT significantly outperforms other models when $0.5 < q < 0.995$, with a large effect size as the corresponding $\hat{A}_{AB}$ values are greater than 0.71. However, at $q = 0.995$, using the Mann-Whitney U-test, when $A \in \{\text{Seq2Seq}, \text{MQCNN}\}$ and $B \in \{\text{TFT}\}$, indicate that their difference is not statistically significant (with a confidence level of 95%), as p-values are larger than 0.05. Therefore, at $q = 0.995$, we conclude that sequence-to-sequence models, i.e., TFT, MQCNN and Seq2Seq, all equally yield the best safety metric prediction accuracy. Finally, we observe that for $q \leq 0.5$, DeepAR consistently outperforms other models with a large effect size.

Summary

Given the results of the ACT case study (section 3.6.3), we observed that, for probabilistic prediction of both *cte* and *he* safety metrics, at all measured quantiles q, with a practical window configuration, i.e., hazard forecast horizon of 3 s (which correlates to the minimum reaction time, as discussed in section 3.6.3) and lookback to forecast horizon ratio of three (which is similar to the ratio used by the literature in multiple forecasting problems [74, 118]), TFT yields significantly more accurate quantile forecasts than Seq2Seq, DeepAR, and MQCNN. Our observations for the *he* safety requirement is the same as *cte*, for $q \geq 0.025$. Whereas, for $q < 0.025$, TFT and DeepAR both yield the highest time series prediction accuracy. Therefore, we can conclude that for the ACT dataset, where the dataset size is large and contains numerous safety violations, TFT is the best model or one

of the best models to be used for probabilistic forecasting of the safety metric values at all quantiles, given a practical window configuration.

For the ADS case study results (section 3.6.3), we observed that for probabilistic safety metric prediction, again with a practical window configuration, i.e., hazard forecast horizon of 3 s and a lookback to forecast horizon ratio of one, as discussed in section 3.6.3, TFT yields significantly more accurate predictions when $0.5 < q < 0.995$. At $q = 0.995$, all sequence-to-sequence models, i.e., TFT, MQCNN and Seq2Seq, yield the lowest q-Risk value. Finally, DeepAR yields significantly more accurate predictions when $q \leq 0.5$. Therefore, for the ADS case study, where the size of the dataset is a fraction of the ACT dataset, as discussed in section 3.6.1, TFT is the best or one of the best models to be used for safety metric forecasting when $q > 0.5$. Though DeepAR is the most accurate model when $q \leq 0.5$, as we will discuss in detail in section 3.6.4, given the definition of the safety metric (Equation 3.1) and the safety violation function (Equation 3.3), forecasts for quantiles $q > 0.5$ are more important for safety violation prediction.

For a the ACT case study, where the size of the dataset is large, given a practical window configuration, i.e., hazard forecast horizon of 3 s and lookback to forecast horizon ratio of 3, TFT is more suitable than Seq2Seq, DeepAR and MQCNN, for probabilistic safety metric forecasting over all reported quantiles, for both *cte* and *he* safety requirements.
For the ADS case study, where the size of the dataset is small, given practical window configuration of $h = 3$ s and $cm = 1$, DeepAR is significantly more accurate than TFT, MQCNN, and Seq2Seq, for $q \leq 0.5$. However, TFT is more the most accurate model, among the evaluated models, when $q > 0.5$, which are more important in a safety monitoring context, as previously discussed.
Therefore, TFT is the most accurate model, when $q > 0.5$, for both case studies.

3.6.4 RQ_2: Safety Violation Prediction Accuracy

In this section, similar to subsection 3.6.3, first we provide the details of our evaluation methodology to answer RQ_2. Then, we present the results for the ACT and ADS case studies. Finally, we draw conclusions from the results of both case studies and present our answer to RQ_2.

Methodology

To answer RQ_2, we reuse the models trained to answer RQ_1 with the aim of predicting safety violations. To do so, we applied the safety violation function (Equation 3.3) to the safety metric values predicted by the forecasting models, as reported in RQ_1. A non-negative safety function value implies a safety violation. We compared the predicted safety violations with the true safety violation value of the test samples used in RQ_1. True safety violation values are calculated by applying the safety violation function to the true safety metric values of the test samples.

Evaluation Metric. To report the safety violation prediction accuracy of the models, we report *Precision* (Pr=TP/(TP+FP)) and *Recall* (Re=TP/(TP+FN)), where *true positives* (TP), *false positive* (FP), and *false negatives* (FN) are the correct, false, and missed safety violation predictions, respectively. Note that Precision measures the fraction of correct warnings that a safety monitor raises, whereas Recall measures the fraction of safety violations that a safety monitor can successfully predict [131]. We further compute and report the F_β score [10] as a weighted balance between precision and recall, with $\beta = 3.0$ ($F_3 = \frac{10 \cdot Precision \times Recall}{9 \cdot Precision + Recall}$), granting higher importance to Recall as compared to Precision, as false negatives have severe consequences for safety-critical systems [41, 131]. We recall that false negatives, in the context of safety-critical systems, are safety violations that were not predicted by the safety monitor and thus could lead to system hazards. In contrast, false positives, although an important consideration for the design of the safety monitor, lead to inconvenience or inefficiencies for the users, which are less harmful than safety violations. For the ACT system specifically, false positives could lead to the disengagement of the autonomous taxiing operation or emergency stops, which could lead to delayed flight operations and schedules.

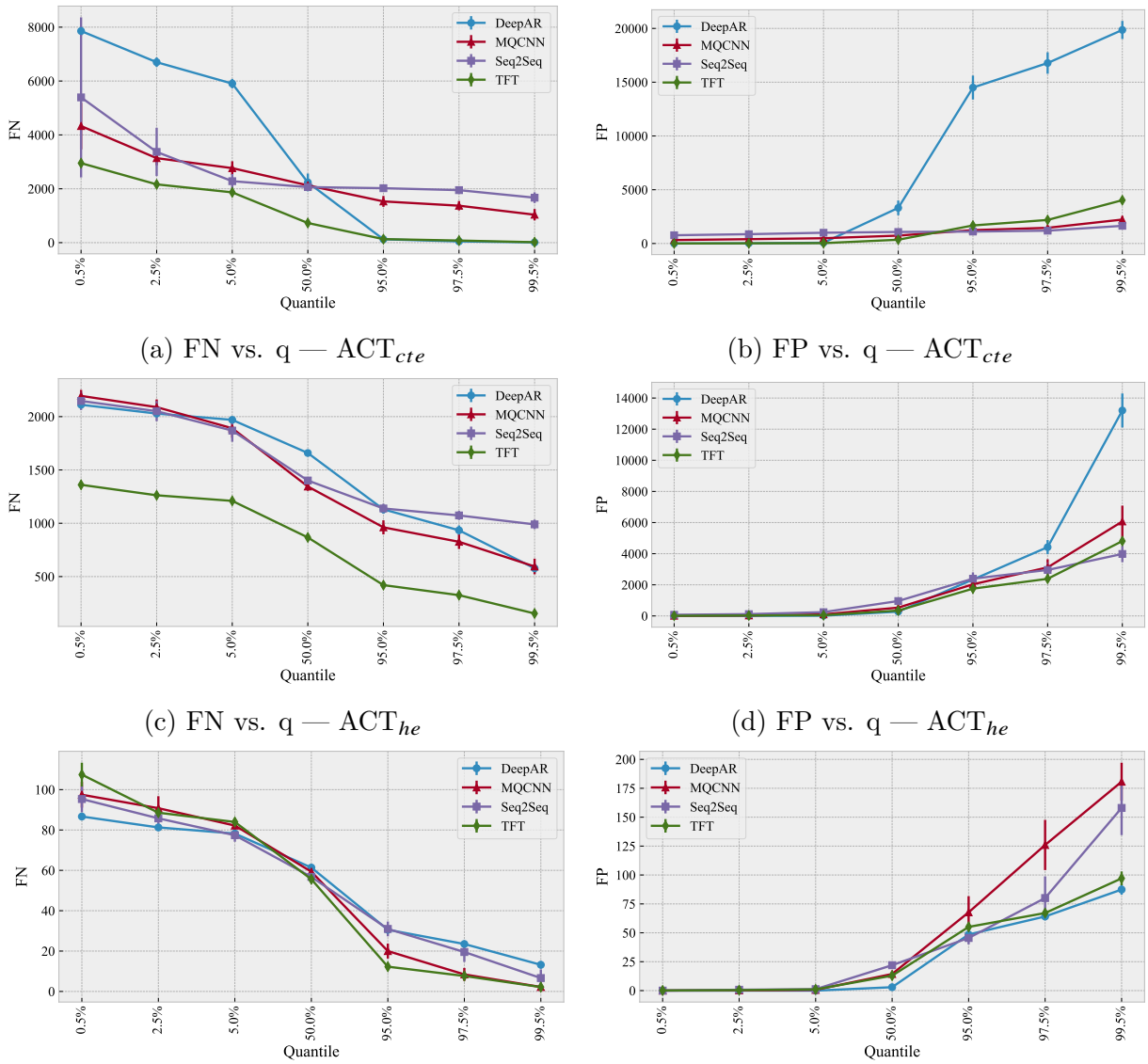
ACT Case Study Results

Figure 3.4a and Figure 3.4b illustrate the False Negative (FN) and False Positive (FP) values for all the models at the reported quantiles, respectively.[9] Overall, we observe that with an increase in the prediction quantile, the FN value decreases while the FP value increases. This is expected based on the definitions of the safety metric and the safety violation function (Equation 3.1[10] and Equation 3.3, respectively). Recall that negative values of the safety metric imply no violation of the safety requirement while non-negative safety metric values imply a violation. Further, the higher the negative values, the closer the system is to a safety violation. Recall that safety metric values predicted at higher prediction quantiles (q) provide the upper bounds of the predicted safety metric value, which are worst-case estimates based on the definition of the safety metric function. Therefore, the higher the prediction quantile, the higher the probability of the safety monitor correctly predicting safety violations (lower FN) and raising false alarms (higher FP).

As discussed in section 3.6.4, the priority in safety-critical applications is having the lowest FN value possible (since FNs lead to system hazards), while having a reasonably low FP value (since FPs lead to inefficiencies) is the second priority. Therefore, for the ACT problem targeted in this chapter, hereafter we focus on the results for prediction quantiles $q \geq 0.5$, for which the FN and FP values are reported in Table 3.4. We have provided the values at other quantiles in the supporting material (subsection 3.6.9). Table 3.4 reports the averages and their 95% confidence intervals for the evaluation metrics, namely Pr, Re, F_3.

[9] Note that the x-axis, i.e., quantile q, is not drawn up to scale for better readability.

[10] As mentioned in section 3.6.1, substituting *cte* with *he* in Equation 3.3, provides us with the safety metric function definition for the *he* safety requirement.



(a) FN vs. q — ACT_{cte}

(b) FP vs. q — ACT_{cte}

(c) FN vs. q — ACT_{he}

(d) FP vs. q — ACT_{he}

(e) FN vs. q — ADS_{cte}

(f) FP vs. q — ADS_{cte}

Figure 3.4: False Negatives (FN) and False Positives (FP) at different prediction quantiles (q), for ***cte*** and ***he*** safety requirements of the ACT case study and the ***cte*** safety requirement of the ADS case study, respectively. Note that the x-axis is *not* drawn to scale, in favor of a more readable presentation.

Overall, Precision for all models drops as q increases, whereas Recall increases, for both *cte* and *he* safety requirements. This general trend is in line with the FN and FP trends above. Since estimates become more conservative, more safety violations are correctly predicted (Recall ↑) while the proportion of false alarms increases (Precision ↓). For the *he* safety requirement, since the proportion of time steps including a safety violation is lower than that of the *cte* safety requirement (item 3.6.1), we expect and observe that the precision scores of the models recorded for *he* are lower than the scores recorded for *cte*. In the case of the *cte* safety requirement, note that TFT and DeepAR reach a Recall value of 1.0 at a q = 0.995, thus indicating all of the safety violations are correctly predicted, which is also confirmed by the low corresponding FN values; Seq2Seq and MQCNN, on the other hand, yield the lowest Recall scores (high FN values). However, we observe that DeepAR has the lowest Precision among the models for q ≥ 0.5, indicating a higher fraction of false alarms, which is also confirmed by the fact that the FP value for DeepAR is an order of magnitude greater than that of other models. In contrast, TFT has a precision above 0.92, for q ≥ 0.5, which makes it a more suitable choice for safety monitoring than DeepAR. In the case of the *he* safety requirement, we observe that TFT consistently yields the highest Precision and Recall scores for q ≥ 0.5, whereas the Recall of other models is 20 − 40% lower. The superiority of TFT over other models is further confirmed by the reported F_3 scores, where TFT consistently has the highest F_3 scores for q ≥ 0.5, for both the *cte* and *he* safety requirements. Note that the highest F_3 scores for each quantile are highlighted in bold. Our visual observations are supported by the statistical comparison results reported in Table 3.6. Columns *A* and *B* indicate the DL-forecasting models being compared. Given a significance level of $\alpha = 0.01$, we observe that the differences between TFT and the other models are significant for all quantiles. Furthermore, for all quantiles, $\hat{A}_{AB}$ is greater than 0.71 when *B* = TFT, indicating that the difference between TFT and other models is large.

ADS Case Study Results

Similar to the ACT case study results (section 3.6.4), Figure 3.4e and Figure 3.4f, indicate that FN decreases and FP increases with increase in prediction quantile. As explained in section 3.6.4, we will focus on the results for prediction quantiles $q \geq 0.5$. We report the averages and 95% confidence intervals of the FN, FP, Precision, Recall and F_3 score values in Table 3.5.

Overall, in line FP and FN trends, Precision for all models drops when q increases, while Recall increases, as discussed in section 3.6.4. Note that TFT and MQCNN yield the highest Recall score of 0.985 at $q = 0.995$, while for the same quantile, MQCNN yields the lowest Precision score. We further observe for $q \geq 0.5$, TFT is the second best performing model in terms of Precision score while yielding the highest Recall score. This results in TFT achieving the highest F_3 score over all quantiles $q \geq 0.5$. Moreover, we observe that DeepAR yields the lowest Recall and F_3 score over all quantiles $q \geq 0.5$.

This case study thus confirms the superiority of TFT over Seq2Seq, MQCNN, and DeepAR, when $q > 0.5$, in terms of F_3 score, based on the results of our statistical analysis, reported in Table 3.6. We further observe that, when $q > 0.5$, TFT is significantly better than other models with a high effect size. However, for $q = 0.5$, we observe that TFT

is not significantly better than Seq2Seq, though it is still outperforming DeepAR and MQCNN significantly. Moreover, the effect size of the difference between the F_3 scores of TFT and MQCNN when $q = 0.5$, i.e., $\hat{A}_{AB}$, when A = TFT and B = MQCNN, is $0.64 < \hat{A}_{AB} = 0.69 < 0.71$, indicating a medium effect size.

Summary

Based on the results of the ACT case study (section 3.6.4), where the dataset is large and includes numerous safety violations, we conclude that, for probabilistic prediction of the safety violation, at all measured quantiles q, with again a hazard forecast horizon of 3 s and a lookback to forecast horizon ratio of 3, as discussed in section 3.6.3, TFT is significantly more accurate than Seq2Seq, DeepAR, and MQCNN, for both *cte* and *he* safety requirements.

Based on the ADS case study results (section 3.6.4), where the size of the dataset is much smaller than the ACT dataset, we observe that, given the hazard forecast horizon of 3 s (equal to the minimum reaction time) and a lookback to forecast horizon ratio of 1, as discussed in section 3.6.3, TFT is significantly more accurate that Seq2Seq, DeepAR, and MQCNN with a large effect size, when $q > 0.5$. At $q = 0.5$, TFT and Seq2Seq both yield the most accurate predictions, while significantly outperforming DeepAR and MQCNN with a large and medium effect size, respectively. Therefore, we conclude that TFT is the most suitable model, among all evaluated models, for safety metric forecasting, for all reported quantiles $q \geq 0.5$. Predictions for $q \leq 0.5$ are in any case not sufficiently accurate regardless of the model employed and therefore comparisons for such q values are not of practical utility.

For the ACT case study, where the size of the dataset is large, given a hazard forecast horizon of 3 s and a lookback to a forecast horizon ratio of 3 for safety violation prediction (minimum reaction time), TFT is significantly more accurate, with a large effect size, than Seq2Seq, DeepAR, and MQCNN for all quantiles q ≥ 0.5, for both the *cte* and *he* safety requirements.
For the ADS case study, where the datset size is much smaller, given a practical window configuration, i.e., $h = 3$ s and $cm = 1$, TFT is the most suitable model, among all evaluated models, for probabilistic safety metric forecasting, for all quantiles $q \geq 0.5$. Predictions for $q \leq 0.5$ are poor for all models.
Therefore, when $q > 0.5$, TFT is consistently the best model for both case studies.

3.6.5 RQ_3: Prediction Accuracy Sensitivity Analysis

In section 3.6.5, we provide the details of our evaluation methodology as it relates to answering RQ_3. As discussed in section 3.6.3, due to the low number of samples available in the dataset of the ADS case study, we were only able to study the effect of varying window sizes on prediction accuracy for the two safety requirements of the ACT case

study, for which we report the results. Finally, we present our answer to RQ_3, based on the reported results.

Methodology

We have answered RQ_1 and RQ_2 based on the minimum reaction time (3 s) for hazard forecast horizon and a commonly used lookback horizon that is three times longer (9 s). However, the size of the hazard forecast and lookback horizons are design choices of the system developers. To investigate the impact of window configuration, for each forecasting model, in terms of safety metric and violation accuracy, inference latency, and computations resource usage, we also assessed the tuned models with different hazard forecast horizons and lookback-to-forecast horizon values.

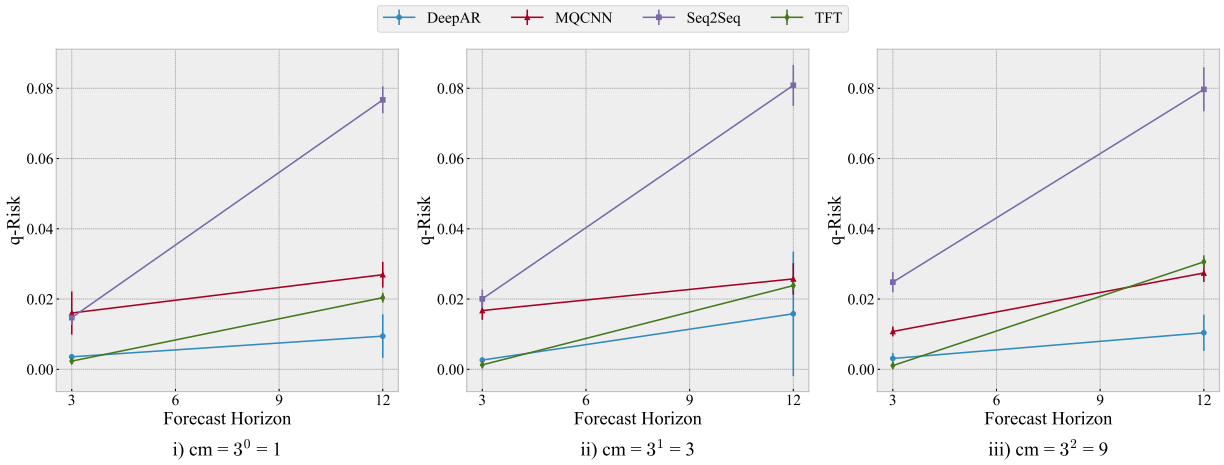
Particularly, we selected the hazard forecast horizon from {3, 12} and the ratio of lookback to forecast horizon window size, also known as *context multiplier* (cm), from {1, 3, 9}. Thus, we studied the following forecast horizon window size and context multiplier combinations (h, cm): $(3, 1)$, $(3, 3)$, $(3, 9)$, $(12, 1)$, $(12, 3)$, and $(12, 9)$. Note that the smallest total window size[11] is $3 + 1 \times 3 = 6$ s, whereas the largest total window size is $12+9\times12 = 120$ s. In our preliminary experiments, we observed that increasing the forecast horizon increases the likelihood that samples include safety violations. We further observed that for forecast horizons longer than 12 s, the distribution of test samples becomes highly imbalanced, leading to biased comparisons of safety violation prediction accuracy metrics between short ($h = 3$ s) and very long ($h > 12$ s) forecast horizons. Furthermore, we would not have been able to study the effect of varying cm on very large forecast horizons since their total window size on higher cm values would become longer than the maximum training sample length. Thus, for the evaluation of RQ_3, we did not include forecast horizons larger than 12 s. The results for a representative instance of our preliminary experiments with long forecast horizons, i.e., a forecast horizon of 36 s and context multiplier of 1, are included in our supporting material (see subsection 3.6.9).

Results

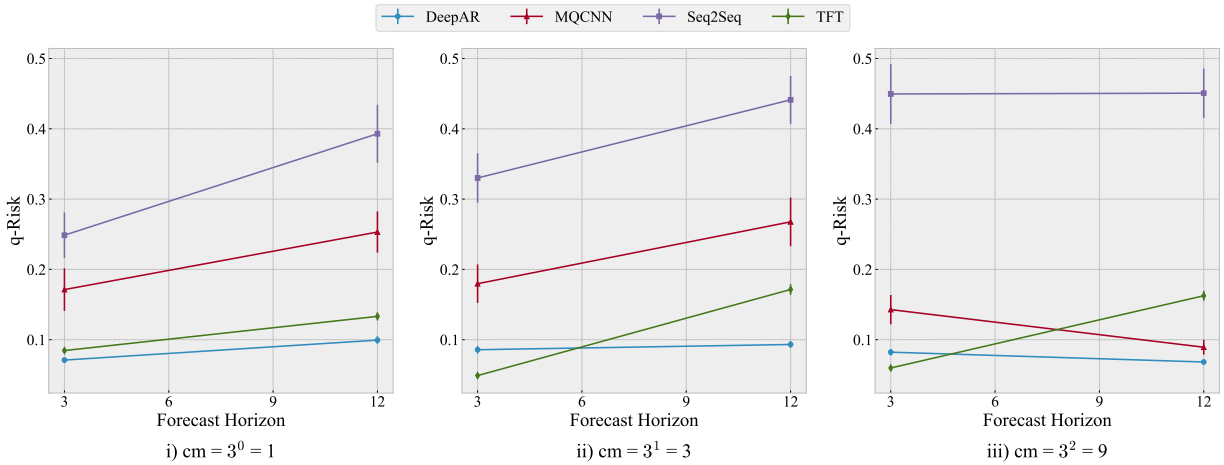
As discussed in section 3.6.4, we are interested in the predictions at quantiles that primarily detect as many hazards as possible while having a low number of false alarms. Due to the safety-critical nature of our problem, we only focus here on predictions at q = 0.995 since it is the most conservative measured prediction quantile (section 3.6.4). Recall that when comparing models, a lower q-Risk value implies a more accurate model at predicting safety metric values.

From Figure 3.5a, we observe that for the cte safety requirement, given a fixed cm value, increasing the forecast horizon h leads to higher q-Risk values and thus lower accuracy. Further, we can also see that, in contrast, for a fixed forecast horizon, increasing cm does not significantly improve q-Risk. We observe that TFT outperforms all models (i.e., has

[11] Total window size = $h + (cm \times h)$



(a) ACT_{cte}



(b) ACT_{he}

Figure 3.5: Safety metric prediction accuracy metrics for various window configurations of DeepAR, MQCNN, Seq2Seq and TFT, for the *cte* and *he* safety requirements, respectively.

the lowest q-Risk value) at the forecast horizon of 3, across all *cm* values. At longer forecast horizons ($h = 12$), DeepAR outperforms all other models. Moreover, note that due to the iterative forecasting architecture of DeepAR, it is prone to accumulating forecasting errors. Thus, its confidence interval increases significantly with the increase in forecast horizon. For instance, at $cm = 3$, the confidence interval of DeepAR includes the q-Risk values for both TFT and MQCNN.

For the *he* safety requirement, similar to *cte*, we observe that given a fixed *cm* value, increasing h leads to higher q-Risk values, except for MQCNN at $cm = 9$, where q-Risk decreases when increasing the forecast horizon.

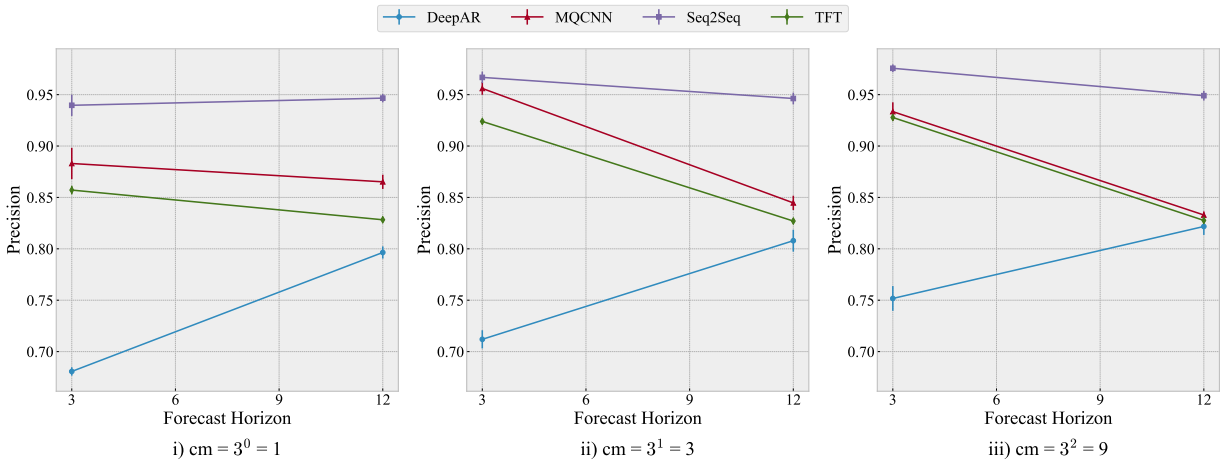
In terms of precision for the *cte* safety requirement (Figure 3.6a), we observe that when increasing forecast horizon, given a constant *cm* value, the precision of models with a sequence-to-sequence architecture (TFT, MQCNN, and Seq2Seq) drops while DeepAR's precision increases, most particularly at $cm = 1$. A similar trend is observed for the

he safety requirement (Figure 3.6b) with the difference that the models reach a lower precision score than similar models evaluated on the *cte* safety requirement data. This can be explained by the lower proportion of samples including safety violations for *he*, when compared to *cte* (as discussed in section 3.6.3). Further note that, for both *cte* and *he*, given a constant forecast horizon, increasing *cm* can slightly improve the precision of the model, most particularly for DeepAR.

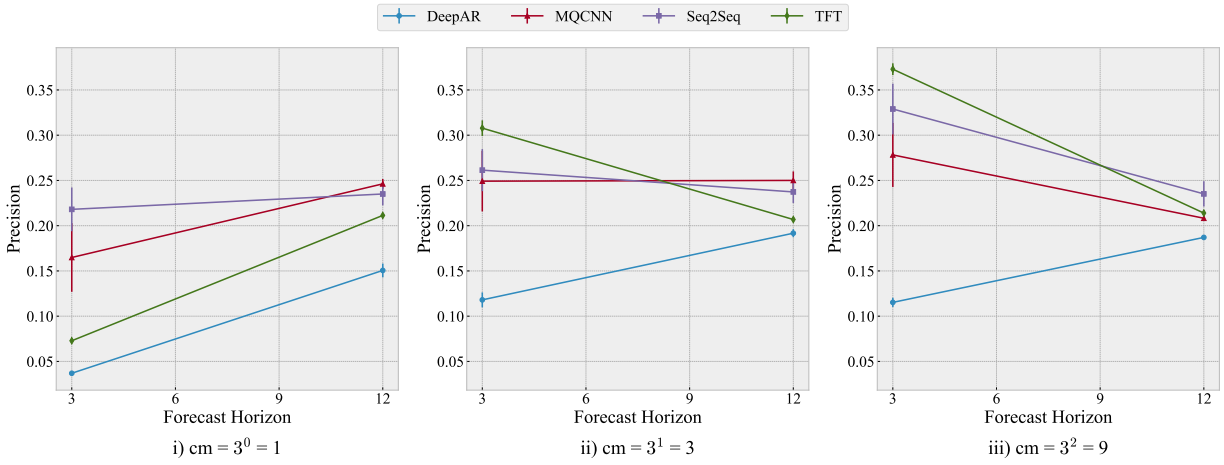
In terms of recall for the *cte* safety requirement (Figure 3.7a), we observe that all models yield a lower recall when increasing forecast horizon while increasing *cm* does not significantly improve recall given the same forecast horizon. Note that TFT and DeepAR similarly reach the highest recall at $h = 3$. Whereas, at $h = 12$, TFT experiences a larger drop in recall and is outperformed by DeepAR. Concerning the effect of *cm*, we observe that MQCNN experiences the most improvement when *cm* changes from 3 to 9. At the same time, we observe that the recall score of DeepAR and Seq2Seq drops when *cm* increases given a fixed forecast horizon. Finally, the recall score of TFT is not significantly affected by changes in *cm* values. Similarly, for the *he* safety requirement, TFT reaches the highest recall score at $h = 3$. However, instead of all models experiencing a drop in recall score with increases in forecast horizon, the recall score for DeepAR increases in all *cm*. MQCNN also experiences a recall score increase with an increase in h at $cm = 9$. To conclude, the effect of *cm* is similar to that observed for the *cte* safety requirement, with the exception of the differences mentioned above.

In terms of overall safety violation prediction accuracy, i.e., F_3 (Figure 3.8), for both *cte* and *he* safety requirements, we observe that TFT yields the highest F_3 score when $h = 3$ for all values of *cm*, except for *he* when $cm = 1$. However, we observe that F_3 is much lower than the scores reached for other *cm* values, suggesting that the small lookback horizon does not contain sufficient information for the models to generate accurate forecasts. Note that for the same window configuration, DeepAR yields the lowest F_3 score, which is mainly due to its very low precision. Nevertheless, given the increase in precision and a smaller decrease in recall for *cte*, as well as an increase for *he*), the F_3 score of DeepAR for configurations with $h = 12$ at any *cm* value rises to become the best model. However, note that the highest F_3 score at $h = 12$ is still lower than the highest F_3 score reached by TFT at $h = 3$, suggesting that increasing the forecast horizon h leads to lower safety violation prediction accuracy in general.

Given a hazard forecast horizon of 3 s, for both *cte* and *he* safety requirements of the ACT case study, TFT yields the most accurate safety metric and safety violation predictions, with an improving accuracy when the lookback horizon length increases ($cm \times h$). For the *he* safety requirement at $cm = 1$, the lookback horizon does not contain sufficient information for any model to generate accurate forecasts. We further conclude that for prediction horizons longer than the minimum reaction time, i.e., $h = 12$ s), regardless of the *cm* value, DeepAR yields the most accurate predictions.

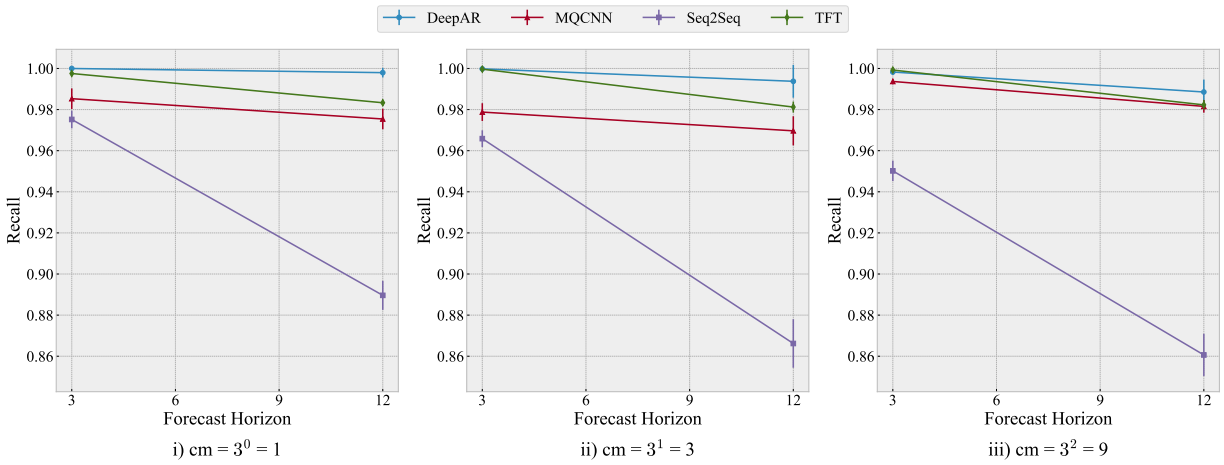


(a) Precision — ACT_{cte}

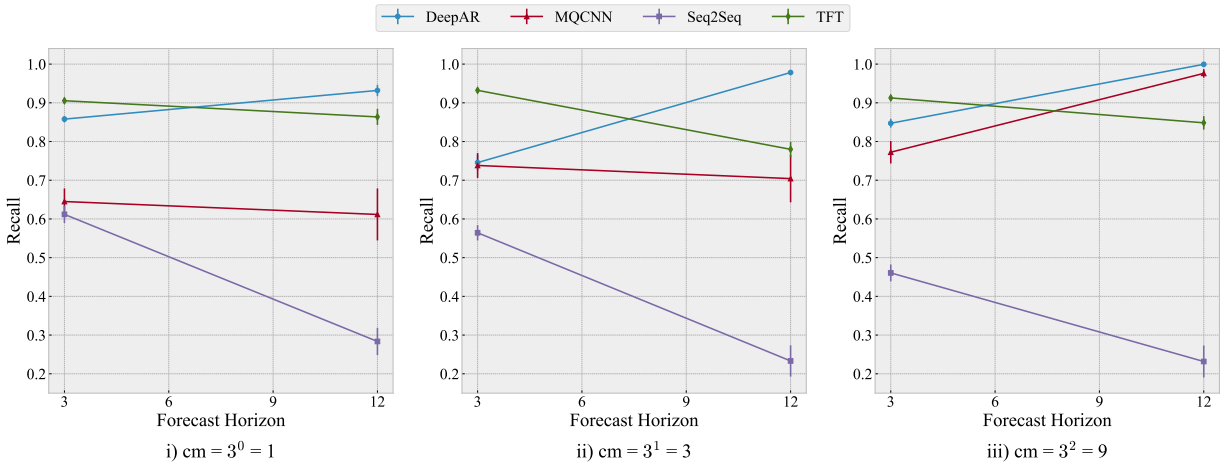


(b) Precision — ACT_{he}

Figure 3.6: Precision score measurements for DeepAR, MQCNN, Seq2Seq, and TFT, in various window configurations, for the ***cte*** (a) and ***he*** (b) safety requirements.

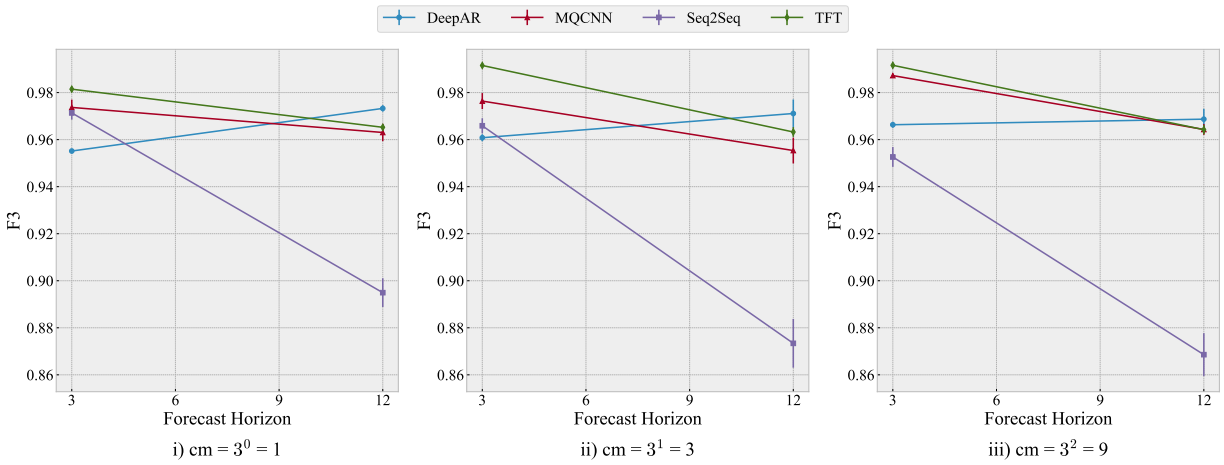


(a) Recall — ACT_{cte}

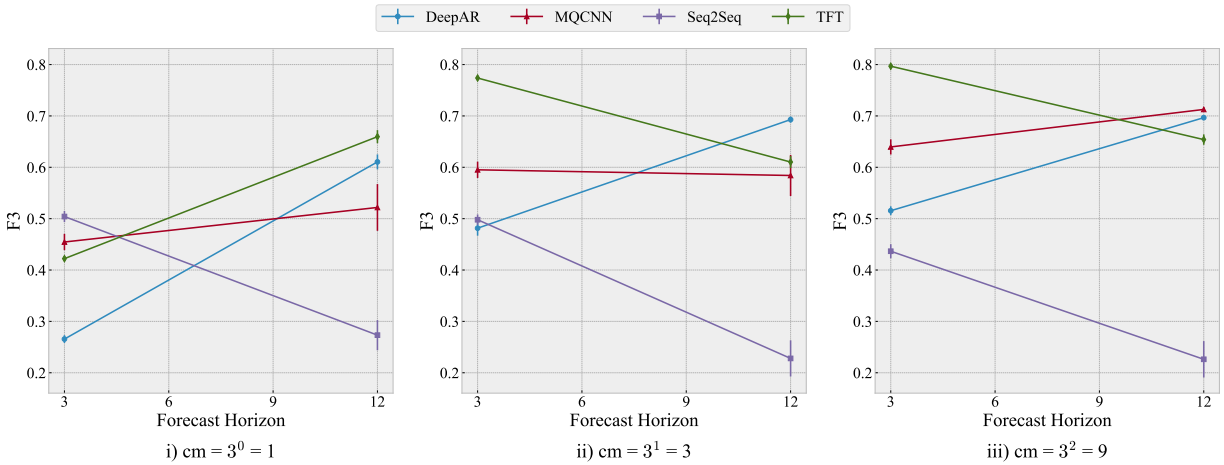


(b) Recall — ACT_{he}

Figure 3.7: Recall score measurements for DeepAR, MQCNN, Seq2Seq, and TFT, in various window configurations, for the ***cte*** (a) and ***he*** (b) safety requirements.



(a) F_3 score — ACT_{cte}



(b) F_3 score — ACT_{he}

Figure 3.8: F_3 score measurements for DeepAR, MQCNN, Seq2Seq, and TFT, in various window configurations, for the ***cte*** (a) and ***he*** (b) safety requirements.

3.6.6 RQ_4: Latency and Memory Overhead

In this section, we provide the details of our evaluation methodology for answering RQ_4. As discussed in the beginning of subsection 3.6.5, we were only able to evaluate the effect of varying window sizes, on runtime performance, for the two safety requirements of the ACT case study. We also measured the runtime performance of the ADS case study, when $h = 3\,\text{s}$ and $cm = 1$. We finally provide our answer to RQ_4 based on the reported results.

Methodology

To answer RQ_4, we queried each of the models trained in RQ_3 over the whole test set for both *cte* and *he* safety requirements of the ACT case study, and collected the average latency (in milliseconds *ms*) and peak GPU memory[12] usage during inference (in megabytes, *MB*). Furthermore, we measure the model size in terms of GPU memory usage (in megabytes, *MB*), by measuring the difference in GPU memory usage before and after a model is loaded to the GPU. Note that the base ML libraries are preloaded in the GPU and their GPU memory usage is not reported. For the ADS case study, we applied the above process to the models that were trained in RQ_1 ($h = 3\,\text{s}$ and $cm = 1$).

Results

The RQ_4 evaluation results for the *cte* safety requirement of the ACT case study are similar to those for its *he* safety requirement. This is to be expected, as the size of the input and output vectors for the same model do not change between the two safety requirements. The only difference between them is due to differences in the hyperparameter values selected to best address each safety requirement, which in turn can change the number of learnable parameters in the model. However, as observed, such change has not substantially changed the results. Therefore, although we present the results for the *cte* safety requirement of the ACT case study (Figure 3.9) in the chapter, our discussion of the results and conclusions similarly hold for the *he* safety requirement. Moreover, we have observed that the ADS case study results, are not substantially different from the ACT case study results for the same window configuration, i.e., when $h = 3\,\text{s}$ and $cm = 1$. Thus, the discussion of the results and conclusions for the *cte* safety requirement of the ACT case study, given a similar window configuration, i.e., when $h = 3\,\text{s}$ and $cm = 1$, similarly holds for the ADS case study. The figures and results for the ADS case study, and the *he* safety requirement of the ACT case study, are available in our replication package (subsection 3.6.9).

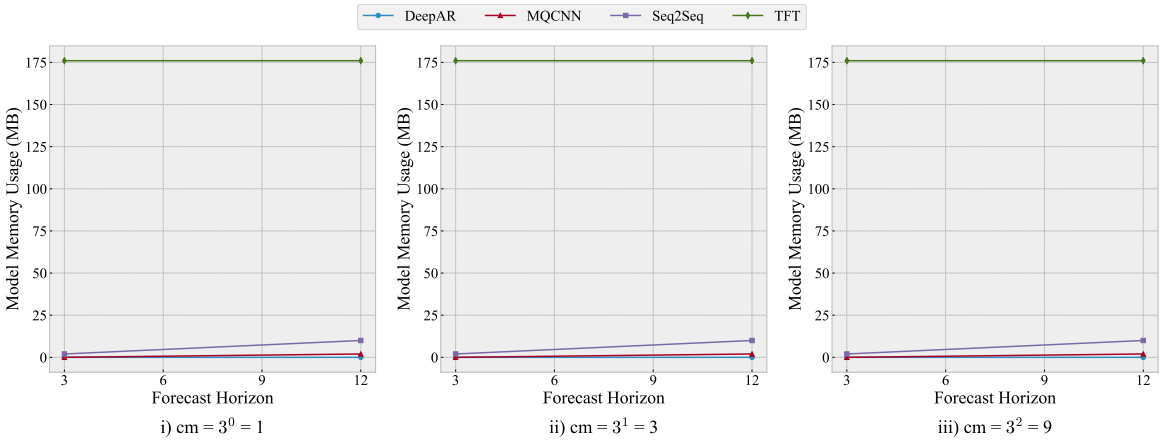
As shown in Figure 3.9a, all models in all window configurations have low GPU memory usage. TFT, which has the highest usage of all, only consumes around 175 MB. We further observe that an increase in context multiplier does not lead to a significant increase in model memory usage, while an increase in forecast horizon leads to slight increases in MQCNN and Seq2Seq model memory usage.

[12]Recall that, as mentioned in Table 3.6.2, we use a GPU to train the models and generate predictions.

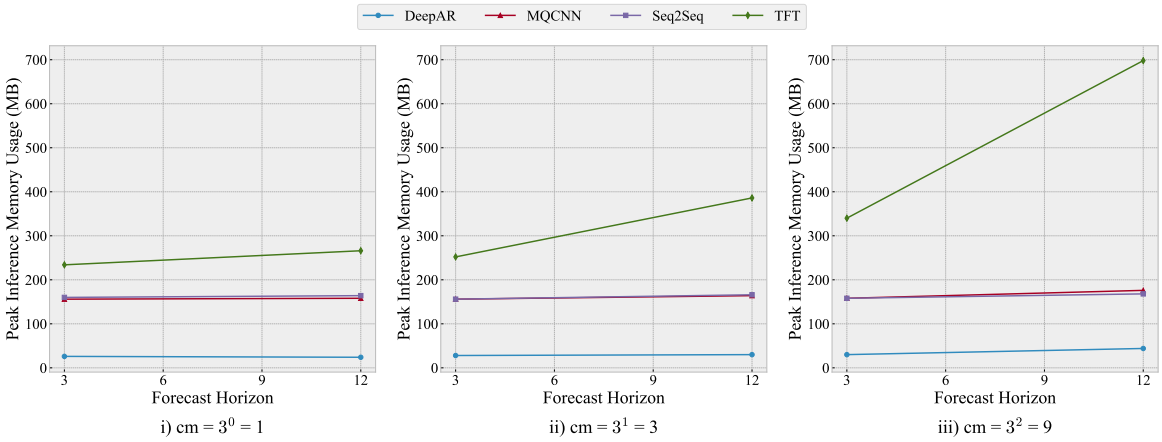
From Figure 3.9b, we observe that an increase in forecast horizon leads to increased peak memory usage during inference. When increasing cm, the peak memory usage of the models does not increase significantly for the same forecast horizon, except for TFT where the rate of increase in peak memory usage with increasing forecast horizon grows at higher cm values. Nevertheless, the largest peak memory usage during inference, which we observe for TFT when $h = 12\,\text{s}$ and $cm = 9$, is 700 MB, only consuming 17.5% of the available memory of an NVIDIA Jetson Nano GPU, the *least* powerful embedded GPU made by NVIDIA. Therefore, we conclude that all the evaluated models, for all h and cm combinations, yield practical GPU memory usage in terms of model size and peak inference memory usage.

Finally, Figure 3.9c suggests that the average inference latency for sequence-to-sequence forecasting models (MQCNN, Seq2Seq, and TFT) does not significantly change when increasing the forecast horizon or context multiplier. However, as expected, DeepAR's inference latency linearly increases with forecast horizon. Furthermore, when increasing cm, DeepAR's prediction latency increases for the same forecast horizon. Thus, for longer prediction horizons or higher context multipliers, DeepAR is prone to having a high inference latency which could render its use prohibitive in the context of safety-critical systems. In general, for a safety monitor to be effective it should be able to predict safety violations and raise an alarm before the planning or control modules update their command. In our ACT example, the system does not include a planner and the controller directly generates control commands based on the perception system outputs. Thus, in our ACT case study, the use of a safety monitor is only meaningful if its average inference latency is less than the controller cycle time, i.e., the time period between two generated control commands. For practical reference, the design requirement for the maximum cycle time of planning and control modules at the Indy Autonomous Challenge [53], is 10 ms [65].[13] In autonomous aviation, Paredes-Vallés et al. [102] and Navardi et al. [94] report an average vision cycle latency of 12 ms and 13 ms, respectively on an NVIDIA Jetson GPU. In another example, for a real-time vision-based drone system developed by Farrukh and West [34], the authors empirically measure and report that the average latency for the vision pipeline, where the safety monitor should have an average 16 ms or less to be useful. Note that DeepAR with a forecast horizon of 12, reaches the maximum latency of 10 ms at $cm = 3$ and largely exceeds it at $cm = 9$. In contrast, sequence-to-sequence models maintain a constant average inference latency of approximately 2 ms for all h and cm values. This is expected since sequence-to-sequence models generate their forecast for all forecast horizon timesteps at once.

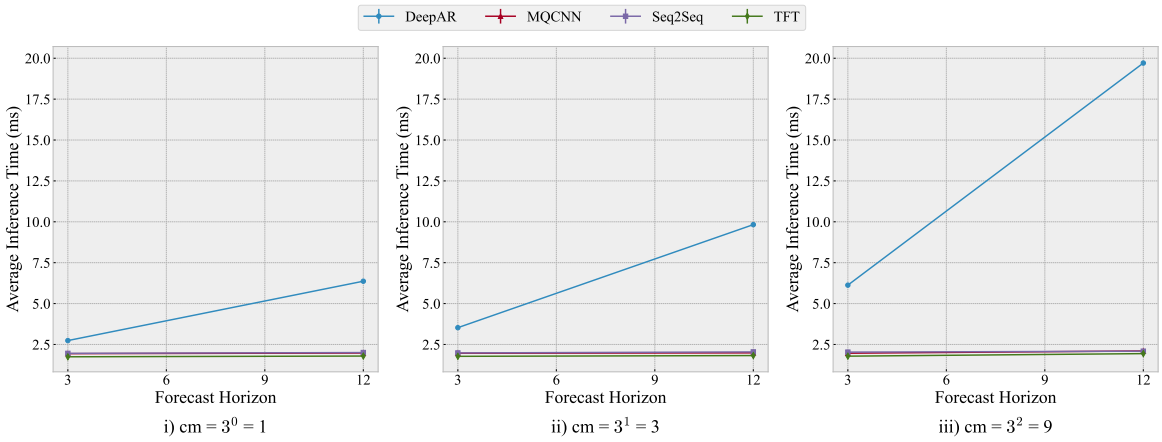
[13]The Indy Autonomous Challenge is an international challenge where teams from universities develop autonomous racing vehicles with the ultimate goal of improving the safety and performance of autonomous driving technology [53].



(a) Model Memory Usage (MB)



(b) Peak Inference Memory Usage (MB)



(c) Average Inference Latency (ms)

Figure 3.9: The plot for a) model memory usage, b) peak inference memory usage, and c) average inference latency for DeepAR, MQCNN, Seq2Seq, and TFT models at different window configurations, for the *cte* safety requirement of the ACT case study, which is similar to the results for the *he* safety requirement. The above results at forecast horizon of 3 s and $cm = 1$ are also similar to the results for the ADS case study.

For both *cte* and *he* safety requirements in the ACT case study, all models in all configurations yield practical model size and peak inference memory usage. Furthermore, although MQCNN, Seq2Seq, and TFT exhibit a constant and very low inference latency, DeepAR's inference latency significantly increases with the forecast horizon and context multiplier, which can render DeepAR impractical for longer forecast horizons at $cm > 1$.
For the ADS case study, all models, in a practical window configuration of $h = 3\,\text{s}$ and $cm = 1$, yield practical model size, peak inference memory usage and inference latency.

3.6.7 Discussion

Safety Monitoring via Safety Metric Forecasting. Overall, the results of our study suggest that safety metric forecasting, given learned component outputs and scenarios, is effective for safety monitoring. Indeed, the models, when evaluated on a dataset with a balanced distribution of safety violations, i.e., the *cte* safety requirement of the ACT case study, have yielded F_3 scores above 95% for all *cm* and *h* combinations, except for Seq2Seq at $h = 12\,\text{s}$.[14] However, the fact that all the evaluated models have high F_3 scores, for the *cte* safety requirement in the ACT case study, does *not* mean that all of them are equally accurate in predicting safety violations. This is highlighted in Table 3.4, where 1-2% differences in F_3 scores, for the *cte* safety requirement, translate into two orders of magnitude increase in the number of false negatives (FN). Furthermore, for both the *he* safety requirement of the ACT case study, where the distribution of safety violations in the dataset contains less safety violations compared to *cte*, and the ADS case study, where the size of the dataset is substantially smaller than that of the ACT case study, we observe that at least one model yields an F_3 score of 77% or more, for $q > 0.5$. This suggests that training the DL-forecasting models to achieve high safety violation prediction accuracy, collecting a large number of diverse safety violations in the dataset is crucial.

Moreover, we observe that the ranking of evaluated models in terms of safety metric accuracy (q-Risk) matches the ranking of models in terms of recall scores, suggesting that q-Risk values could be indicators of the recall scores for safety violation prediction (compare Figure 3.5 with Figure 3.7). However, we observe that the same order does not hold for precision and F_3 scores. For instance, MQCNN and Seq2Seq have higher (q-Risk) values than DeepAR for $h = 3\,\text{s}$ (Figure 3.5), whereas their precision and F_3 scores (Figure 3.6 and Figure 3.8, respectively) are significantly higher. Thus, we conclude that although q-Risk values, which are readily available after training the models and testing them on the test dataset, might be an indicator of the recall score for safety violation prediction, they are not indicators of precision and overall accuracy (F_3) scores for safety violation prediction. Thus, in practice, one needs to compute precision and F_3 scores before choosing a model for runtime deployment, and not only rely on q-Risk scores.

[14] Recall that by safety monitoring, we refer to runtime monitoring of learned components and the system operational context to predict a system safety requirement violation, which is, different from predicting when a learned component might mispredict.

Our results further illustrate that the use of DL-based probabilistic forecasting methods, especially those with sequence-to-sequence architecture, leads to low inference latency while consuming feasible computing resources in terms of model size and peak memory usage during inference.

Furthermore, the results confirm the superiority of probabilistic forecasting over point forecasting for use in safety monitoring. This conclusion is drawn based on our empirical results and the fact that point forecast predictions correspond to the median ($q = 0.5$) value of the probability distribution predicted by probabilistic forecasting methods [13]. Our empirical results show that using values from the tail-end of the forecast probability distribution ($q \geq 0.95$ in our case) leads to more accurate safety metric and safety violation predictions than predictions for $q = 0.5$.

Window Configurations. We explored the effect of different combinations of varying hazard forecast horizons and context multipliers (*window configurations*), on prediction accuracy (RQ_3) and runtime performance (RQ_4), on the ACT case study only, due to the limited size of the dataset used for the ADS case study (as discussed in section 3.6.3). Given all the results discussed in section 3.6.5 and section 3.6.6, we conclude that for both ***cte*** and ***he*** safety requirements in the ACT safety monitoring problem, TFT is the best model to be used for predicting imminent safety violations, i.e., $h = 3\,\text{s}$, for all ***cm*** values. We further suggest that high ***cm*** values ($cm = 9$) be used as they improve overall safety violation accuracy. Nevertheless, as the peak inference memory of TFT increases with the increase in ***cm***, a lower ***cm*** might also be considered depending on the available GPU memory onboard the learning-enabled autonomous system. The results for $h = 12\,\text{s}$ further highlight that, although DeepAR has superior prediction accuracy than other models, it is not as accurate as TFT for $h = 3\,\text{s}$. Furthermore, the high average inference latency of DeepAR at $h = 12\,\text{s}$ prohibits it from being used as a safety monitor of the learned component, as discussed in section 3.6.6. However, if model optimizations and specialized inference hardware can reduce the DeepAR's inference latency to an acceptable range, it can be considered for predicting longer horizon safety violations given its good prediction accuracy. Nevertheless, considering both *accuracy* and *inference latency*, using TFT on shorter forecast horizons than 12 s, is a better option.

Challenging Scenarios. Although the trained safety metric forecasters yield high overall accuracy in predicting safety violations, it is important to identify the scenarios during which the safety monitor is more likely to mispredict safety violations. Characterizing such scenarios will allow the developers to generate more relevant execution data which can be used to train the safety monitors further and increase their safety violation prediction accuracy. Moreover, knowing the scenarios under which the safety monitor is expected to yield lower safety violation prediction accuracy would allow a system to be vigilant during the run-time of such scenarios and intervene in the automated operation of the system, if necessary.

One potential method relies on fitting a regression tree [77] to the safety violation prediction results (such as the ones provided in section 3.6.4). Concretely, a regression tree is fitted to a dataset whose features are the scenario parameters, and target variable is the F_3 score that the safety monitor yields for the corresponding scenario. A notable benefit of

a regression tree is that it allows the extraction of explainable rules that specify the part of the scenario space where the safety monitor yields a lower accuracy.

As an example, to explore the feasibility of explaining variation in safety violation prediction accuracy, we have fitted a regression tree to the safety violation prediction accuracy results of the safety monitor based on the TFT forecasting model, at the prediction quantile $q = 0.995$, for the *cte* safety requirement of the ACT case study (section 3.6.4). Therefore, the features of the regression tree are the ACT scenario parameters, i.e., *time of day*, *cloud cover*, and *starting cte and he of the aircraft* [15], while the target variable is the F_3 score of the TFT model at $q = 0.995$. Using grid search, we fitted regression trees by exploring combinations of values for its hyperparameters, i.e., *maximum depth* and *minimum number of samples per leaf node*. We computed the average *mean squared error* (MSE) for each model using 10-fold cross validation [35] and selected the most accurate tree, i.e., the one with the lowest average MSE over all ten folds ($MSE = 3 \times 10^{-4}$). The computed measure of determination (R^2) for the most accurate model is 0.68, indicating that most of the variance in F_3 is explained by the tree. We have provided the dataset used to train the regression tree, as well as the selected regression tree (with detailed accuracy metrics) in our replication package (see subsection 3.6.9).

Based on the most accurate regression tree, we observe that the following three rules characterize part of the scenario space where the safety monitor yields its lowest F_3 score:

- $\mathsf{time\_of\_day} \in \{\mathsf{afternoon}\} \wedge \mathsf{cloud\_cover} \in \{\mathsf{moderate}, \mathsf{high}\} \Longrightarrow F_3 = 0.952$
- $\mathsf{time\_of\_day} \in \{\mathsf{afternoon}\} \wedge \mathsf{cloud\_cover} \in \{\mathsf{none}, \mathsf{low}\} \Longrightarrow F_3 = 0.973$
- $\mathsf{time\_of\_day} \in \{\mathsf{morning}\} \wedge \mathsf{he}_{\mathsf{start}} \in [-10°, -7.5°] \Longrightarrow F_3 = 0.984$

Given the above rules, we observe that time of day, cloud cover conditions and starting *he* values are the most important features explaining variations in safety violation prediction accuracy across scenarios. We further conclude, based on the above rules, that the lowest accuracy scores are observed during the afternoon, when the sky is moderately or highly cloudy ($\mathsf{cloud\_cover} \in \{\mathsf{moderate}, \mathsf{high}\}$). As mentioned earlier, knowing the scenario subspace where prediction is challenging, specified by the above rules, can help the developer understand where more scenarios can be generated to re-train the safety monitor and potentially increase its prediction accuracy for low-accuracy scenarios. Moreover, the user can take the uncertainty of the safety monitor predictions into account at runtime, e.g., by being vigilant during scenarios where safety violation prediction accuracy is low and intervening when necessary.

3.6.8 Threats to Validity

In this section, we discuss potential threats to the validity of our study, namely internal, external, conclusion and construct validity [127, 145, 156].

[15] The detailed description and value ranges for the scenario parameters are provided in our replication package (subsection 3.6.9).

Internal Validity. Internal validity is concerned with the accuracy of the cause-and-effect relationships established by the experiments. Due to the limitations of the GluonTS library at the time of our evaluation, we had to use models for evaluation that were implemented in different ML frameworks. Concretely, DeepAR, MQCNN, and Seq2Seq models were implemented in MXNet while the TFT model was implemented in PyTorch, as the MXNet implementation of the TFT model was faulty and the MXNet models for MQCNN and Seq2Seq were not available. The use of different ML frameworks could impact the internal validity of the results. However, we conducted preliminary experiments on a model implemented in both frameworks[16] to compare the impact of differences in ML framework on safety metric forecasting accuracy and runtime performance (memory and time overhead). We found that the differences in accuracy and runtime performance metrics were less than the standard deviation of the measurements and negligible.

External Validity. External validity is concerned with the generalizability of our results.

One notable factor to consider is that in this study, we relied only on a specific ACT system (TinyTaxiNet) and simulation platform (X-Plane), for the ACT case study, and relied on a specific ADS system (Dave-2) and a simulation platform (Udacity simulator). However, X-Plane is a widely used high-fidelity simulator, and TinyTaxiNet was the best open-source ACT available at the time of our evaluation. Through preliminary experiments, we confirmed the superior cte estimation accuracy of TinyTaxiNet against two other pre-trained models that were available on the NASA ULI X-Plane Simulator project repository on GitHub [67]. Regarding the ADS case study, Dave-2 is a widely used lane keeping ADS, and Udacity is a popular simulator used for closed-track simulation of ADS. Nonetheless, further studies involving other learning-enabled autonomous systems in aviation and autonomous driving, as well as other domains, such as autonomous agriculture, and manufacturing, are required. We should however keep in mind that experiments such as the ones reported here entail substantial computations and extensive calendar time, i.e., 7500+ hours of GPU computation which were performed over 42 calendar days, thanks to having access to multiple GPUs on the Digital Research Alliance of Canada compute clusters.

Another relevant factor is that due to X-Plane's capabilities, we were only able to set the weather statically, i.e., without sudden changes that rarely happen. The same static weather has been used in the ADS simulation used by our study [131]. Nevertheless, both X-Plane and Udacity simulator are widely used high-fidelity simulators, as mentioned above. Moreover, given that the maximum duration of a scenario execution, in both the ACT and ADS cases studies, are less than 4 min and 2 min, respectively, assuming a static weather over the execution is not unreasonable.

An additional factor potentially impacting our proposed method's generalizability, is the fact that we assume that the monitored safety metrics are directly measurable (e.g., cte_{act} can be measured directly using a GPS) or can be estimated during the system operation (e.g., Time-To-Collision or TTC in the case of autonomous driving is estimated based the relative distance and velocity between the ADS and the object in front of it [90], which can be measured using a front radar on the ADS). However, this is not a restrictive

[16] At the time of our evaluation, only similar DeepAR implementations were available in both frameworks.

assumption as safety requirements, similar to any type of requirement, should have already been defined by the system developers and safety engineers, such that they are *measurable* [60]. Therefore, the safety requirements are expected to rely on metrics that can be measured or estimated to assess their satisfaction or violation.

Another factor that could impact the generalizability of our results relates to the fact that we have not evaluated the performance results of our models on embedded hardware similar to the one that might be used during the operation of real ACT and ADS systems. Nevertheless, our analysis revealed that the memory demand by the models is quite low and in line with what is reported in resource-constrained environments. Regarding the average inference latency measurements, model optimizations such as model quantization [117], can potentially improve the average inference latency of models, especially DeepAR such that its latency falls below runtime constraints. However, we have performed our latency measurements using cloud-strength NVIDIA V100 GPUs, in an inference setting, e.g., disabling gradient calculations [108, 109].

Conclusion Validity. Conclusion validity relates to the conclusions that can be drawn from the collected data and their statistical significance. We followed the widely accepted rule-of-thumb of 30 repetitions for the experiments and we report every statistical value with its confidence interval.

Construct Validity. Construct validity is concerned with the degree to which the measured variables in the study represent the underlying concept being studied. As discussed in section 3.6.3, q-Risk is a widely used metric to measure the accuracy of time series predictions (forecast safety metric values in our case) for a specific prediction quantile [13, 74, 75, 118], since it provides a summation of quantile loss (QL) over the forecast horizon for all predictions, normalized over all samples in the test set. As discussed in section 3.6.4, we have used precision and recall which are widely used as metrics to capture the accuracy of the models in terms of missed safety violations and false prediction, respectively. Furthermore, similar to [131], we have used F_3 score as an aggregate metric to compare the overall safety violation prediction accuracy of safety monitors while capturing the relative importance of false negatives and false positives. As discussed in section 3.6.6, the performance overhead introduced by the safety monitor can be refined to space and time overheads. Since models are loaded in the GPU, GPU memory usage is a metric that successfully captures the space overhead of the models as model size and peak memory usage during inference. Whereas, average prediction latency effectively captures the time overhead of the model at runtime.

3.6.9 Data Availability

The evaluated DL-based probabilistic forecasting models have been implemented in Python. We made the aforementioned implementations, the instructions to set up the ACT case study, the detailed description of the scenario parameters used to generate data, the generated ACT dataset, the raw and preprocessed ADS dataset, and the detailed evaluation results, for both the ACT and ADS case studies, available in our replication package [123].

3.7 Conclusion

In this chapter, we proposed a method for safety monitoring of learned components in autonomous systems via probabilistic safety metric forecasting. We address the practical challenges of lacking access to internal information of the learned component and the system having limited operational resources, by using state-of-the-art DL-based probabilistic time series forecasters. They rely on scenarios and learned component output values to provide predictions of the safety metric probability distribution with acceptable inference latency and memory usage. We apply these forecasters to widely used case studies in autonomous aviation and autonomous driving, namely ACT and lane keeping ADS, respectively, where we run extensive experiments to evaluate the safety metric and violation prediction accuracy, inference latency, and computation resource usage of state-of-the-art models, with a varying lookback and hazard forecast horizons while comparing them against a very competitive baseline (DeepAR). Our evaluation results suggest that probabilistic forecasting of safety metrics, given learned component outputs and scenarios, is effective for safety monitoring. Moreover, the evaluation results show that using Temporal Fusion Transformer (TFT) for predicting imminent safety violations ($h = 3\,\mathrm{s}$), for all lookback horizons, leads to the most accurate predictions with acceptable inference latency while consuming reasonable computational resources.

Table 3.4: Various safety violation prediction accuracy metric values for different models and quantiles, for ***cte*** and ***he*** (rows highlighted in gray) safety requirements.

Model	Average Metric Value ± $0.5 \times CI_{0.95}$			
	$q = 0.5$	$q = 0.95$	$q = 0.975$	$q = 0.995$
		FN		
Seq2Seq	2062.9 ± 111.5	2022.0 ± 114.5	1949.7 ± 142.3	1669.6 ± 198.9
	1400.9 ± 32.9	1138.8 ± 43.8	1073.4 ± 42.0	989.4 ± 44.7
DeepAR	2236.5 ± 329.5	124.5 ± 73.9	46.0 ± 31.1	10.3 ± 7.6
	1658.8 ± 20.3	1130.5 ± 40.0	934.1 ± 45.2	577.7 ± 46.2
MQCNN	2128.0 ± 199.9	1531.2 ± 201.6	1372.3 ± 174.0	1037.6 ± 211.2
	1344.7 ± 40.3	961.7 ± 65.9	826.2 ± 67.1	594.6 ± 73.2
TFT	729.2 ± 27.2	131.4 ± 17.6	78.1 ± 11.6	16.8 ± 2.5
	866.9 ± 20.4	420.0 ± 24.3	325.7 ± 22.3	153.9 ± 15.6
		FP		
Seq2Seq	1073.4 ± 129.0	1108.9 ± 150.6	1191.4 ± 175.6	1642.5 ± 291.9
	958.8 ± 165.1	2395.2 ± 402.2	2953.1 ± 460.7	3982.8 ± 528.9
DeepAR	3311.0 ± 684.6	14501.4 ± 1113.0	16781.9 ± 998.2	19858.7 ± 842.3
	276.4 ± 24.8	2337.9 ± 240.0	4420.1 ± 455.4	13203.1 ± 1093.1
MQCNN	732.8 ± 110.2	1243.0 ± 160.6	1457.3 ± 186.4	2224.9 ± 336.1
	533.7 ± 76.3	2032.3 ± 363.5	3118.9 ± 522.3	6069.3 ± 1018.0
TFT	348.4 ± 13.3	1677.3 ± 61.3	2190.1 ± 78.9	4026.0 ± 138.0
	344.6 ± 19.1	1752.8 ± 73.0	2385.5 ± 102.9	4808.9 ± 221.0
		Precision		
Seq2Seq	0.978 ± 0.0026	0.977 ± 0.0030	0.975 ± 0.0035	0.967 ± 0.0056
	0.496 ± 0.0304	0.343 ± 0.0254	0.309 ± 0.0237	0.261 ± 0.0233
DeepAR	0.936 ± 0.0120	0.773 ± 0.0138	0.746 ± 0.0116	0.712 ± 0.0089
	0.693 ± 0.0156	0.338 ± 0.0186	0.242 ± 0.0162	0.118 ± 0.0082
MQCNN	0.985 ± 0.0022	0.975 ± 0.0031	0.970 ± 0.0036	0.956 ± 0.0062
	0.646 ± 0.0261	0.422 ± 0.0330	0.343 ± 0.0304	0.249 ± 0.0335
TFT	0.993 ± 0.0003	0.967 ± 0.0012	0.957 ± 0.0015	0.924 ± 0.0024
	0.804 ± 0.0081	0.515 ± 0.0085	0.451 ± 0.0087	0.308 ± 0.0086
		Recall		
Seq2Seq	0.958 ± 0.0023	0.959 ± 0.0023	0.960 ± 0.0029	0.966 ± 0.0041
	0.383 ± 0.0145	0.499 ± 0.0193	0.527 ± 0.0185	0.564 ± 0.0197
DeepAR	0.954 ± 0.0067	0.997 ± 0.0015	0.999 ± 0.0006	1.000 ± 0.0002
	0.270 ± 0.0089	0.502 ± 0.0176	0.589 ± 0.0199	0.746 ± 0.0203
MQCNN	0.956 ± 0.0041	0.969 ± 0.0041	0.972 ± 0.0036	0.979 ± 0.0043
	0.408 ± 0.0178	0.577 ± 0.0290	0.636 ± 0.0295	0.738 ± 0.0322
TFT	0.985 ± 0.0006	0.997 ± 0.0004	0.998 ± 0.0002	1.000 ± 0.0001
	0.618 ± 0.0090	0.815 ± 0.0107	0.857 ± 0.0098	0.932 ± 0.0069
		F_3		
Seq2Seq	0.960 ± 0.0019	0.960 ± 0.0020	0.962 ± 0.0024	0.966 ± 0.0032
	0.390 ± 0.0125	0.472 ± 0.0122	0.486 ± 0.0107	0.498 ± 0.0103
DeepAR	0.952 ± 0.0049	0.969 ± 0.0012	0.966 ± 0.0015	0.961 ± 0.0015
	0.287 ± 0.0090	0.476 ± 0.0135	0.510 ± 0.0129	0.481 ± 0.0146
MQCNN	0.959 ± 0.0036	0.969 ± 0.0035	0.972 ± 0.0030	0.976 ± 0.0034
	0.423 ± 0.0170	0.549 ± 0.0212	0.576 ± 0.0182	0.595 ± 0.0159
TFT	**0.986 ± 0.0005**	**0.994 ± 0.0002**	**0.994 ± 0.0001**	**0.992 ± 0.0003**
	0.633 ± 0.0084	**0.770 ± 0.0079**	**0.785 ± 0.0064**	**0.774 ± 0.0048**

Table 3.5: Various safety violation prediction accuracy metric values for different models and quantiles, for the ADS case study.

Model	Average Metric Value ± $0.5 \times CI_{0.95}$			
	$q = 0.5$	$q = 0.95$	$q = 0.975$	$q = 0.995$
		FN		
Seq2Seq	56.8 ± 1.4	31.0 ± 3.7	19.5 ± 4.9	6.6 ± 4.3
DeepAR	61.4 ± 0.7	30.6 ± 1.2	23.5 ± 1.5	13.2 ± 1.8
MQCNN	59.3 ± 2.0	20.0 ± 3.7	8.4 ± 3.2	2.2 ± 1.8
TFT	55.6 ± 1.8	12.3 ± 1.6	7.7 ± 1.2	2.2 ± 0.3
		FP		
Seq2Seq	22.0 ± 1.9	45.5 ± 5.5	80.1 ± 18.7	158.0 ± 23.5
DeepAR	2.9 ± 0.3	48.5 ± 3.1	64.1 ± 3.2	87.4 ± 4.4
MQCNN	14.4 ± 2.5	67.7 ± 14.0	126.0 ± 21.7	180.7 ± 16.4
TFT	12.9 ± 1.5	55.1 ± 3.1	67.1 ± 3.0	97.1 ± 6.0
		Precision		
Seq2Seq	0.810 ± 0.0128	0.728 ± 0.0153	0.649 ± 0.0380	0.505 ± 0.0449
DeepAR	0.968 ± 0.0035	0.711 ± 0.0120	0.663 ± 0.0101	0.610 ± 0.0113
MQCNN	0.867 ± 0.0179	0.676 ± 0.0311	0.555 ± 0.0405	0.460 ± 0.0282
TFT	0.880 ± 0.0105	0.714 ± 0.0101	0.679 ± 0.0085	0.605 ± 0.0142
		Recall		
Seq2Seq	0.619 ± 0.0092	0.792 ± 0.0246	0.869 ± 0.0329	0.955 ± 0.0289
DeepAR	0.588 ± 0.0047	0.795 ± 0.0079	0.842 ± 0.0097	0.911 ± 0.0121
MQCNN	0.602 ± 0.0131	0.866 ± 0.0248	0.943 ± 0.0216	0.985 ± 0.0122
TFT	0.627 ± 0.0121	0.918 ± 0.0109	0.949 ± 0.0080	0.985 ± 0.0022
		F_3		
Seq2Seq	0.634 ± 0.0084	0.784 ± 0.0197	0.833 ± 0.0215	0.866 ± 0.0144
DeepAR	0.612 ± 0.0046	0.785 ± 0.0062	0.820 ± 0.0079	0.868 ± 0.0096
MQCNN	0.620 ± 0.0119	0.838 ± 0.0169	0.874 ± 0.0099	0.880 ± 0.0028
TFT	**0.645 ± 0.0113**	**0.892 ± 0.0086**	**0.912 ± 0.0057**	**0.927 ± 0.0031**

Table 3.6: Statistical comparison of F_3 score values for different DL-based forecasters at quantiles q≥ 0.5, for ***cte*** and ***he*** safety requirements of the ACT case study and the ***cte*** safety requirement of the ADS case study, respectively.

Comparison		F_3 score							
A	*B*	$q = 0.5$		$q = 0.95$		$q = 0.975$		$q = 0.995$	
		p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$	p	$\hat{A}_{AB}$
TFT	Seq2Seq	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
TFT	DeepAR	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
TFT	MQCNN	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
Seq2Seq	DeepAR	1.84×10^{-2}	0.68	1.16×10^{-7}	0.10	1.60×10^{-3}	0.26	3.39×10^{-2}	0.66
Seq2Seq	MQCNN	7.62×10^{-1}	0.48	2.28×10^{-5}	0.18	1.25×10^{-5}	0.17	3.37×10^{-5}	0.19
DeepAR	MQCNN	3.64×10^{-2}	0.34	1.37×10^{-1}	0.39	4.71×10^{-4}	0.24	7.69×10^{-8}	0.10
		ACT_{he}							
TFT	Seq2Seq	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
TFT	DeepAR	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
TFT	MQCNN	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00	3.02×10^{-11}	1.00
Seq2Seq	DeepAR	8.99×10^{-11}	0.99	5.59×10^{-1}	0.46	5.26×10^{-4}	0.24	3.64×10^{-2}	0.66
Seq2Seq	MQCNN	3.34×10^{-3}	0.28	3.01×10^{-7}	0.11	1.55×10^{-9}	0.05	1.96×10^{-10}	0.02
DeepAR	MQCNN	6.70×10^{-11}	0.01	8.20×10^{-7}	0.13	6.53×10^{-7}	0.13	7.39×10^{-11}	0.01
		ADS_{cte}							
TFT	Seq2Seq	1.41×10^{-1}	0.61	2.92×10^{-9}	0.95	2.19×10^{-8}	0.92	1.61×10^{-11}	1.00
TFT	DeepAR	2.58×10^{-5}	0.82	3.01×10^{-11}	1.00	3.01×10^{-11}	1.00	4.96×10^{-11}	0.99
TFT	MQCNN	9.88×10^{-3}	0.69	4.98×10^{-7}	0.88	1.98×10^{-8}	0.92	7.85×10^{-12}	1.00
Seq2Seq	DeepAR	2.12×10^{-4}	0.78	5.89×10^{-1}	0.46	2.77×10^{-1}	0.58	1.52×10^{-1}	0.61
Seq2Seq	MQCNN	6.25×10^{-2}	0.64	9.79×10^{-5}	0.21	3.25×10^{-2}	0.34	6.87×10^{-1}	0.47
DeepAR	MQCNN	2.97×10^{-1}	0.42	1.49×10^{-6}	0.14	1.74×10^{-8}	0.08	6.60×10^{-3}	0.30

Chapter 4

Discussion

In this chapter, we discuss the broader impacts that the outcomes of the research presented in this thesis can have on practitioners and industry, as well as policy makers, and the general public (society).

4.1 Industry Considerations

In this section we discuss the higher level, practical considerations of the tools presented in this thesis, to be used by system developers.

Safety requirements. It is important to note that to be able to successfully use the hazard boundary identification and safety monitoring tool, system safety requirements should be measurable, i.e., their (degree of) satisfaction can be measured via safety metrics. However, given that measurability is one of the fundamental characteristics of a well-formed requirement [60], we can assume that applying the methods and tools proposed in this research does not pose additional burden in terms of reframing requirements to be measurable, as this should have already been done by system and requirement engineers during the system design process [60]. It is worth noting that autonomous systems deployed in safety-critical settings, such as an ADS or a service robot, are interacting with the public directly. Thus, they should have to additionally satisfy system safety requirements stemming from ethical and trust considerations. The satisfaction of such requirements are more challenging to measure, as they involve normative and subjective concepts that have to be operationalized, as opposed to objective physical properties involved in physics-based safety requirements, e.g., minimum safe distance.

Simulation infrastructure. As discussed in chapter 2, to evaluate the safety outcome of a combination of scenario and learned component output, the system has to be executed in a simulator to enable measurements the safety metric values. Although the search algorithm that we proposed is significantly more effective and efficient compared to other baselines, we observed that increasing the compute resource for simulation increases the number of combinations that are close to the hazard boundary (Figure 2.6). Furthermore, the larger the number of combinations searched by MLCSHE, the larger the number of hazard

boundary samples upon which safety monitors (chapter 3) can be trained on. Therefore, it is important for the developers of AI-enabled autonomous systems to setup a simulation infrastructure that is:

1. *configurable:* the relationship among environmental objects are captured by a scenario domain model where their parameters can easily be controlled to define (ideally using a dedicated *scenario specification language*) and simulate a wide variety of scenarios;

2. *high-fidelity:* can realistically simulate physical interactions (ideally without requiring prohibitively high amount of computation), e.g., implementation of variable road friction based on different weather conditions for ADS;

3. *controllable:* can easily measure, record and control environmental and system parameters, usually implemented through an API;

4. *concurrent:* multiple simulations can run in parallel on distributed or dedicated compute resources without adversely impacting each other.

There are also ways to improve the compute demand for simulations, which will be discussed in section 5.2.

Parameter observability. In both chapter 2 and chapter 3, we have assumed that many parameters are observable during simulation and operation of the system. They include scenario parameters such as environmental conditions, as well as the time series of parameters necessary for computing the safety metric values. This highlights the need for a simulator with sufficient fidelity in identifying the hazard boundary. The required level of simulator fidelity depends the operational context (complexity) of the learning-enabled system. For instance, a simulator for a robotic arm involved in automated manufacturing does not need to be as high-fidelity as a simulator for an AV, as the former has a relatively constant surrounding environment while the latter has a much more diverse and complex surrounding environment. Due to the recent leaps in simulator quality and fidelity, we expect this to be less of an issue in the future. Finally, as noted above, for safety monitoring, parameters selected to characterize a scenario and compute the safety metric should be observable by the system. For instance, in the ACT case study in chapter 3, weather conditions and centerline deviation (to compute the safety metrics) are measurable via sensors other than the camera, i.e., the former is provided by weather satellites while the latter can be computed using GPS coordinates (although with higher latency than the camera-based estimator). Again, the advances in sensor development and low latency connectivity (5G and 6G communication), have improved and will continue to improve the limitations caused due to this consideration.

Safety Monitor Evaluation. In chapter 3, we proposed a safety monitoring method evaluated in terms of prediction accuracy and runtime constraints. However, developers should also consider assessing the safety monitor in terms of *robustness*, *explainability*, and *completeness*. In subsection 3.6.7, we introduced and demonstrated a potential method for evaluating the explainability of safety monitors. This approach identifies the operational contexts in which the safety monitor forecasts are less accurate and serves as an additional

uncertainty metric associated with the safety monitor forecasts at runtime. In the context of safety monitoring, *robustness* refers to the ability of the safety monitor to provide consistent forecasts under perturbed inputs. While robustness analysis methods such as fuzz testing [70] are indeed necessary for further evaluating safety monitors before deployment, the method discussed above can also offer valuable insights into the safety monitor's robustness to variations in operational contexts. Finally, regarding *completeness*, we argue that practitioners should ensure a safety monitor is trained on and evaluated against an authoritative set of scenarios (operational contexts) before runtime deployment. This scenario set should be carefully defined for the target geographic area and operational design domains (ODDs). For example, the scenario set for an autonomous taxi operating in Ottawa, ON, should not only include nominal highway and urban driving scenarios but also extreme winter conditions, such as freezing rain or blizzards on rural roads lacking visible markings. Recent initiatives, such as the *Safety Pool Scenario Database*[1], offer a valuable repository of scenarios collected from municipalities worldwide, serving as an excellent starting point for practitioners.

4.2 Regulatory and Societal Considerations

The methods and tools presented in this thesis can be applied to any learning-enabled autonomous system such as in driving, aviation, manufacturing, and agriculture. However, discussing their impact on regulations and society depends on their application context. For the rest of this discussion, we focus on automated driving as it is arguably the field with the most development, interest, investment and near-term impact on society.

As described in the hierarchical socio-technical safety control structure of a system, presented by Leveson [73], a system's operation and development process is subject to regulations enforced by local, national and international authorities. For instance, for an autonomous vehicle to be able to operate in California, its operation should have been sanctioned by US Department of Transportation (USDOT), National Highway Transportation Safety Administration (NHTSA), and California Department of Motor Vehicles (CA DMV), the national and state regulatory bodies. This happens either through permits issued under previously issued regulations or under special permits which are issued before comprehensive regulation is enacted, such as the Autonomous Testing and Deployment permits issued by CA DMV[2].

Currently, the overall regulatory environment is characterized by a mix of federal, state, and international laws, which is rapidly evolving while trying to establish frameworks that establish a balance among various aspects such as safety, innovation, public acceptance, integration with existing transportation system and liability [143]. Due to the dynamic and evolving regulatory landscape, many regulators, in different jurisdictions, can mainly rely on international industry standards as a baseline for safety evaluation and certification of autonomous vehicles to issue public road testing and deployment permits [93].

[1] `https://www.safetypool.ai/database`

[2] `https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-testing-permit-holders/`

Notable industry standards concerned with functional and system safety of autonomous vehicles include: ISO 26262 [57], ISO 21448 [58], and ANSI/UL 4600 [4].

The ISO 26262 standard [57], titled "*Road Vehicles – Functional Safety*," is a standard for the functional safety of electrical and electronic systems in road vehicles. It focuses on ensuring that automotive systems meet safety requirements throughout their lifecycle, mostly in the context of potential hazards caused by system malfunctions [57]. The standard provides a risk-based approach, which classifies hazards according to severity, exposure, and controllability to determine the appropriate Automotive Safety Integrity Level (ASIL) [57]. It is worth noting that ISO 26262 covers safety from component to system level, over all the stages of the system development lifecycle, including testing and operation.

The ISO 21448 standard [58], titled "*Road Vehicles – Safety of the Intended Functionality (SOTIF)*", focuses on addressing safety concerns that arise from the intended functions of a system, particularly in advanced driver assistance systems (ADAS) and automated driving systems (ADS). Unlike ISO 26262, which deals with risks due to system malfunctions [57], ISO 21448 addresses scenarios where the system functions as designed but may still encounter safety issues due to limitations in sensor performance, environmental conditions, or unexpected use cases [58]. The standard emphasizes a comprehensive process that starts with hazard identification and risk assessment, followed by design and development activities aimed at mitigating identified risks.

Testing plays a crucial role in ISO 21448, where verification and validation processes are used to ensure that the system performs safely in real-world conditions [58]. This includes rigorous scenario-based testing to evaluate the system response to a wide range of operational scenarios, including those that might not have been foreseen during the design phase. The standard also highlights the importance of simulation and field testing to capture edge cases and scenarios that are difficult to replicate in controlled environments [58].

Runtime monitoring is another essential aspect of ISO 21448, focusing on the continuous observation of system performance during operation. This involves implementing mechanisms to detect when the system is approaching the limits of its intended functionality, enabling real-time adjustments or interventions to maintain safety [58].

Finally, the ANSI/UL 4600 standard [4], titled "*Standard for Safety for the Evaluation of Autonomous Products*", is a safety standard specifically designed for autonomous systems, including autonomous vehicles and robots. It focuses on addressing functional safety, system safety, and validation of fully autonomous systems, where there is no human intervention or fallback in case of system failure. UL 4600 differs from ISO 26262 and ISO 21448 in that it is designed specifically for fully autonomous operations and covers a broader range of autonomous systems beyond just road vehicles. The standard relies on the concept of a *safety case*, which is a structured argument backed by evidence demonstrating that the autonomous system operates safely. The safety case must address potential hazards and ensure that risks are reduced to an acceptable level [4].

Similar to ISO 26262 and ISO 21448, UL 4600 requires performing a hazard and risk analysis. However, UL 4600 focuses on identifying hazards specifically related to autonomous decision-making and environmental perception. The standard outlines a process

for safety assessment that includes scenario-based testing, simulation, and field testing, ensuring that systems can operate safely under both nominal and edge case conditions. UL 4600 requires that autonomous systems be designed with capabilities for continuous monitoring and self-assessment during operation. This ensures that the system can detect anomalies, adapt to new conditions, and be updated as needed to address unforeseen safety issues [4]. UL 6400 also addresses many other aspects related to autonomous vehicles such as cybersecurity, human-machine interaction, and ethical considerations. We refer the interested reader to the standards themselves for further details.

The methods for hazard boundary identification and safety monitoring that we proposed in this thesis can be useful in ensuring that an AV adheres to the above standards. As mentioned above, ISO 26262 focuses on safety violations resulting from faults in components, including learning-based software components. Our hazard boundary identification method (MLCSHE), described in chapter 2, provides the means to identify the cases where the learned component malfunction, e.g., miclassification of objects detected by the perception component, leads to a system-level safety violation. Moreover, as MLCSHE searches the scenario space using a high-fidelity simulator, the identified hazard boundary can take into account the edge cases, as well as system safety violations caused by complex component interactions, which is the focus for ISO 21448, as discussed above. Finally, the approximate hazard boundary of the learned component, identified by MLCSHE, can be used as (in)direct supporting evidence for the safety case arguments developed under UL 4600 guidelines. As discussed in subsection 2.4.4, the identified hazard boundary is independent of the learned component implementation itself. Thus, by comparing the outputs generated by the actual learned component, used in an AV during operation, under various scenarios, with the approximated hazard boundary, one can demonstrate that the system does not violate the safety requirements under the combinations covered by the hazard boundary.

Regarding runtime operations of AVs, the safety monitoring method proposed in chapter 3, not only monitors for safety violations that are caused by learned component faults, it also takes into account edge cases and complex component interactions as the dataset used to train the safety monitors is based on high-fidelity simulations of the system in diverse scenarios. Thus, the method presented in chapter 3, provides a valuable means to ensure that AVs can adhere to runtime monitoring demands of ISO 26262, ISO 21448, and UL 4600.

Finally, we would like to briefly address the impact our tools and methods can have on the overall safety certification process performed by the certification authorities. Recent examples of the authorities delegating their certification expertise to system developers themselves, e.g., the delegation of many certification activities of the B-737 Max aircraft to Boeing itself by the Federal Aviation Authority (FAA) [98], have shown to be fatal, as they "erode the oversight capability" [98] of the authority. Therefore, certification authorities need to be capable in evaluating the safety of AVs using unified methods and tools that can also preserve the confidentiality of the design of various AVs. Overall, black-box methods, such as the ones proposed in this thesis, that enable identification of the hazard boundary of the system and runtime monitoring, without making specific assumptions on the architecture of the learned component, allow them to be applied to

various AV providers in the industry, without requiring confidential system information. Specifically, MLCSHE can be used to identify the hazard boundaries of various critical learned components of the AVs without requiring the certification authority to have access to their design details. Recall that an executable model (or even and API) of the system that can be executed in a simulator is sufficient to run the search. Alternatively, the AV developers can execute the search, where the algorithm and its convenience functions and tools are encapsulated in a library approved by the certification authority, and report the execution results to such authority. Therefore, MLCSHE provides the regulators and certification authorities with a unified means to evaluate the impact of learned components on the safety of learning-enabled autonomous systems during the certification process.

Furthermore, the black-box safety monitoring method proposed in chapter 3, can potentially provide a path for policy makers to demand runtime safety monitoring mechanisms from AV providers. The safety monitoring method based on safety metric forecasting requires execution data which can be collected while simulation-based testing of the AV is being conducted by the developers. Note that simulation-based and scenario-based testing of AVs is common practice and is usually conducted as part of their development and testing. Furthermore, our evaluation methodology provides the metrics and the process to assess the accuracy of the trained safety monitors.

Chapter 5

Conclusion

In this chapter, we provide a summary of contributions and identify future work items.

5.1 Summary of Contributions

In summary, this thesis provides the following contributions:

1. *MLCSHE*, a hazard boundary identification method that can identify learned-component output and scenario combinations that exist on the *hazard boundary* of a learned component, using cooperative coevolutionary search (section 2.5). Our contributions related to MLCSHE include:
   (a) A probabilistic fitness function to estimate the distance of a learned component output and scenario combination to the hazard boundary (subsection 2.5.2).
   (b) An application of MLCSHE to a complex autonomous driving case study, i.e., Pylot and CARLA (section 2.6).
   (c) A Python-based implementation of MLCSHE with a simulation manager to run parallel simulations (subsection 2.6.5).
   (d) An empirical evaluation of the effectiveness and efficiency of MLCSHE and its comparison with random search and genetic algorithm baselines, using large-scale Pylot and CARLA experiments (section 2.6).
2. A runtime safety monitoring method that provides early warnings of safety violations using probabilistic temporal forecasting of system safety metrics. Relevant contributions to our safety monitoring method are:
   (a) A low-latency method to forecast the safety metric values in the near future using the history of the safety metric, learned component outputs and its operational context (section 3.5).

(b) An application of the safety monitoring method to two safety requirements of a widely used autonomous aviation case study, including a dataset generated from system-in-the-loop simulations (section 3.6).

(c) A large-scale empirical evaluation of state-of-the-art DL-based time series forecasting models assessing safety metric and violation prediction accuracy as well as runtime latency and memory usage over various window configurations (section 3.6).

(d) A Python-based implementation of the models as well as detailed hyperparameter tuning and data generation instructions are provided (subsection 3.6.9).

3. A discussion on potential impacts of the above methods and tools on industry and society.

Table 5.1 summarizes the relevance of the contributions of this thesis with respect to the objectives stated in section 1.2, which are repeated below for convenience:

TO_1 (Hazard Boundary Identification): Identify the combinations of system contexts and learned components' behaviors that lead the system to a hazard state, i.e., the hazard boundary of learned components.

TO_2 (Safety Monitoring): Monitor the system during operation (at runtime) and predict if the behavior of learned components can lead to system safety violations (i.e., hazard) given its operational context.

Table 5.1: Thesis objective-contribution traceability matrix

Thesis Objectives	Contributions		
	1	2	3
TO_1	x		x
TO_2		x	x

5.2 Future Work

As learning-enabled autonomous systems become more complex and ubiquitous, so does the need and potential for growth in research aimed at making them safer at the design, development, and operation stages of their life cycle. Below, we present several immediate and long-term opportunities for future research.

Other case studies. Although we have studied various case studies in the automated driving and autonomous taxiing domain over large-scale experiments, there is still a need to further evaluate and validate the proposed methods and tools in other application settings, e.g., industrial manufacturing, agriculture, search and rescue. Moreover, the same case

studies can be further studied under different ODDs, e.g., a highway driving ODD for the automated driving case study evaluated in chapter 2.

Integration testing of learned components. We believe that the identified hazard boundary using methods and tools introduced in chapter 2 can also be used to further improve the integration testing of a learned component. As described in chapter 2, MLCSHE jointly searches the space of scenarios and learned component behavior without relying on a specific implementation of the learned component. Thus, if the ODD and the architecture of the system remains the same, the identified hazard boundary does not change as well. Therefore, using the information in the hazard boundary, we can test to ensure that given a scenario, a learned component does not generate an output sequence that we know will lead to system safety violation. This is useful in cases where the model is changed, via re-training for instance, or is replaced with another model during the system operation.

Increased MLCSHE search efficiency. We have already shown in our evaluations that MLCSHE is more efficient than its comparable baselines. Nevertheless, it is still a costly search that can benefit from further efficiency in search. One way to improve the search can be to leverage ideas from surrogate-assisted search methods [43], where the outcome of a scenario and learned component output combination is estimated, instead of being executed in a simulator, based on surrogate models that are trained using a limited number of simulations. Over the course of the search, these surrogate models can become more accurate using additional simulations for cases where the surrogate models' uncertainty grows large [43]. We expect the above suggestion to provide a major improvement to search efficiency as the high-fidelity simulations conducted during the hazard boundary search consume the most significant portion of the search budget. Another potential way to improve the search efficiency is to take advantage of methods developed based on formal methods. Methods such as the ones mentioned in section 2.4, can provide over-approximations of the learned component and system output regions given various scenarios. Despite the fact that they currently face many practicality and scalability limitations, if such methods were to be successfully combined with MLCSHE in such a way that they could determine the regions that the search should focus on, e.g., the over-approximated hazard boundary, they can reduce the size of the search space significantly, thus improving the efficiency of the search.

Increased tool availability. The tools developed during the course of this research are available online and have been developed such that they can be reused by other researchers and practitioners. However, we believe that they can be made more available and usable for a wider community use. First, the code for the MLCSHE algorithm can be further refactored such that it can be integrated into one or more widely used searched-based algorithms libraries such as DEAP[1]. Second, the safety monitoring tools developed in this thesis and the used models can be packaged in Docker[2] images significantly facilitating the setup, training and deployment of such models.

Emerging learning-enabled autonomous systems. Recent advances in machine learning have led to the emergence of novel learning-enabled autonomous system architectures. Con-

[1] `https://deap.readthedocs.io/en/master/`

[2] `https://docs.docker.com/`

ventionally, autonomous systems observe, process and interact with the environment using a perception-prediction-control cycle [88]. However, the advent of end-to-end learning-enabled systems and foundation models is starting to change that. In the case of end-to-end learning-enabled systems, such as the ones mentioned in the survey by Chib and Singh [25], a large deep learning based model consisting of various layers and architectures, directly connects the environmental inputs to control actions of the system. In another case, with the emergence of Large Language Models (LLMs) and (vision) foundation models, new autonomous system architectures have already been proposed that utilize such models to perceive the environment, reason and make decisions [23,30]. Furthermore, foundation models have shown promising results in handling novel scenarios (i.e., zero-shot inference) [31]. Both of the emerging autonomous systems mentioned above introduce additional safety challenges. The end-to-end autonomous systems currently lack the modularity that exists in the conventional component-based autonomous systems, which can be used to introduce redundancy to the architecture and thus make them more reliable. For foundation model-based autonomous systems, aside from them being at early stages of research and development, leading to basic challenges such as runtime performance [88], they potentially suffer from having a single point of failure, i.e., the foundation model, which is known to have its unique safety challenges such as hallucinations [9, 150]. Nevertheless, due to the promising effectiveness of the above architectures, they are likely to be deployed in safety-critical applications in the future. Therefore, the need for further research on making these emerging autonomous systems safer is crucial. We believe that the ideas and tools introduced in this thesis can play an integral part in defining the hazard boundary of such systems and monitoring them at runtime.

Improved usability. It is *unlikely* that the practitioners would like to use the tools and methods proposed in this thesis are experts in cooperative coevolutionary search and probabilistic temporal forecasting. Rather, they are probably safety engineers who would like to rely on our proposed tools as part of their overall safety assurance framework. Although we have tried to explicitly describe the decision making process for choosing the parameters of the search algorithm and the forecasting models, and provide detailed installation and execution instructions in our implementations, more thorough guidelines and rules of thumb (such as the ones provided in handbooks [38, 119]) can be developed both in general terms as well as context-specific cases for various common application contexts, to further improve the usability of such tools for practitioners.

Multiple safety metrics. In this thesis, we have proposed methods and tools while focusing on a single safety requirement and metric at a time[3]. However, a learning-enabled autonomous system has to satisfy numerous safety requirements, each of which can potentially be measured using multiple safety metrics. Although our proposed methods and tools can be applied to various safety requirements and metrics individually, we believe that they can also be modified to take multiple safety metrics into account at the same time. In the context of safety monitoring, this would translate to probabilistic *multivariate* forecasting where multiple safety metrics forecasts can be jointly generated. Thus, the use of alternative probabilistic multivariate DL-based forecasting models [95, 111, 112] should

[3]We have addressed different safety requirements and metrics over the various case studies evaluated in this research.

be considered. For hazard boundary identification, the search problem would potentially be changed into a many-objective cooperative coevolutionary search problem [55], which would probably impact the definition of the fitness function. In both cases, further research and investigation are required.

References

[1] CARLA Team, Intel Autonomous Agents Lab, the Embodied AI Foundation, and AlphaDrive. CARLA Autonomous Driving Leaderboard. `https://leaderboard.carla.org/leaderboard/`, 2022. [Online; accessed 09-November-2022].

[2] Arden Albee, Steven Battel, Richard Brace, Garry Burdick, John Casani, Jeffrey Lavell, Charles Leising, Duncan MacPherson, Peter Burr, and Duane Dipprey. Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions. *Jet Propulsion Laboratory, Pasadena, CA, Tech. Report. JPL D-18709*, 2000.

[3] Alexander Alexandrov, Konstantinos Benidis, Michael Bohlke-Schneider, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Danielle C. Maddix, Syama Rangapuram, David Salinas, Jasper Schulz, Lorenzo Stella, Ali Caner Türkmen, and Yuyang Wang. GluonTS: Probabilistic and Neural Time Series Modeling in Python. *Journal of Machine Learning Research*, 21(116):1–6, 2020.

[4] ANSI/UL. Evaluation of Autonomous Products. Standard, American National Standards Institute and UL Standards and Engagements, Northbrook, IL, USA, March 2023.

[5] Panos J. Antsaklis, James A. Stiver, and Michael Lemmon. Hybrid System Modeling and Autonomous Control Systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, pages 366–392, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[6] Erfan Asaadi, Steven Beland, Alexander Chen, Ewen Denney, Dragos Margineantu, Matthew Moser, Ganesh Pai, James Paunicka, Douglas Stuart, and Huafeng Yu. Assured Integration of Machine Learning-based Autonomy on Aviation Platforms. In *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, pages 1–10, San Antonio, TX, USA, 2020. IEEE, Institute of Electrical and Electronics Engineers (IEEE).

[7] Erfan Asaadi, Ewen Denney, and Ganesh Pai. Towards Quantification of Assurance for Learning-Enabled Components. In *2019 15th European Dependable Computing Conference (EDCC)*, pages 55–62, New York, NY, US, Sep. 2019. IEEE.

[8] Erfan Asaadi, Ewen Denney, and Ganesh Pai. Quantifying Assurance in Learning-Enabled Systems. In António Casimiro, Frank Ortmeier, Friedemann Bitsch, and

Pedro Ferreira, editors, *Computer Safety, Reliability, and Security*, pages 270–286, Cham, 2020. Springer International Publishing.

[9] Suriya Ganesh Ayyamperumal and Limin Ge. Current state of LLM Risks and AI Guardrails. `https://arxiv.org/abs/2406.12934`, 2024.

[10] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. ACM, New York, NY, USA, 1999.

[11] Subho S. Banerjee, Saurabh Jha, James Cyriac, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Hands Off the Wheel in Autonomous Vehicles?: A Systems Perspective on over a Million Miles of Field Data. In *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 586–597, 2018.

[12] Tony Bellotti, Ilia Nouretdinov, Meng Yang, and Alexander Gammerman. Chapter 6 - feature selection. In Vineeth N. Balasubramanian, Shen-Shyang Ho, and Vladimir Vovk, editors, *Conformal Prediction for Reliable Machine Learning*, pages 115–130. Morgan Kaufmann, Boston, 2014.

[13] Konstantinos Benidis, Syama Sundar Rangapuram, Valentin Flunkert, Yuyang Wang, Danielle Maddix, Caner Turkmen, Jan Gasthaus, Michael Bohlke-Schneider, David Salinas, Lorenzo Stella, François-Xavier Aubet, Laurent Callot, and Tim Januschowski. Deep Learning for Time Series Forecasting: Tutorial and Literature Survey. *ACM Comput. Surv.*, 55(6), dec 2022.

[14] Jennifer Black and Philip Koopman. System Safety as an Emergent Property in Composite Systems. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 369–378. IEEE, 2009.

[15] Daniel Bogdoll, Maximilian Nitsche, and J. Marius Zöllner. Anomaly Detection in Autonomous Driving: A Survey. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 4488–4499, New York, NY, USA, June 2022. IEEE.

[16] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[17] Markus Borg, Jens Henriksson, Kasper Socha, Olof Lennartsson, Elias Sonnsjö Lönegren, Thanh Bui, Piotr Tomaszewski, Sankar Raman Sathyamoorthy, Sebastian Brink, and Mahshid Helali Moghadam. Ergo, SMIRK is safe: A Safety Case for a Machine Learning Component in a Pedestrian Automatic Emergency Brake System. *Software Quality Journal*, 31(2):335–403, 2023.

[18] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[19] Lawrence D. Brown, T. Tony Cai, and Anirban DasGupta. Interval Estimation for a Binomial Proportion. *Statistical Science*, 16(2):101 – 133, 2001.

[20] Anthony Bucci and Jordan B. Pollack. On Identifying Global Optima in Cooperative Coevolution. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, GECCO '05, page 539–544, New York, NY, USA, 2005. Association for Computing Machinery.

[21] Carla Contributors. CARLA Python API Documentation. `https://carla.readthedocs.io/en/0.9.14/python_api/#carlawalker`, 2022. [Online; accessed 25-January-2022].

[22] Cristian Challu, Kin G. Olivares, Boris N. Oreshkin, Federico Garza Ramirez, Max Mergenthaler Canseco, and Artur Dubrawski. Nhits: Neural hierarchical interpolation for time series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(6):6989–6997, Jun. 2023.

[23] Junzhou Chen and Sidi Lu. An advanced driving agent with the multimodal large language model for autonomous vehicles. In *2024 IEEE International Conference on Mobility, Operations, Services and Technologies (MOST)*, pages 1–11, 2024.

[24] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An Analyzer for Non-linear Hybrid Systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 258–263, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[25] Pranav Singh Chib and Pravendra Singh. Recent advancements in end-to-end autonomous driving using deep learning: A survey. *IEEE Transactions on Intelligent Vehicles*, 9(1):103–118, 2024.

[26] Darren Cofer, Isaac Amundson, Ramachandra Sattigeri, Arjun Passi, Christopher Boggs, Eric Smith, Limei Gilham, Taejoon Byun, and Sanjai Rayadurgam. Run-Time Assurance for Learning-Enabled Systems. In Ritchie Lee, Susmit Jha, Anastasia Mavridou, and Dimitra Giannakopoulou, editors, *NASA Formal Methods*, pages 361–368, Cham, 2020. Springer International Publishing.

[27] European Commission. Regulation of the european parliament and of the council laying down harmonized rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206`, 2021. European Union Proposal for a Regulation, COM/2021/206 final.

[28] On-Road Automated Driving (ORAD) Committee. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, apr 2021.

[29] Nello Cristianini and John Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[30] Can Cui, Yunsheng Ma, Xu Cao, Wenqian Ye, Yang Zhou, Kaizhao Liang, Jintai Chen, Juanwu Lu, Zichong Yang, Kuei-Da Liao, Tianren Gao, Erlong Li, Kun Tang, Zhipeng Cao, Tong Zhou, Ao Liu, Xinrui Yan, Shuqi Mei, Jianguo Cao, Ziran Wang, and Chao Zheng. A survey on multimodal large language models for autonomous driving. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) Workshops*, pages 958–979, January 2024.

[31] Zeyu Dong, Yimin Zhu, Yansong Li, Kevin Mahon, and Yu Sun. Generalizing End-To-End Autonomous Driving In Real-World Environments Using Zero-Shot LLMs. In *8th Annual Conference on Robot Learning*, 2024.

[32] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, 2021.

[33] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[34] Anam Farrukh and Richard West. FlyOS: Rethinking Integrated Modular Avionics for Autonomous Multicopters. *Real-Time Systems*, 59(2):256–301, 2023.

[35] Tadayoshi Fushiki. Estimation of Prediction Error by using K-fold Cross-Validation. *Statistics and Computing*, 21:137–146, 2011.

[36] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1050–1059, New York, NY, USA, 20–22 Jun 2016. PMLR.

[37] Sicun Gao, Soonho Kong, and Edmund M. Clarke. dReal: An SMT Solver for Nonlinear Theories over the Reals. In Maria Paola Bonacina, editor, *Automated Deduction – CADE-24*, pages 208–214, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[38] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics*. Springer Publishing Company, Incorporated, 2nd edition, 2010.

[39] Ionel Gog, Sukrit Kalra, Peter Schafhalter, Matthew A. Wright, Joseph E. Gonzalez, and Ion Stoica. Pylot: A Modular Platform for Exploring Latency-Accuracy Tradeoffs in Autonomous Vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8806–8813, 2021.

[40] Aaron Gordon. Cruise Self-Driving License Revoked After It Withheld Pedestrian Injury Footage, DMV Says. *VICE News*, 2023. `https://tinyurl.com/2rcaywkw` [Online; accessed 17-September-2024].

[41] Ruben Grewal, Paolo Tonella, and Andrea Stocco. Predicting Safety Misbehaviours in Autonomous Driving Systems using Uncertainty Quantification. In *Proceedings of 17th IEEE International Conference on Software Testing, Verification and Validation*, ICST '24, page 12 pages. IEEE, 2024.

[42] Stephen Haben, Marcus Voss, and William Holderbaum. *Time Series Forecasting: Core Concepts and Definitions*, pages 55–66. Springer International Publishing, Cham, 2023.

[43] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. Efficient Online Testing for DNN-Enabled Systems Using Surrogate-Assisted and Many-Objective Optimization. In *Proceedings of the 44th International Conference on Software Engineering*, ICSE '22, page 811–822, New York, NY, USA, 2022. Association for Computing Machinery.

[44] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. Can Offline Testing of Deep Neural Networks Replace Their Online Testing? *Empirical Software Engineering*, 26(5):90, 2021.

[45] Franz Hell, Gereon Hinz, Feng Liu, Sakshi Goyal, Ke Pei, Tetiana Lytvynenko, Alois Knoll, and Chen Yiqiang. Monitoring Perception Reliability in Autonomous Driving: Distributional Shift Detection for Estimating the Impact of Input Data on Prediction Accuracy. In *Proceedings of the 5th ACM Computer Science in Cars Symposium*, CSCS '21, New York, NY, USA, 2021. Association for Computing Machinery.

[46] Dan Hendrycks and Kevin Gimpel. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, online, 2017. OpenReview.net.

[47] Jens Henriksson, Christian Berger, Markus Borg, Lars Tornberg, Cristofer Englund, Sankar Raman Sathyamoorthy, and Stig Ursing. Towards Structured Evaluation of Deep Neural Network Supervisors. In *2019 IEEE International Conference On Artificial Intelligence Testing (AITest)*, pages 27–34, New York, NY, USA, April 2019. IEEE.

[48] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS quarterly*, 28(1):75–105, 2004.

[49] Julia Hoffman and Dev Metha. Artificial Intelligence: In-depth Market Analysis Market Insights Report. Technical report, Statista, 2023.

[50] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability. *Computer Science Review*, 37:100270, 2020.

[51] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. DeepCrime: Mutation Testing of Deep Learning Systems based on Real Faults. In *Proceedings of the 30th*

ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2021, page 67–78, New York, NY, USA, 2021. Association for Computing Machinery.

[52] Manzoor Hussain, Nazakat Ali, and Jang-Eui Hong. DeepGuard: A Framework for Safeguarding Autonomous Driving Systems from Inconsistent Behaviour. *Automated Software Engineering*, 29(1):1, 2022.

[53] Indy autonomous challenge. `https://www.indyautonomouschallenge.com/`, 2024. [Online; accessed 22-March-2024].

[54] IEC. 61025 Fault Tree Analysis (FTA). Technical report, IEC, 2006.

[55] Hisao Ishibuchi, Hiroyuki Masuda, Yuki Tanigaki, and Yusuke Nojima. Review of Coevolutionary Developments of Evolutionary Multi-objective and Many-objective Algorithms and Test Problems. In *2014 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, pages 178–184, 2014.

[56] Chris Isidore. Tesla Recalls Nearly All 2 Million of Its Vehicles on US Roads. *CNN*, 2023. `https://www.cnn.com/2023/12/13/tech/tesla-recall-autopilot/` [Online; accessed 17-September-2024].

[57] ISO. Road vehicles — Functional safety. Standard, International Organization for Standardization, Geneva, CH, March 2018.

[58] ISO. Road vehicles — Safety of the intended functionality. Standard, International Organization for Standardization, Geneva, CH, March 2022.

[59] ISO/IEC JTC 1/SC 42. `https://www.iso.org/committee/6794475.html`. [Online; accessed 17-September-2024].

[60] ISO/IEC/IEEE. Systems and software engineering – Life cycle processes – Requirements engineering. *ISO/IEC/IEEE 29148:2018(E)*, pages 1–104, 2018.

[61] Radoslav Ivanov, James Weimer, Rajeev Alur, George J. Pappas, and Insup Lee. Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, DESTION '19, pages 169–178, New York, NY, USA, apr 2019. ACM.

[62] Gunel Jahangirova, Andrea Stocco, and Paolo Tonella. Quality Metrics and Oracles for Autonomous Vehicles Testing. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pages 194–204, 2021.

[63] Tim Januschowski, Jan Gasthaus, Yuyang Wang, David Salinas, Valentin Flunkert, Michael Bohlke-Schneider, and Laurent Callot. Criteria for Classifying Forecasting Methods. *International Journal of Forecasting*, 36(1):167–177, 2020. M4 Competition.

[64] Kyle D. Julian, Mykel J. Kochenderfer, and Michael P. Owen. Deep Neural Network Compression for Aircraft Collision Avoidance Systems. *CoRR*, abs/1810.04240, 2018.

[65] Chanyoung Jung, Andrea Finazzi, Hyunki Seong, Daegyu Lee, Seungwook Lee, Bosung Kim, Gyuri Gang, Seungil Han, and David Hyunchul Shim. An Autonomous System for Head-to-Head Race: Design, Implementation and Analysis; Team KAIST at the Indy Autonomous Challenge. `https://arxiv.org/abs/2303.09463`, 2023.

[66] Guy Katz, Clark Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. In Rupak Majumdar and Viktor Kunčak, editors, *Computer Aided Verification*, pages 97–117, Cham, 2017. Springer International Publishing.

[67] Sydney M. Katz, Anthony Corso, Sandeep Chinchali, Amine Elhafsi, Apoorva Sharma, Mykel J. Kochenderfer, and Marco Pavone. NASA ULI X-Plane Simulator. `https://github.com/StanfordASL/NASA_ULI_Xplane_Simulator`, 2021. [Online; accessed 23-September-2024].

[68] Alex Kendall and Yarin Gal. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision? In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, New York, NY, USA, 2017. Curran Associates, Inc.

[69] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. `https://arxiv.org/abs/1412.6980`, 2017.

[70] George Klees, Andrew Ruef, Benji Cooper, Shiyi Wei, and Michael Hicks. Evaluating fuzz testing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 2123–2138, New York, NY, USA, 2018. Association for Computing Machinery.

[71] Stephan Kolassa. Sometimes It's Better to Be Simple than Correct. *Foresight: The International Journal of Applied Forecasting*, (40):20 – 26, 2016.

[72] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30, New York, NY, USA, 2017. Curran Associates, Inc.

[73] Nancy G. Leveson. *Engineering a Safer World*. The MIT Press, January 2012.

[74] Bryan Lim, Sercan Ö. Arık, Nicolas Loeff, and Tomas Pfister. Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[75] Bryan Lim and Stefan Zohren. Time-series forecasting with deep learning: a survey. *Philosophical Transactions of the Royal Society A*, 379(2194):20200209, 2021.

[76] Felipe Tomazelli Lima and Vinicius M.A. Souza. A large comparison of normalization methods on time series. *Big Data Research*, 34:100407, 2023.

[77] Wei-Yin Loh. Classification and Regression Trees. *WIREs Data Mining and Knowledge Discovery*, 1(1):14–23, 2011.

[78] Alessio Lomuscio and Lalit Maganti. An Approach to Reachability Analysis for Feed-forward Relu Neural Networks. *arXiv preprint arXiv:1706.07351*, 2017.

[79] Sean Luke. *Essentials of Metaheuristics*. Lulu, second edition, 2013. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

[80] Björn Lundgren. Safety Requirements vs. Crashing Ethically: What Matters Most for Policies on Autonomous Vehicles. *AI & SOCIETY*, 36(2):405–415, 2021.

[81] Yuan Luo, Ya Xiao, Long Cheng, Guojun Peng, and Danfeng (Daphne) Yao. Deep Learning-based Anomaly Detection in Cyber-physical Systems: Progress and Opportunities. *ACM Comput. Surv.*, 54(5), may 2021.

[82] R. J. Beckman M. D. Mckay and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output From a Computer Code. *Technometrics*, 42(1):55–61, 2000.

[83] Xiaoliang Ma, Xiaodong Li, Qingfu Zhang, Ke Tang, Zhengping Liang, Weixin Xie, and Zexuan Zhu. A Survey on Cooperative Co-Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 23(3):421–441, 2019.

[84] David J. C. MacKay. A Practical Bayesian Framework for Backpropagation Networks. *Neural Computation*, 4(3):448–472, 05 1992.

[85] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. The M4 Competition: 100,000 Time Series and 61 Forecasting Methods. *International Journal of Forecasting*, 36(1):54–74, 2020. M4 Competition.

[86] Spyros Makridakis, Evangelos Spiliotis, Assimakopoulos Vassilios, Artemios-Anargyros Semenoglou, Gary Mulder, and Konstantinos Nikolopoulos. Statistical, Machine Learning and Deep Learning Forecasting Methods: Comparisons and Ways Forward. *Journal of the Operational Research Society*, 74(3):840–859, 2023.

[87] Henry B Mann and Donald R Whitney. On a Test of Whether One of Two Random Variables is Stochastically Larger than the Other. *The annals of mathematical statistics*, pages 50–60, 1947.

[88] Jiageng Mao, Junjie Ye, Yuxi Qian, Marco Pavone, and Yue Wang. A Language Agent for Autonomous Driving. `https://arxiv.org/abs/2311.10813`, 2024.

[89] D. Meltz and H. Guterman. Functional Safety Verification for Autonomous UGVs-Methodology Presentation and Implementation on a Full-Scale System. *IEEE Transactions on Intelligent Vehicles*, 4(3):472–485, 2019.

[90] Michiel M. Minderhoud and Piet H.L. Bovy. Extended Time-To-Collision Measures for Road Traffic Safety Assessment. *Accident Analysis & Prevention*, 33(1):89–97, 2001.

[91] Seyedali Mirjalili. Genetic algorithm. *Evolutionary Algorithms and Neural Networks: Theory and Applications*, pages 43–55, 2019.

[92] Sina Mohseni, Haotao Wang, Chaowei Xiao, Zhiding Yu, Zhangyang Wang, and Jay Yadawa. Taxonomy of Machine Learning Safety: A Survey and Primer. *ACM Comput. Surv.*, 55(8), dec 2022.

[93] National Highway and Transportation Safety Authority (NHTSA). Federal Automated Vehicles Policy. `https://www.transportation.gov/sites/dot.gov/files/docs/AV%20policy%20guidance%20PDF.pdf`, 2016. [Online; accessed 23-September-2024].

[94] Mozhgan Navardi, Aidin Shiri, Edward Humes, Nicholas R. Waytowich, and Tinoosh Mohsenin. An Optimization Framework for Efficient Vision-Based Autonomous Drone Navigation. In *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, pages 304–307, 2022.

[95] Nam Nguyen and Brian Quanz. Temporal Latent Auto-Encoder: A Method for Probabilistic Multivariate Time Series Forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9117–9125, May 2021.

[96] NHTSA. Summary Report: Standing General Order on Crash Reporting for Automated Driving Systems. *Department of Transportation. DOT HS 813 324*, 2022.

[97] Justin Norden, Matthew O'Kelly, and Aman Sinha. Efficient Black-box Assessment of Autonomous Vehicle Safety. `https://ui.adsabs.harvard.edu/abs/2019arXiv191203618N`, 2019.

[98] Majority Staff of The Committee on Transportation and Infrastructure. Final Committee Report: The Design, Developement & Certification of The Boeing 737 MAX. *The US House Committee on Transportation and Infrastructure*, 2022. [Online; accessed 20-September-2024].

[99] Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, D. Sculley, Sebastian Nowozin, Joshua Dillon, Balaji Lakshminarayanan, and Jasper Snoek. Can You Trust Your Model's Uncertainty? Evaluating Predictive Uncertainty under Dataset Shift. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32, New York, NY, USA, 2019. Curran Associates, Inc.

[100] Liviu Panait. Theoretical Convergence Guarantees for Cooperative Coevolutionary Algorithms. *Evolutionary computation*, 18(4):581–615, 2010.

[101] Liviu Panait, Sean Luke, and Joseph F Harrison. Archive-based Cooperative Coevolutionary Algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 345–352, 2006.

[102] F. Paredes-Vallés, J. J. Hagenaars, J. Dupeyroux, S. Stroobants, Y. Xu, and G. C. H. E. de Croon. Fully Neuromorphic Vision and Control for Autonomous Drone Flight. *Science Robotics*, 9(90):eadi0591, 2024.

[103] Corina S. Păsăreanu, Ravi Mangal, Divya Gopinath, Sinem Getir Yaman, Calum Imrie, Radu Calinescu, and Huafeng Yu. Closed-Loop Analysis of Vision-Based Autonomous Systems: A Case Study. In Constantin Enea and Akash Lal, editors, *Computer Aided Verification*, pages 289–303, Cham, 2023. Springer Nature Switzerland.

[104] Ken Peffers, Tuure Tuunanen, Marcus A. Rothenberger, and Samir Chatterjee. A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, 24(3):45–77, 2007.

[105] Marco Peixeiro. *Time Series Forecasting in Python*. Simon and Schuster, New York City, NY, USA, 2022.

[106] Mitchell A. Potter and Kenneth A. De Jong. A Cooperative Coevolutionary Approach to Function Optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature — PPSN III*, pages 249–257, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[107] Stephen Prajna and Ali Jadbabaie. Safety Verification of Hybrid Systems Using Barrier Certificates. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 477–492, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[108] Mxnet predict mode. `https://mxnet.apache.org/versions/1.6/api/python/docs/api/autograd/index.html#mxnet.autograd.predict_mode`, 2024. [Online; accessed 23-September-2024].

[109] Pytorch inference mode. `https://pytorch.org/docs/stable/generated/torch.autograd.grad_mode.inference_mode.html`, 2024. [Online; accessed 23-September-2024].

[110] Quazi Marufur Rahman, Peter Corke, and Feras Dayoub. Run-Time Monitoring of Machine Learning for Robotic Perception: A Survey of Emerging Trends. *IEEE Access*, 9:20067–20075, 2021.

[111] Kashif Rasul, Calvin Seward, Ingmar Schuster, and Roland Vollgraf. Autoregressive Denoising Diffusion Models for Multivariate Probabilistic Time Series Forecasting. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 8857–8868. PMLR, 18–24 Jul 2021.

[112] Kashif Rasul, Abdul-Saboor Sheikh, Ingmar Schuster, Urs Bergmann, and Roland Vollgraf. Multivariate Probabilistic Time Series Forecasting via Conditioned Normalizing Flows. https://arxiv.org/abs/2002.06103, 2021.

[113] Elizabeth M Renieris, David Kiron, and Steven Mills. Building Robust RAI Programs as Third-party AI Tools Proliferate. *MIT Sloan Management Review*, 2023.

[114] Quelita A. D. S. Ribeiro, Moniky Ribeiro, and Jaelson Castro. Requirements Engineering for Autonomous Vehicles: A Systematic Literature Review. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 1299–1308, New York, NY, USA, 2022. Association for Computing Machinery.

[115] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. Testing Machine Learning based Systems: A Systematic Mapping. *Empirical Software Engineering*, 25(6):5193–5254, 2020.

[116] Vincenzo Riccio and Paolo Tonella. Model-based Exploration of the Frontier of Behaviours for Deep Learning System Testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 876–888, 2020.

[117] Wolfgang Roth, Günther Schindler, Bernhard Klein, Robert Peharz, Sebastian Tschiatschek, Holger Fröning, Franz Pernkopf, and Zoubin Ghahramani. Resource-Efficient Neural Networks for Embedded Systems. *Journal of Machine Learning Research*, 25(50):1–51, 2024.

[118] David Salinas, Valentin Flunkert, Jan Gasthaus, and Tim Januschowski. DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[119] Paul M Salmon, Neville A Stanton, Guy H Walker, Adam Hulme, Natassia Goode, Jason Thompson, and Gemma JM Read. *Handbook of Systems Thinking Methods*. CRC Press, 2022.

[120] S.M. Sanchez. Work Smarter, Not Harder: Guidelines for Designing Simulation Experiments. In *Proceedings of the Winter Simulation Conference, 2005.*, pages 14 pp.–, 2005.

[121] Sepehr Sharifi and Donghwan Shin. MLCSHE Replication Package. *figshare*, 2023. URL: https://doi.org/10.6084/m9.figshare.21965021.

[122] Sepehr Sharifi, Donghwan Shin, Lionel C. Briand, and Nathan Aschbacher. Identifying the Hazard Boundary of ML-Enabled Autonomous Systems Using Cooperative Coevolutionary Search. *IEEE Transactions on Software Engineering*, 49(12):5120–5138, 2023.

[123] Sepehr Sharifi, Andrea Stocco, and Lionel Briand. Safety Metric Forecaster Replication Package. https://figshare.com/articles/journal_contribution/SMForecaster_Replication_Package/25868947, 5 2024.

[124] Sepehr Sharifi, Andrea Stocco, and Lionel C. Briand. System safety monitoring of learned components using temporal metric forecasting. `https://arxiv.org/abs/2405.13254`, 2024.

[125] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015.

[126] Aman Sinha, Matthew O' Kelly, Russ Tedrake, and John C Duchi. Neural Bridge Sampling for Evaluating Safety-Critical Autonomous Systems. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6402–6416. Curran Associates, Inc., 2020.

[127] Dag I. K. Sjøberg and Gunnar Rye Bergersen. Construct Validity in Software Engineering. *IEEE Transactions on Software Engineering*, 49(3):1374–1396, 2023.

[128] Mark A. Skoog, Loyd R. Hook, and Wes Ryan. Leveraging ASTM Industry Standard F3269-17 for Providing Safe Operations of a Highly Autonomous Aircraft. In *2020 IEEE Aerospace Conference*, pages 1–7, Big Sky, Montana, USA, 2020. Institute of Electrical and Electronics Engineers (IEEE).

[129] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[130] British Standard and BS IEC61882. Hazard and Operability Studies (HAZOP Studies)-Application Guide. *International Electrotechnical Commission*, 2001.

[131] Andrea Stocco, Paulo J. Nunes, Marcelo D'Amorim, and Paolo Tonella. ThirdEye: Attention Maps for Safe Autonomous Driving Systems. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA, 2023. Association for Computing Machinery.

[132] Andrea Stocco and Paolo Tonella. Confidence-driven Weighted Retraining for Predicting Safety-critical Failures in Autonomous Driving Systems. *Journal of Software: Evolution and Process*, 34(10):e2386, 2022.

[133] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. Misbehaviour Prediction for Autonomous Driving Systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 359–371, New York, NY, USA, 2020. Association for Computing Machinery.

[134] H. Tran, P. Musau, D. Manzanas Lopez, X. Yang, L. V. Nguyen, W. Xiang, and T. T. Johnson. Parallelizable Reachability Analysis Algorithms for Feed-Forward Neural Networks. In *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pages 51–60, 2019.

[135] Hoang-Dung Tran, Diago Manzanas Lopez, Patrick Musau, Xiaodong Yang, Luan Viet Nguyen, Weiming Xiang, and Taylor T. Johnson. Star-Based Reachability Analysis of Deep Neural Networks. In Maurice H. ter Beek, Annabelle McIver, and José N. Oliveira, editors, *Formal Methods – The Next 30 Years*, pages 670–686, Cham, 2019. Springer International Publishing.

[136] Cumhur Erkan Tuncali, James Kapinski, Hisahiro Ito, and Jyotirmoy V Deshmukh. INVITED: Reasoning about Safety of Learning-Enabled Components in Autonomous Cyber-physical Systems. In *2018 55TH ACM/ESDA/IEEE DESIGN AUTOMATION CONFERENCE (DAC)*, San Francisco, CA, 2018. IEEE; ACM; ESDA.

[137] Udacity's self-driving car simulator. `https://github.com/udacity/self-driving-car-sim`, 2022. [Online; accessed 27-September-2024].

[138] András Vargha and Harold D. Delaney. A Critique and Improvement of the CL Common Language Effect Size Statistics of McGraw and Wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[139] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[140] Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. Dissector: Input Validation for Deep Learning Applications by Crossing-layer Dissection. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, page 727–738, New York, NY, USA, 2020. Association for Computing Machinery.

[141] Ruofeng Wen, Kari Torkkola, Balakrishnan (Murali) Narayanaswamy, and Dhruv Madeka. A Multi-horizon Quantile Recurrent Forecaster. In *NeurIPS 2017*, 2017.

[142] Darrell Whitley. *Next Generation Genetic Algorithms: A User's Guide and Tutorial*, pages 245–274. Springer International Publishing, Cham, 2019.

[143] Wikipedia contributors. Regulation of Self-driving Cars — Wikipedia, The Free Encyclopedia. `https://en.wikipedia.org/w/index.php?title=Regulation_of_self-driving_cars&oldid=1245156164`, 2024. [Online; accessed 23-September-2024].

[144] D Randall Wilson and Tony R Martinez. Improved Heterogeneous Distance Functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[145] Hyrum K. Wright, Miryung Kim, and Dewayne E. Perry. Validity Concerns in Software Engineering Research. In *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, page 411–414, New York, NY, USA, 2010. Association for Computing Machinery.

[146] X-Plane Core Team. X-Plane 11 Flight Simulator Software, 2024.

[147] Yan Xiao, Ivan Beschastnikh, David S. Rosenblum, Changsheng Sun, Sebastian Elbaum, Yun Lin, and Jin Song Dong. Self-Checking Deep Neural Networks in Deployment. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 372–384, New York, NY, US, 2021. IEEE.

[148] James Yae. Unintended Look-ahead Bias in Out-of-Sample Forecasting. *Applied Economics Letters*, 0(0):1–5, 2022.

[149] Zhenyu Yang, Ke Tang, and Xin Yao. Large Scale Evolutionary Optimization using Cooperative Coevolution. *Information Sciences*, 178(15):2985–2999, 2008. Nature Inspired Problem-Solving.

[150] Jia-Yu Yao, Kun-Peng Ning, Zhen-Hui Liu, Mu-Nan Ning, Yu-Yang Liu, and Li Yuan. LLM Lies: Hallucinations are not Bugs, but Features as Adversarial Examples. *ArXiv e-prints*, 2024.

[151] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2022.

[152] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE 2018, pages 132–142, New York, NY, USA, 2018. ACM.

[153] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ASE '18, page 132–142, New York, NY, USA, 2018. Association for Computing Machinery.

[154] Qianqian Zhang, Haifeng Wang, Hongya Lu, Daehan Won, and Sang Won Yoon. Medical Image Synthesis with Generative Adversarial Networks for Tissue Recognition. In *2018 IEEE International Conference on Healthcare Informatics (ICHI)*. IEEE, 2018.

[155] Juan Zhao, QiPing Feng, Patrick Wu, Roxana A. Lupu, Russell A. Wilke, Quinn S. Wells, Joshua C. Denny, and Wei-Qi Wei. Learning from Longitudinal Data in Electronic Health Record and Genetic Data to Improve Cardiovascular Event Prediction. *Scientific Reports*, 9(1):717, 2019.

[156] Xin Zhou, Yuqin Jin, He Zhang, Shanshan Li, and Xin Huang. A Map of Threats to Validity of Systematic Literature Reviews in Software Engineering. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*, pages 153–160, 2016.

[157] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, and Ramesh S. SMARLA: A Safety Monitoring Approach for Deep Reinforcement Learning Agents. `https://arxiv.org/abs/2308.02594`, 2024.