



Titre: Towards Reliable, Production-Ready Deep Learning and Deep
Title: Reinforcement Learning Systems

Auteur: Ahmed Haj Yahmed
Author:

Date: 2023

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Haj Yahmed, A. (2023). Towards Reliable, Production-Ready Deep Learning and
Citation: Deep Reinforcement Learning Systems [Mémoire de maîtrise, Polytechnique
Montréal]. PolyPublie. <https://publications.polymtl.ca/57003/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/57003/>
PolyPublie URL:

**Directeurs de
recherche:** Foutse Khomh
Advisors:

Programme: Génie informatique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Towards Reliable, Production-Ready Deep Learning and Deep Reinforcement
Learning systems**

AHMED HAJ YAHMED

Département de génie informatique et génie logiciel

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie informatique

Décembre 2023

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Towards Reliable, Production-Ready Deep Learning and Deep Reinforcement
Learning systems**

présenté par **Ahmed HAJ YAHMED**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Mohammad HAMDAQA, président

Foutse KHOMH, membre et directeur de recherche

Sarath Chandar ANBIL PARTHIPAN, membre

DEDICATION

To my family and friends ...

ACKNOWLEDGEMENTS

In the scientific labyrinth, gratitude blooms like rare flowers. I would like to extend my sincere gratitude to Professor Foutse Khomh for providing me with the opportunity to delve into the realm of research, offering patient guidance throughout my Master's program, and providing invaluable feedback and support that ultimately led to the successful completion of my thesis. I would like to express my gratitude to my mentor, Dr. Housseem Ben Braiek, for his invaluable guidance and contributions, which have greatly enhanced my academic pursuits. I want to also thank Prof. Sonia Bouzidi and Dr. Rania Zaatour for their constant support, wise counsel, and important assistance throughout my very first research project. A heartfelt recognition is reserved for Rached Bouchoucha, my partner in crime. Thank you for being a great friend and making sure we survived this academic rollercoaster without losing our minds—or at least not all of it. I would like to express my gratitude to Dr. Amin Nikanjam and Prof. Heng Li for their valuable support and critical feedback. I would like to extend my sincere appreciation to Prof. Mohammad Hamdaqa and Prof. Sarath Chandar for their insightful assessment of my thesis. Last but definitely not least, a massive thank you to my parents, siblings, future wife, and friend squad for being the real MVPs. Your support kept me sane through the good, the bad, and the thesis-induced madness. I promise the next adventure will involve less late-night typing and more normal human activities.

RÉSUMÉ

Nous sommes au bord d'une nouvelle révolution technologique alimentée par l'intelligence artificielle (IA) qui remodèle les industries et modifie notre vie quotidienne. Au cours des dernières années, des progrès notables ont été réalisés dans les domaines de l'apprentissage profond (DL) et du apprentissage par renforcement profond (DRL), les sous-domaines les plus recherchés en matière d'IA. Ces progrès ont conduit à des réalisations importantes dans plusieurs domaines tels que la robotique, la médecine et les transports. En outre, l'accessibilité d'une puissance de calcul abordable et l'expansion rapide des données sont des facteurs clés pour le développement de modèles DL et DRL de pointe. Toutefois, générer du profit grâce à l'apprentissage automatique (ML) exige plus que simplement de développer le meilleur modèle ML; il exige l'exploitation d'un système ML holistique qui fonctionne de manière fiable et s'adapte aux changements de la dynamique de l'entreprise. Cette nécessité d'exploitation a mis en évidence la nécessité de mettre en place une discipline axée sur l'acheminement de la ML à la production. Opérations d'apprentissage automatique (MLOps) a attiré un intérêt considérable dans le domaine de l'ingénierie ML en raison de ses améliorations notables en matière de qualité du logiciel et de temps de mise sur le marché. MLOps est une méthodologie d'ingénierie de ML qui formalisent et automatisent les différentes étapes du cycle de vie de ML, avec l'objectif d'unifier le développement et les opérations de ML.

Les rapports actuels du secteur mettent l'accent sur les difficultés des entreprises de ML, en particulier celles qui exploitent DL et DRL, à appliquer les principes MLOps et à passer d'une solution de preuve de concept à un système ML fiable et prêt pour la production. Cela est attribué à l'opérationnalisation actuelle, lente, manuelle et difficile des systèmes ML, qui appelle à plus de dévouement et d'attention de la part du monde universitaire et de l'industrie. Plusieurs défis entravent la production transparente du ML dans des applications du monde réel, couvrant toutes les phases du cycle de vie de l'opérationnalisation du ML, y compris la validation, le déploiement et la surveillance des modèles. En effet, il existe un manque important d'outils, de lignes directrices et de recherches spécifiquement axés sur la phase de déploiement de DL et DRL. Les universités et les entreprises sont principalement concernées par la phase de développement des systèmes ML, avec une attention minimale sur le déploiement. La rareté des recherches sur les défis de déploiement spécifiques aux DRL a laissé les praticiens mal informés des problèmes potentiels qu'ils peuvent rencontrer lors du déploiement de systèmes basés sur les DRL. De plus, on s'appuie toujours sur des procédures traditionnelles, souvent inefficaces, pour évaluer la fiabilité des stratégies de compression post-entraînement, telles que la quantification. Dans les environnements de production réels,

ces techniques traditionnelles d'évaluation de la fiabilité affectent la validation du modèle, ce qui constitue un obstacle à l'évaluation précise du comportement du modèle. De plus, le phénomène de dérive des données, c'est-à-dire l'altération de la distribution des données au fil du temps dans les contextes opérationnels, continue de constituer un obstacle important à la surveillance du ML en production. Cela peut entraîner une détérioration notable des performances du modèle. Lorsque cette condition n'est pas rapidement identifiée et résolue, elle peut augmenter considérablement les coûts opérationnels et les temps d'arrêt. Compte tenu des défis susmentionnés, il est primordial d'appliquer davantage les principes MLOps relatifs à chaque phase d'opérationnalisation du ML et d'améliorer la fiabilité et la préparation à la production des systèmes DL et DRL.

Dans cette thèse, nous souhaitons contribuer à trois aspects clés couvrant des phases distinctes du cycle de vie de l'opérationnalisation du ML. Notre objectif est de relever les défis qui surviennent tout au long des phases de déploiement, de validation et de surveillance du modèle. Cette approche ciblée permet un examen plus approfondi des défis auxquels les praticiens du ML sont confrontés lors de la transition de la validation de principe à des systèmes fiables et prêts pour la production. Tout d'abord, nous abordons le déploiement du modèle. Plus précisément, nous contribuons à améliorer notre compréhension des défis pratiques rencontrés par les développeurs lors du déploiement de systèmes DRL en menant une étude empirique sur Stack Overflow (SO), le plus grand forum mondial de développeurs. Dans cette étude, nous avons examiné les articles de SO sur le déploiement de DRL, étudié l'état actuel et mis en évidence les défis associés au déploiement de systèmes DRL. Ensuite, nous examinons la fréquence et la difficulté de ces défis. Ensuite, nous avons examiné la fiabilité des techniques de quantification des réseaux neuronaux profonds (DNN) dans le but de découvrir l'impact réel de ces techniques dans les environnements de production. Nous proposons DiverGet, une approche de test basée sur la recherche pour évaluer l'impact réel des méthodes de quantification. DiverGet vise à détecter les entrées induisant des différences qui provoquent des désaccords comportementaux entre un DNN original et sa version quantifiée, permettant ainsi une évaluation complète de l'influence des approches de quantification. Enfin, notre troisième étude plonge dans le suivi des modèles, en se concentrant sur le phénomène de dérive des données dans les DRL (également appelée dérive environnementale). La dérive environnementale est susceptible de provoquer une dégradation des performances prédictives du système DRL en production. L'apprentissage continu (CL) est actuellement la solution incontournable pour guérir les systèmes DRL après des dérives environnementales. Cependant, en raison des inefficacités de la CL, telles que l'oubli catastrophique et la lente convergence, la CL peut entraîner des cycles de guérison longs, voire infructueux. Pour atténuer ce problème, nous avons introduit Dr.DRL, une nouvelle approche d'auto-guérison DRL qui intègre

un mécanisme d’oubli intentionnel. Dr.DRL est conçu pour donner systématiquement la priorité à l’adaptation des capacités de résolution de problèmes d’un agent DRL dans des contextes de production réels en présence de dérive environnementale. Notre approche réinitialise délibérément les comportements mineurs du système DRL afin de mettre l’accent sur l’adaptation du comportement majeur, améliorant ainsi la vitesse d’auto-guérison.

L’éventail des questions de recherche présentées dans cette étude couvre l’ensemble du cycle de vie de l’opérationnalisation du ML, abordant des défis allant de la validation du modèle à la surveillance du modèle. Notre étude vise à alléger le fardeau de l’opérationnalisation du ML pour les praticiens du ML et à servir de base sur laquelle la communauté du ML peut s’appuyer vers des systèmes DL et DRL plus fiables et prêts pour la production.

ABSTRACT

We stand on the verge of a new technological revolution powered by Artificial Intelligence (AI) that is reshaping industries and altering our daily lives. In recent years, there have been notable advancements in Deep Learning (DL) and Deep Reinforcement Learning (DRL), the most sought-after subfields of AI. These advancements have led to significant accomplishments in several domains like robotics, medicine, and transportation. Furthermore, the accessibility of affordable computational power and the rapid expansion of data are key factors for the development of cutting-edge DL and DRL models. However, producing profit through Machine Learning (ML) requires more than simply developing the finest ML model; it necessitates the operationalization of a holistic ML system that performs reliably and adapts to changes in the business’s dynamics. This need for operationalization has highlighted the necessity for a discipline focusing on delivering ML to production. Machine Learning Operations (MLOps) has attracted significant interest in the field of ML engineering owing to its notable improvements in software quality and time to market. MLOps is an ML engineering methodology that formalizes and automates the various stages of the ML lifecycle, with the objective of unifying ML development and operations.

Current industry reports emphasize the struggles of ML firms, especially those leveraging DL and DRL, in applying MLOps principles, and transitioning from a proof-of-concept solution to a reliable, production-ready ML system. This is attributed to the current slow, manual, and challenging operationalization of ML systems, which urges more dedication and attention from both academia and industry. Several challenges hinder the seamless productionization of ML in real-world applications, spanning all phases of the ML operationalization lifecycle, including model validation, deployment, and monitoring. Indeed, there is a significant lack of tools, guidelines, and research, that are specifically focused on the deployment phase of DL and DRL. Academia and business are primarily concerned with the development phase of ML systems, with minimal focus on deployment. The scarcity of research on DRL-specific deployment challenges has left practitioners uninformed of potential issues they may encounter while deploying DRL-based systems. Furthermore, there is still a reliance on traditional, often ineffective, procedures for assessing the reliability of post-training compression strategies, such as quantization. In real-world production settings, these traditional reliability assessment techniques affect model validation, posing a hurdle in accurately gauging the model’s behavior. In addition, the phenomenon of data drift, the alteration in the data distribution over time in operational settings, continues to pose a significant obstacle to in-production ML monitoring. This can lead to a notable deterioration in the model performance. When

this condition is not promptly identified and resolved, it can significantly increase operational costs and downtime. Considering the aforementioned challenges, it’s paramount to further apply MLOps principles relative to each ML operationalization phase and enhance the reliability and production-readiness of DL and DRL systems.

In this thesis, we are interested in contributing to three key aspects covering distinct phases within the ML operationalization lifecycle. Our aim is to address challenges that arise throughout the model deployment, validation, and monitoring phases. This targeted approach allows for a more in-depth examination of the challenges that ML practitioners face as they transition from proof-of-concept to reliable, production-ready systems. First, we tackle model deployment. More precisely, we contribute to improving our understanding of practical challenges encountered by developers when deploying DRL systems by conducting an empirical study on Stack Overflow (SO), the world’s largest forum for developers. In this study, we reviewed SO posts about DRL deployment, investigated the current state, and highlighted the challenges associated with deploying DRL systems. Then, we look into the frequency and difficulty of these challenges. Next, we examined the reliability of Deep Neural Network (DNN) quantization techniques with the aim of uncovering the real impact of these techniques in production settings. We propose DiverGet, a search-based testing approach to evaluate the real impact of quantization methods. DiverGet aims to detect difference-inducing inputs that cause behavioral disagreements between an original DNN and its quantized version, thereby enabling a comprehensive evaluation of the influence of quantization approaches. Finally, our third study dives into model monitoring, focusing on the phenomenon of data drifts in DRL (also known as environmental drift). Environmental drift is able to cause a degradation in the predictive performance of the DRL system in production. Continual learning (CL) is currently the go-to solution for healing DRL systems after environmental drifts. However, due to CL’s inefficiencies such as catastrophic forgetting and slow convergence, CL can result in long or even failed healing cycles. To alleviate this issue, we introduced Dr.DRL, a new DRL self-healing approach that incorporates an intentional forgetting mechanism. Dr.DRL is designed to systematically prioritize the adaptation of a DRL agent’s problem-solving abilities in real-world production settings in the presence of environmental drift. Our approach purposefully resets the DRL system’s minor behaviors in order to emphasize the adaptation of the major behavior, hence enhancing the speed of self-healing.

The range of research questions presented in this study spans the entire ML operationalization lifecycle, addressing challenges ranging from model validation to model monitoring. Our study aims to relieve the burden of ML operationalization for ML practitioners and serve as a foundation for the ML community to build on, toward more reliable and production-ready

DL and DRL systems.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	viii
TABLE OF CONTENTS	xi
LIST OF TABLES	xiv
LIST OF FIGURES	xv
LIST OF SYMBOLS AND ACRONYMS	xvi
LIST OF APPENDICES	xvii
CHAPTER 1 INTRODUCTION	1
1.1 Productionizing DL and DRL systems: A primer	1
1.2 Productionizing DL and DRL systems: Challenges and limitations	3
1.2.1 Deployment tooling, guidelines, and studies:	3
1.2.2 Quality assurance in model compression:	3
1.2.3 Model adaptation to changing environments:	4
1.3 Research Statement	4
1.4 Thesis Overview	5
1.5 Thesis contributions	6
1.6 Organisation of the thesis	7
CHAPTER 2 BACKGROUND	9
2.1 Deep Learning and Deep Reinforcement Learning	9
2.1.1 Deep Neural Networks	9
2.1.2 Deep Learning	11
2.1.3 Deep Reinforcement Learning	11
2.1.4 DRL development and deployment	12
2.2 MLOps: Towards production-ready ML systems	13

2.2.1	Quantization	14
2.2.2	Environment Drift and Continual Learning	15
2.2.3	Software Self-Healing and Intentional Forgetting	16
2.3	Software Testing Techniques in Machine Learning	16
2.3.1	DL Testing:	16
2.3.2	Metamorphic Testing:	17
2.3.3	Search-based Software Testing (SBST)	18
2.3.4	Differential Testing	19
2.4	Chapter Summary	19
CHAPTER 3 LITERATURE REVIEW		20
3.1	Studies on DL Test Case Generation	20
3.2	Studies on DL and DRL Deployment Challenges	21
3.3	Studies on DL and DRL Repair and Healing	22
3.4	Chapter Summary	23
CHAPTER 4 DEPLOYING DEEP REINFORCEMENT LEARNING SYSTEMS: A TAXONOMY OF CHALLENGES		24
4.1	Methodology	25
4.2	RQ1: Level of Interest	30
4.3	RQ2: Taxonomy of Challenges	31
4.3.1	Common Challenges to all platforms	33
4.3.2	Other Challenges in Browser	37
4.3.3	Other Challenges in Mobile/Embedded System	37
4.3.4	Other Challenges in Server/Cloud	38
4.3.5	Survey Results	38
4.4	RQ3: Analysis of Challenges	39
4.5	Implications	41
4.6	Threats to Validity	43
4.7	Chapter Summary	43
CHAPTER 5 DIVERGET: A SEARCH-BASED SOFTWARE TESTING APPROACH FOR DEEP NEURAL NETWORK QUANTIZATION ASSESSMENT		44
5.1	Methodology	46
5.1.1	Quantization Assessment via Systematic Detection of Behavioral Di- vergenences	46
5.1.2	Search for Transformations instead of Data	47

5.1.3	Domain-specific Metamorphic Relations	49
5.1.4	Search-based Approach for Data Transformation	52
5.2	Experiments	59
5.2.1	Experimental Setup	59
5.2.2	RQ1: The Effectiveness of DiverGet	62
5.2.3	RQ2: DiverGet Facing Multiple Settings	64
5.2.4	RQ3: DiverGet vs DiffChaser	69
5.3	Threats to Validity	72
5.4	Chapter Summary	75
CHAPTER 6 AN INTENTIONAL FORGETTING, SELF-HEALING APPROACH FOR DEEP REINFORCEMENT LEARNING SYSTEMS		
6.1	Motivation Example	76
6.2	Approach	79
6.2.1	Problem formulation	79
6.2.2	Intentional Forgetting for DRL Self-Healing Enhancement	80
6.2.3	<i>Dr. DRL</i> : The Self-healing Workflow	84
6.3	Evaluation	85
6.3.1	Experimental Subjects	86
6.3.2	Research Questions and Answers	89
6.4	Threats to Validity	95
6.5	Chapter Summary	95
CHAPTER 7 CONCLUSION		
7.1	Summary	97
7.2	Limitations and Future work	99
REFERENCES		102
APPENDICES		121

LIST OF TABLES

Table 4.1	DRL deployment challenges and their categories. “Description” gives an overview of the category; “S/C”, “M/E”, “G.E”, “B” and “A” show the % of each category in Server/Cloud, Mobile/Embedded, Game Engine, Browser, and All Platforms, respectively.	32
Table 4.2	Validation Survey Results	40
Table 4.3	The popularity/difficulty per topic.	40
Table 5.1	Presentation of the used HSI distortions	51
Table 5.2	Details of DNNs and datasets used to evaluate DiverGet. The Overall Accuracies (OAs) are taken from [1,2]	60
Table 5.3	Comparison of the number of DII found in the original test data and by RS using all seeds	63
Table 5.4	Comparison of the median values of DiR and VR between RS and DiverGet	64
Table 5.5	Comparison of the median values of DiR and VR when using DiverGet with f^{cov} and f^{div}	65
Table 5.6	Statistics over the pairwise distances of transformation vectors generated using f^{div} and f^{cov}	66
Table 5.7	Comparison of the median values of DiR and VR between PSO and GA as metaheuristics for DiverGet	67
Table 5.8	DiverGet vs DiffChaser	70
Table 5.9	Comparison of the median values of DiR and SR between DiffChaser, PSO-based DiverGet and GA-based DiverGet	71
Table 5.10	Comparison of the median values of FDI and FDI* between DiffChaser, PSO-based DiverGet and GA-based DiverGet	73
Table 6.1	Comparison of Healing Time and Fine-Tuning Episodes Decreases Using <i>Dr. DRL</i> over vanilla CL.	89
Table 6.2	Comparison of healed DRL Agent Reward Increases Using <i>Dr.DRL</i> over CL.	94

LIST OF FIGURES

Figure 4.1	Overview of our study’s methodology.	26
Figure 4.2	Number of Questions per Year.	33
Figure 4.3	Number of Question with No Accepted Answer.	33
Figure 4.4	Time Needed To Receive an Accepted Answer.	34
Figure 4.5	Bubble plot of topic popularity and difficulty.	42
Figure 5.1	The workflow of DiverGet: starting from a given HSI (or a batch of HSIs), our proposed framework creates a set of transformations vectors. These latter can be used, in a post-processing step, to generate difference-inducing HSIs.	48
Figure 5.2	Formatting all distortions into a transformation vector	53
Figure 5.3	Overview of the experiment design and evaluation metrics.	62
Figure 5.4	t-SNE visualization of transformation vectors induced by f^{div} and f^{cov}	66
Figure 5.5	Comparison of the median values of DiR between DiverGet implementations using QAT-based and PTQ-based quantization models. All results have a p-value $< 5.8 * 10^{-9}$ and an effect size > 0.84 (large).	68
Figure 5.6	Comparison of the median values of DiR between DiverGet implementations using different datasets. All results have a p-value $< 1.5 * 10^{-11}$ and an effect size > 0.89 (large).	69
Figure 6.1	(A) The DQN agent’s average reward per episode on MountainCar adaption curves: (a) shows the average reward curve when Dr. DRL is used, whereas (b) shows the average reward curve when CL is used. (B) Graphic representation of the MountainCar environment as explained in [3].	78
Figure 6.2	<i>Dr. DRL</i> : The Self-healing Workflow	84
Figure 6.3	Box Plots of Healing Time and Episode Fine-Tuning Using <i>Dr. DRL</i> and vanilla CL.	90
Figure 6.4	%AR distribution across the four identified groups by healing method and RL environment	92
Figure 6.5	A Comparative Analysis of Average Reward Increase in Non-Adaptable Environments Utilising Dr.DRL vs CL.	94

LIST OF SYMBOLS AND ACRONYMS

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
MLOps	Machine Learning Operations
DNN	Deep Neural Network
CL	Continuous Learning
NN	Neural Network
DQN	Deep Q-Network
PPO	Proximal Policy Optimization
SAC	Soft Actor-Critic
TF	TensorFlow
CT	Continuous Training
CI	Continuous Integration
CD	Continuous Delivery
SO	Stack Overflow
PTQ	Post Training Quantization
QAT	Quantization Aware Training

LIST OF APPENDICES

Appendix A	CO-AUTHORSHIP	121
------------	-------------------------	-----

CHAPTER 1 INTRODUCTION

In 1880, Thomas Edison patented the light bulb, a breakthrough invention that would forever change our world. Yet, despite this historic innovation, it wasn't until 1925 that half of the households in the United States were electrified [4]. This long timeline from discovery to broad adoption serves as a reminder of the challenges that are often encountered during the deployment of a ground-breaking technology.

Today, we are witnessing a new technological revolution, and at its heart lies Artificial Intelligence (AI), the ability of computers to acquire knowledge and engage in decision-making processes that were once exclusive to humans [5, 6]. Technical advances in recent years, particularly in the most sought-after subfields of AI; i.e., Deep Learning (DL), and Deep Reinforcement Learning (DRL), have resulted in a number of achievements in domains such as robotics, medicine, and transportation [7–11]. Just as electricity reshaped the twentieth century and the internet revolutionized the twenty-first, AI is projected to further transform our lives in the forthcoming years [12]. However, at present, the full-scale adoption and deployment of AI, especially DL and DRL, is in its initial stages. Just as Edison's invention took decades to light up houses across America, the process of integrating AI into our daily lives is long and incremental. The operationalization of reliable, production-ready ML systems is a slow, manual, and challenging process, urging increased efforts and attention from both academics and industry. Indeed, there is a notable scarcity of tools, guidelines, and studies dedicated to DL and DRL deployment. Furthermore, conventional, often ineffective, strategies continue to be employed to assess the reliability of post-training compression strategies, such as quantization [13]. In addition, data drift remains a critical challenge for in-production ML systems, resulting in substantial maintenance costs and downtime.

This thesis focuses on three critical aspects that encompass different stages of the ML operationalization lifecycle. Our goal is to tackle challenges that emerge throughout the model deployment, validation, and monitoring stages. This focused approach allows for a more in-depth analysis of the issues that ML practitioners confront as they go from proof-of-concept to reliable, production-ready solutions.

1.1 Productionizing DL and DRL systems: A primer

The availability of low-cost computing power and the exponential growth of data has sparked significant advancements in AI, particularly allowing Deep Learning (DL) and Deep Rein-

forcement Learning (DRL) to thrive.

Deep Neural Networks (DNNs) are the backbone component of DL [14] and DRL [15]. In simple terms, DNNs consist of multiple layers of interconnected neurons, allowing them to learn increasingly abstract representations of input data. This ability has led to remarkable achievements in a variety of applications, including image recognition [16], natural language processing [17], and speech recognition [18]. A DL system is any software system having at least one DNN component [19]. DL systems vary from being designed purely through arrays of DNN components (e.g., object detection systems) or a combination of DNN components and traditional software components (e.g., recommendation systems). Unlike typical software systems, where the system behavior is governed by the business logic, DL systems, in addition to the business logic, learn their rules from data. DL algorithms are well recognized for their computational demands, needing high-performance hardware to ensure efficient execution. As a result, several post-training compression approaches, including quantization or pruning, have been introduced to mitigate model size for edge device deployment.

Reinforcement Learning (RL) is a subfield of ML that is concerned with autonomous learning and decision-making via interacting with an environment [15]. RL follows a trial-and-error paradigm where an agent interacts with its environment and learns to adapt its behavior to achieve a goal by observing the outcomes (i.e., rewards) of its actions [15, 20, 21]. DRL, the blend of DL and RL, has shown promising industrial achievements in recent years, such as AlphaStar [22], Netflix’s customized recommendation system [23], and Facebook ReAgent [24]. DNN-based function approximators allow the use of RL methods for complex state and action spaces, overcoming the limitations of traditional RL.

Achieving business value ML necessitates more than the mere development of an optimal ML model. It entails the construction of a unified ML system that operates reliably and adapts to changes in the business’s dynamics. This encompasses the steps of feature preparation, large-scale model training, model deployment, and model performance monitoring in production. Machine Learning Operations (MLOps) is an ML engineering methodology that formalizes and automates all of the above steps with the goal of blending ML development (ML) with ML operations (Ops). MLOps refers to a set of standardized procedures and technologies for the effective and reliable production of ML systems. Currently, many ML firms, particularly those leveraging DL and DRL, are having difficulty adopting MLOps principles and transitioning from a proof-of-concept solution to a production-ready ML system. In the next section, we’ll list some of these challenges and present the limitations of existing techniques.

1.2 Productionizing DL and DRL systems: Challenges and limitations

Despite the growing prominence of ML as a fundamental component of digital transformation, the ability to achieve successful deployments and maintain efficient operations remains a barrier to harnessing ML’s full potential. According to the Capgemini Research Institute’s survey ¹, only 53% of organizations surveyed have progressed beyond the initial stages of proof of concept. Also, 72% of organizations that initiated AI pilots before 2019 have yet to deploy the first application into production. Furthermore, Algorithmia’s 2020 ML enterprise study ² found that 55% of the organizations surveyed have not deployed a single ML model. Nowadays, the main challenge isn’t building an ML model, but rather building an integrated ML system that continuously operates in production. Listed below are three of the main challenges in productionizing ML systems that we address in this thesis as well as the limitations of existing approaches.

1.2.1 Deployment tooling, guidelines, and studies:

Challenges. Academia and industry are working together to assist researchers and practitioners in handling ML’s new challenges. However, they focus mostly on the development phase of ML systems, with little attention paid to deployment. Remarkably, certain emergent ML subfields, such as DRL, still lack deployment tools, guidelines, and studies. Indeed, DRL-specific deployment challenges remain unstudied, leaving practitioners unaware of the challenges they may encounter while deploying DRL-based systems [25].

Limitations of current practice. To the best of our knowledge, no study has yet tried to identify and understand the challenges practitioners face when deploying DRL-based systems. The current DRL deployment research focuses on addressing the problem from an academic perspective, often relying on systematic literature reviews [26]. Yet, these current studies may not adequately convey practitioners’ real-world deployment issues. To bridge the knowledge gap between research and practice, a more practitioner-oriented perspective is needed in DRL deployment research.

1.2.2 Quality assurance in model compression:

Challenges. After deployment, models often need to be transformed and exported to new infrastructures for inference. When deployed in edge devices, ML models need to be compressed, using techniques such as quantization, to fit these devices’ low computational power.

¹<https://www.capgemini.com/gb-en/insights/research-library/the-ai-powered-enterprise/>

²<https://www.strong.io/blog/state-of-enterprise-machine-learning-2020>

With the rising demand for post-training compression strategies, ensuring these strategies’ reliability is paramount.

Limitations of current practice. Currently, traditional techniques are used to assess post-training compression strategies such as quantization. More precisely, developers use the test dataset to measure to what extent the performance of the model is preserved after compression. However, the test dataset cannot represent the immensity of the test input space, and this can lead to the two model versions (i.e., the original and compressed model) having the same level of performance on test data yet triggering divergent behavior in real-world settings.

1.2.3 Model adaptation to changing environments:

Challenges. Monitoring the performance of models is an important task that guarantees their effectiveness over time. It entails continuous assessment, anomaly detection, and model fine-tuning to maintain optimal performance in dynamic environments. Environment drift is a phenomenon that causes a DRL agent’s performance to deteriorate over time. Environment drift refers to changes in the characteristics of the RL environment during a given time period when the DRL agent is being deployed. This phenomenon can dramatically raise operational expenses and thus must be addressed rapidly.

Limitations of current practice. Continual Learning (CL) is the go-to self-healing approach for adapting DRL systems in response to environmental drifts. However, recent studies [27,28] have shown that specific environment drifts tend to drive CL into long, or even unsuccessful, healing cycles, which leads to significant financial burdens for system maintenance.

1.3 Research Statement

The ML code represents only a small proportion of real-world ML systems, and the rest is a broad set of surrounding infrastructure and operations to support these systems’ evolution. Building reliable, production-level ML systems presents several challenges that are not present in proof-of-concept solutions. The major challenges such systems may encounter are now linked to their production rather than development. Upon reviewing research on DL and DRL, as well as the current MLOps state of practice, the following observations were made:

- MLOps is an emerging field that needs more attention from academia. The community currently focuses mostly on the development phase of ML systems, with little attention

devoted to ML deployment. This is clearly seen in some emerging ML fields such as DRL. To the best of our knowledge, no studies have been proposed to investigate the challenges practitioners face when deploying DRL systems.

- Post-training compression strategies such as quantization can induce divergent behaviors between the original model and its compressed version causing problems in production. Ensuring the reliability of these strategies thus becomes a critical challenge. Traditional testing methods, which gather more and more genuine data for better assessment, are often not practical because of the large size of the input space and the high similarity between the original and the compressed model. In contrast, domain knowledge and advanced search-based approaches can be leveraged to build input transformations that assess the reliability of post-training compression strategies.
- Environment drift, the change in the characteristics of the RL environment in production, can cause a DRL agent’s performance to deteriorate over time, dramatically raising operational expenses. Continuous Learning (CL) is the go-to self-healing approach for adapting DRL systems in response to the environment shift. However, in DRL, recent studies [27] have shown that environmental drifts can drive CL into long, or even unsuccessful, healing cycles. More efficient and effective self-healing strategies are thus needed.

This thesis makes the following contribution to three important aspects related to the production-readiness of ML systems: (i) an empirical study to uncover and understand the challenges practitioners face when deploying DRL systems; (ii) a search-based testing approach to assess the reliability of post-training compression strategies, notably DNN quantization; (iii) a self-healing approach for DRL in order to accelerate the agent’s adaptation to environment drift (i.e., data drift). Our research extends its scope beyond classical ML types to include DL and DRL and spans various domains, encompassing hyperspectral remote sensing imaging and gaming applications. By addressing production-readiness challenges in diverse ML contexts, we aim to emphasize the principles of MLOps and enhance the reliability and deployment efficiency of ML systems across industries.

1.4 Thesis Overview

In this section, we present an overview of the key studies that form the foundation of this thesis, providing readers with a roadmap to go through our research journey.

1. *An empirical study on DRL deployment challenges.* We conduct an empirical study

on Stack Overflow (SO) leveraging a variety of qualitative and quantitative techniques to investigate the challenges in deploying DRL systems. We categorize relevant SO posts by deployment platforms: server/cloud, mobile/embedded system, browser, and game engine. After filtering and manual analysis, we examined 357 SO posts about DRL deployment, investigated the current state, and identified the challenges related to deploying DRL systems. Then, we investigate the prevalence and difficulty of these challenges.

2. *DiverGet: A search-based testing approach for DNN quantization assessment.* We propose a search-based testing approach dedicated to quantization assessment. The goal is to evaluate the real impact of quantization methods by detecting difference-inducing inputs that produce behavioral disagreements between an original DNN and its quantized version. To do so, our approach (i) extends the search space using semantically preserving metamorphic relations, then (ii) explores the search space following population-based metaheuristics, and (iii) uses diverse and complementary fitness functions to guide the search process and generate quality test data. we evaluate our approach on Remote Sensing Hyperspectral Images (HSIs) to demonstrate its effectiveness in detecting behavioral divergence caused by quantization.
3. *Dr. DRL: A self-healing approach for DRL systems.* We introduce a new DRL self-healing approach leveraging an intentional forgetting mechanism to systematically prioritize the adaptation of a DRL agent’s problem-solving ability in the presence of environmental drift in real-world production settings. Our approach deliberately erases the minor behaviors of the DRL system in order to systematically emphasize the adaptation of the major behavior, hence enhancing the speed of self-healing. Dr. DRL heals the DRL agent in two steps: (i) re-initializes the weights of hypoactive neurons with low-scaled values, and (ii) executes continuous learning updates of the patched DRL agent on the newly-drifted environment. To assess the effectiveness of our proposed improvement, we perform a series of self-healing experiments on different environments with distinct parameter changes. These experiments involve employing several DRL agents, using both Dr. DRL and vanilla CL.

1.5 Thesis contributions

- We conduct an empirical study on Stack Overflow (SO) leveraging a variety of qualitative and quantitative techniques. Our empirical study enables practitioners to understand the challenges faced when deploying DRL systems. Results of our study show that

the general interest in DRL deployment is growing, confirming the study’s relevance and importance. Results also show that DRL deployment is more difficult than other DRL issues. Additionally, we built a taxonomy of 31 unique challenges in deploying DRL to different platforms. On all platforms, RL environment-related challenges are the most popular, and communication-related challenges are the most difficult among practitioners.

- We propose, DiverGet, a DNN quantization assessment approach that combines metamorphic relations and population-based metaheuristics. Our search-based testing approach uncovers the real impact of quantization techniques by detecting difference-inducing inputs that produce behavioral disagreements between an original DNN and its quantized version. Results using state-of-the-art HSI DNNs show that our approach succeeds in generating meaningful test inputs that induce behavioral disagreements among DNN versions. It also succeeds in outperforming the state-of-the-art testing framework, DiffChaser, in detecting divergences induced by quantization, with a success rate that is (on average) four times higher.
- We propose Dr. DRL, an effective self-healing approach for DRL systems that introduces a novel mechanism of intentional forgetting to improve CL. Dr. DRL erases the DRL system’s minor behaviors in order to accentuate the adaptation of the major behavior, hence reducing self-healing time. Dr. DRL achieves (i) faster healing of the DRL agent, (ii) a higher number of adaptable environments, and (iii) equal or more elevated average rewards than vanilla CL. More precisely, Dr. DRL is able to reduce, on average, the healing time and fine-tuning episodes by, respectively, 18.74% and 17.72%. Dr. DRL successfully helps agents to adapt to 19.63% of drifted environments left unsolved by vanilla CL while maintaining and even enhancing by up to 45% the obtained rewards for drifted environments that are resolved by both approaches.
- To encourage additional research on these topics and the conduct of replication studies, we provide the following replication packages [29, 30].

1.6 Organisation of the thesis

The remainder of this thesis is as follows:

- Chapter 2 introduces the key ideas and techniques linked to MLOps as well as DL and DRL production readiness that are required to comprehend our studies.
- Chapter 3 provides a literature review of related studies.

- Chapter 4 presents the methodology and results of our empirical study on DRL deployment challenges.
- Chapter 5 presents our search-based testing approach for DNN quantization assessment, its design, implementation, and evaluation on state-of-the-art DNNs applied to hyperspectral remote sensing images.
- Chapter 6 describes our self-healing approach for DRL systems, its design, implementation, and evaluation using three popular gym environments and three well-established DRL algorithms.
- Chapter 7 summarizes the thesis and highlights the thesis limitations as well as future work.

CHAPTER 2 BACKGROUND

Chapter Overview. In this chapter, we introduce deep learning and deep reinforcement learning fundamentals (Section 2.1), and present some MLOps concepts such as quantization, environmental drift, continuous learning, and software healing (Section 2.2). We also introduce software testing techniques applied in machine learning such as metamorphic testing, search-based software testing, and differential testing (Section 2.3). Finally, Section 2.4 summarizes the chapter.

2.1 Deep Learning and Deep Reinforcement Learning

This section provides an overview of the key concepts underlying deep learning and deep reinforcement learning.

2.1.1 Deep Neural Networks

A deep neural network (DNN) is an artificial network composed of a collection of interconnected neurons arranged in a series of several layers, hence the term “deep”. More formally, a DNN with K layers is a tuple $\mathcal{D} = (L, C, F)$, where $L = \{l_k \mid k \in \{1..K\}\}$ denotes the set of layers, $C \subseteq L \times L$ represents the connections between layers, and $F = \{f_k \mid k \in \{2, \dots, K\}\}$ denotes the set of functions, with each function corresponding to a non-input layer. Within a DNN, the initial layer, l_1 , serves as the input layer. The last layer, l_K , serves as the output layer. Hidden layers are any layers that exist between the input and output layers. A DNN is a mathematical model that represents a mapping function $y = f(x)$, where x is the input and y is the corresponding output. f is defined as the composition of all the subfunctions $f(f_{k-1}; W_k, b_k)$, where W_k and b_k represent namely the weights and biases, respectively. These two entities represent the learnable parameters of a DNN.

$$f(x; W, b) = f_K \left(f_{K-1} \left(\dots f_1 \left(x; W^{[1]}, b^{[1]} \right) \right) \dots; W^{[K]}, b^{[K]} \right) \quad (2.1)$$

$$W = \{W^{[k]}, \forall k \in [1, K]\}; b = \{b^{[k]}, \forall k \in [1, K]\} \quad (2.2)$$

DNNs update their learnable parameters through the use of a backpropagation algorithm. This algorithm iteratively adjusts the parameters of the DNN in order to approach an ideal parameter configuration, hence generating a model that exhibits good fitting capabilities and

minimal error. The backpropagation consists of two primary steps: the forward pass and the backward pass.

Forward Pass. In the forward pass, each hidden layer k conducts a linear transformation $z^{[k]}$ on its inputs, which are computed by the preceding layer. Subsequently, a non-linear activation function g is applied. Therefore, the forward pass may be expressed in the following manner:

$$\begin{aligned} z^{[k]} &= W^{[k]}a^{[k-1]} + b^{[k]} \\ a^{[k]} &= g\left(z^{[k]}\right) \end{aligned} \quad (2.3)$$

Backward Pass. The backward pass is executed by calculating the gradients of the loss function, denoted as $\oplus$, with respect to the weighted-sum outputs of each layer, represented as $z^{[k]}$. Subsequently, the gradients are computed with respect to the layer parameters, namely $W^{[k]}$ and $b^{[k]}$, as expressed in the equations below. The use of the derivative chain rule allows for the gradients in a layer to be utilized in the calculation of gradients in neighboring layers.

$$\frac{\partial \oplus}{\partial z^{[k]}} = \left[W^{[k+1]}\right]^T \cdot \frac{\partial \oplus}{\partial z^{[k+1]}} * g^{[k]'}\left(z^{[k]}\right) \quad (2.4)$$

$$\frac{\partial \oplus}{\partial W^{[k]}} = \frac{\partial \oplus}{\partial z^{[k]}} \cdot \left[a^{[k-1]}\right]^T \quad ; \quad \frac{\partial \oplus}{\partial b^{[k]}} = \frac{\partial \oplus}{\partial z^{[k]}} \quad (2.5)$$

Hence, the learnable parameters are updated in the opposite direction of their gradients with respect to the loss function $\oplus$, employing a learning rate α , as expressed in the following equation.

$$W^{(i+1)} = W^{(i)} - \alpha \frac{\partial \oplus}{\partial W^{[k]}} \quad ; \quad b^{(i+1)} = b^{(i)} - \alpha \frac{\partial \oplus}{\partial b^{[k]}} \quad (2.6)$$

Hypoactive Neurons vs Hyperactive Neurons. Recently, several studies have been conducted to illustrate the interpretability of DNNs. These studies have revealed that hyperactive neurons, characterized by their elevated levels of activity, can contribute to the acquisition of valuable learning patterns [19,31]. In their study, Ma et al. [19] identify neurons that exhibit hyperactivity (i.e., hyperactive neurons) and analyze their combinations to determine their role in DNN's major behavior. Conversely, they assign minor DNN behavior to neurons that display hypoactivity (i.e., hypoactive neurons). In our third study (Chapter 6), we leverage these previous findings and utilize the established characterization of hyperactive and hypoactive neurons in order to discern the underlying structural patterns associated with major and minor behaviors within DNNs. DNNs serve as the foundation for

DL and DRL. They allow these two ML fields to overcome the curse of dimensionality and reach human-level autonomy in previously intractable fields such as robotics [32], computer games [33], and computer vision [26]. In the subsequent sections, we define DL and DRL, along with an overview of relevant concepts associated with these fields.

2.1.2 Deep Learning

Deep learning is a subfield within machine learning that aims to construct representations of data at a higher degree of abstraction by utilizing DNNs. DL is often regarded as the go-to solution in fields with extensive and high-dimensional datasets. This is mostly due to the superior performance of DNNs compared to shallow ML algorithms across several applications including text [34], vision [35], and audio [36]. With the expansion in data volume and computational capacity, DL systems have gained significant interest and have been applied in a variety of sectors [37].

A DL system is any software system that has at least one DNN component [19]. This type of system might be constituted of solely DNN components (e.g., object detection systems) or a combination of DNN components and other traditional software collaborating to achieve a final outcome (e.g., recommendation systems). Unlike a typical software system, where the system behavior is governed by the business logic, the DL system, in addition to the business logic, learns its rules automatically from data. Its behavior is defined both by the code that implements it and by the data used to train its DNN components.

The data-driven characteristic of DL systems has significant implications for their reliability. These systems are susceptible to poor data quality, necessitating the use of sophisticated evaluation methods to assure their optimal performance in real-world operational settings. For example, the evaluation of post-training compression strategies like quantization necessitates the utilization of advanced assessment techniques to accurately determine the practical implications of these strategies in a production. A more comprehensive examination of quantization and its effects will be presented in Section 5.

2.1.3 Deep Reinforcement Learning

Reinforcement Learning (RL) [38] is a mathematical framework for experience-driven autonomous learning. RL aims to create fully autonomous agents that interact with their environments to learn optimum behaviors and improve over time, through a trial-and-error paradigm [38]. Formally, the agent is given a state s at each time-step t of its interaction with the environment, and it decides which action to take based on that state [20]. Those

actions are determined by its on-learning policy, π , encoded as state-to-action mappings. Once an action is taken, the agent receives a reward and the environment moves to the next state.

Deep reinforcement learning (DRL), leverages DL to scale RL to complex, high-dimensional state and action spaces [20,21]. Neural Networks (NN) are non-linear function approximators that are trained to estimate quantities that depend on state and/or actions, e.g., a policy π or an action-value function Q . DRL algorithms are either model-free or model-based [21]. To avoid modeling biases and solve environments with complex dynamics, we decided to focus our study on model-free DRL algorithms. These algorithms can be divided into three categories [21].

Policy gradients algorithms. The policy π is represented by a neural network whose weights are updated by maximizing the expected return. This optimization is performed on policy, meaning that each update is carried out with data (i.e., states, actions, and rewards) coming exclusively from the actual policy.

Value-based algorithms. The action-value function, Q , is represented by a neural network whose weights are updated based on the Bellman equation [39]. This optimization is performed off-policy, meaning that it leverages data coming from any policy, not only from the actual policy. Based on a fitted Q function, the policy is derived from choosing the action with the highest Q value for all the states.

Hybrid algorithms. DRL algorithms belonging to this category use both policy gradients and value-based methods. One notable class of algorithms from this category is the Actor-Critic algorithm which has two networks: Actor and Critic. The actor decides which action should be taken and critics inform the actor how good the action was and how it should adjust. The learning of the actor is based on policy gradient approach, the critic evaluates the action produced by the actor by computing the value function.

2.1.4 DRL development and deployment

This section briefly discusses DRL-based system development and deployment. Interested readers might turn to [40] for a more extensive discussion about the concrete design lifecycle of a DRL-based robot.

DRL Development Lifecycle: The development of DRL-based systems involves four main steps: design, control, training, and verification [40]. First, developers select and combine the components of the system to construct its final structure (i.e., design step). Then, they start generating modules to control the internal and external behavior of the system (i.e.,

control step). For instance, developers in this step design object detection generators to perceive the environment and behavior generators to control the robot’s motion. Next, we have the training step where developers configure the DRL agent, train it, and fine-tune its hyperparameters. Finally, developers repeatedly verify and update the system’s settings from the control and training steps to improve the agent’s learning capabilities.

DRL Deployment Lifecycle: After verifying and testing the DRL agent, the system is ready for deployment by transferring the learned control policy to a physical or virtual platform for real-world use [40]. A common approach is to deploy DRL systems on physical servers or on Cloud [41–44]. This approach offers developers tools and services like TensorFlow Serving [45] or Amazon Sagemaker [46] to accelerate and facilitate deployment.

In addition, other platforms for deploying DRL systems, such as mobile and embedded devices, are becoming popular [47,48]. However, practical-sized DRL agents cannot be deployed directly to these edge platforms because of their low computational power, memory size, and energy cost. To cope with limited edge platforms’ resources, lightweight DL frameworks, such as TF Lite [49] and Core ML [50], are built to reduce the DNN footprint. These frameworks are also used in DRL to compress the agent’s DNN [51]. To decrease memory cost and processing overhead, TF Lite [49] and Core ML [50] employ model compression strategies such as quantization before deploying DRL models to edge platforms.

Furthermore, DRL systems may also be deployed in virtual platforms like browsers [52,53] and game engines [40,54]. For browser deployment, developers employ particular frameworks and libraries for adapting DRL agents, such as TensorFlow.js and brain.js. Game engines, on the other hand, are frameworks conceived primarily for the design of video games. Popular game engines, such as Unity3d, provide neural network inference packages (e.g., Barracuda [55]) that enable the usage of DRL agents within games. Consequently, developers may build DRL agents using frameworks like TensorFlow and PyTorch before deploying them in a game engine for inference.

2.2 MLOps: Towards production-ready ML systems

DevOps and DataOps have gained popularity in the fields of software engineering and data engineering since they significantly improve software quality and time to market. With the increasing rise of ML, comparable approaches must be developed within the field of ML engineering, while accounting for the unique properties of ML systems. Indeed, achieving business value through ML requires more than simply developing the best ML model; it is about building a cohesive ML system that functions consistently and adjusts to changes in

the business’s dynamics. This includes preparing datasets and features; training models at scale; serving models; and monitoring models’ performance in production.

Machine Learning Operations (MLOps) is an ML engineering methodology that formalizes and automates all of the above steps with the goal of blending ML development (ML) with ML operations (Ops). MLOps encompasses a collection of standardized procedures and technologies for the efficient and reliable production of ML systems. MLOps offers three levels of automation for ML lifecycle steps that characterize the maturity of the ML systems and reflect the velocity of training and serving new ML models. Level 0, the most common level, involves no automation in building and deploying ML models. Level 1 automates the ML pipeline, adds continuous training (CT), and enables continuous delivery (CD) of model prediction service. Level 2 automates both ML and CI/CD pipelines. This level extends beyond continuously deploying new models, to continuously deploying new pipeline components to the target environment.

In the following, we provide an overview of MLOps principles pertaining to quantization, environmental drift, continuous learning, and software healing.

2.2.1 Quantization

Quantization is one of the most used compression methods to host large-scale DNNs on edge devices. It converts the involved data tensors to low-precision arithmetic types with the aim of downsizing the model footprint. For instance, a variant of quantization consists of decreasing the precision of weights and activations from single-precision floating-point format (32 bits) to half-precision floating-point format (16 bits) or integer format (8 bits) [56].

Among the benefits of the DNN’s footprint reduction, we find faster computation, lower memory usage, as well as lower power consumption [57]. Nonetheless, the resulting compression often comes with a degradation in the predictive performance of the quantized DNN. To assess this degradation, practitioners often gather as much labeled test data as possible and run it through the quantized DNN to quantify the drop in accuracy.

Recent quantization techniques perform well in keeping the accuracy of quantized DNNs equally matching their full-precision versions. In the following, we describe the most recent quantization methods [58, 59] that have been leveraged in our quantization assessment study (Chapter 5):

- Post Training Quantization (PTQ) compresses the DNNs by reducing the precision of either their weights or both their weights and activations, from a 32-bit floating-point format to a lower format without requiring retraining the model. It is a very simple

approach and it permits quantization with little to no data.

- Quantization Aware Training (QAT) performs quantization during training by adding quantization operators to the DNN, to simulate the drop in precision in the forward propagation pass (the backward pass remains unchanged). This trick will add up the quantization error to the total loss and will make the optimizer reduce it appropriately. Therefore, QAT is known to provide better accuracies than PTQ, though it requires available training data.

2.2.2 Environment Drift and Continual Learning

Standard RL environment settings are often not representative of all possible contexts that agents will encounter when completing a task [26, 60]. The behavior of these environments is subject to changes, which can manifest as alterations in real deployment environments or shifts in the parameters of virtual environments [61, 62]. These challenges, known as partial observability and non-stationarity [60] in DRL literature, hinder productionizing DRL to real-world problems [26, 60]. For instance, recommendation systems have no observations of the ever-changing mental state of the users and need to always heal their behavior accordingly. These changes in the production RL environment, called environment drift, may affect the performance of trained agents. The environment’s behavioral changes become classified as environmental drifts when their severity/intensity is substantial, preventing the trained agent from achieving its task. Although a drifted environment yields different rewards for a subset of state-action pairs, it should not be considered a completely new environment. In fact, the original environment and its drifted counterpart share some regions of state-action space.

However, in a completely new environment, the state-action space has no guaranteed connections with any previous environment. Hence, an agent should begin the learning process from scratch in a completely new environment because it requires replacement learning, whereas learning in a drift environment requires healing by further fine-tuning [63], i.e., the fusion of new information with the existing knowledge.

This required fine-tuning is called the continual learning (CL) problem in DL [64], which investigates whether DNNs can acquire new knowledge incrementally. In spite of its success in several use cases, it has been shown that new knowledge may interfere with the existing one, which results in replacing it entirely. This failure of stability in CL is called catastrophic forgetting [65], in which new experience overwrites previous experience. Thus, the fundamental objective is to design an advanced self-healing approach to improve the adaptability of DRL systems effectively and efficiently.

2.2.3 Software Self-Healing and Intentional Forgetting

More than ever, software systems must deal with change and perform under conditions other than those for which it was intended. Continuous software availability requires considering shifts in production conditions as part of the development process. Software healing [66] has been explored to fix software system issues “in the field”, as opposed to software repair [67], which is used “in-house”. Indeed, healing approaches respond to software failures in production by making the appropriate adjustments, at runtime, to restore the system’s normal behavior [66]. When these adjustments are fully automated without human intervention, the approaches are called software self-healing [68–70]. Software healing starts with the detection of misbehaving software that deviates from its intended function. Then, the healing operation involves the resolution of detected failures, or at least, mitigating their undesirable effects. Finally, a verification step ensures that the software system is restored properly. Otherwise, the healing step may be repeated until the issue is fixed or no more actions are possible [66]. DRL systems are data-driven and able to continuously adjust their internals to environmental shifts [27]. Nevertheless, the CL of DRL systems can be less effective and more time-consuming as environmental drift becomes more severe. In this thesis, we aim to propose a novel advanced self-healing operation for DRL systems that leverages an intentional forgetting mechanism (more details can be found in Chapter 6).

Inspired by human cognition, intentional forgetting mechanisms are designed to improve decision-making by maintaining relevant knowledge [71]. Indeed, researchers have employed intentional forgetting in computer science fields including cybersecurity [72] and machine learning [71, 73, 74]. In general, systems should be able to purposefully remove behaviors that lose relevance in relation to certain condition changes, either totally or partly [75].

2.3 Software Testing Techniques in Machine Learning

This section introduces DL testing and provides an overview of software testing techniques employed in ML-based systems; including metamorphic testing, search-based software testing, and differential testing.

2.3.1 DL Testing:

The growing integration of Deep Learning (DL) systems within safety-critical domains, coupled with their inherently opaque nature, underscores the critical importance of rigorously testing and evaluating these systems.

However, it is difficult to test large-scale DL systems with millions of neurons and thousands of parameters for all potential inputs (including all corner cases). In the following, we briefly present definitions and concepts to introduce DL testing. A more in-depth discussion may be found in surveys such as [76, 77].

First, *DL bugs* are any defects in a DL part that produce differences between current and required DL system behavior [76]. DL systems may have different types of required behaviors, such as correctness, robustness, fairness, and privacy. Correctness is the extent to which a DL system can behave exactly as planned, whereas robustness is the persistence of an ML system’s correctness in the presence of stressful conditions. Fairness is the degree to which a DL system is free of biases, while privacy is the DL system’s capacity to safeguard private data information [76].

Deep Learning Testing is any activity that aims to uncover deep learning bugs and vulnerabilities [76]. Currently, DL testing tools are regularly designed to accomplish a testing objective (such as fairness and robustness). Indeed, their purpose is to evaluate various components of DL systems and discover various types of DL bugs. DeepHunter [31], a deep learning coverage-guided fuzzing tool, revealed thousands of incorrect corner case behaviors in various image classifiers; Themis [78], a fairness testing tool for detecting causal discrimination, detected significant model discrimination towards gender, marital status, or race.

While aiming to assess the quality of systems, DL testing and software testing are widely different [76]. Traditional software testing finds errors in the code, whereas DL testing detects bugs in the data, the DNN, and the DL framework. Traditional software’s behavior is often fixed over time after development, while, the behavior of the DL model might vary over time (data drift and concept drift). Traditional software testing assumes the availability of a test oracle (correct behavior). In DL, however, the property checking of the predicted output is frequently handled manually and is difficult to express formally.

2.3.2 Metamorphic Testing:

In software testing, an oracle is a technique that can help assess if a program is working properly [79].

An oracle can be as simple as directly comparing the program’s output with the correct answer. More complicated oracles may need the execution of another program to validate the output of the target program.

For instance, model-based oracle [80] and statistical oracle [81] test and verify system behavior employing model-based testing [82] techniques and probabilistic analysis, respectively.

The peculiarity of ML is that there is generally no oracle without human intervention (e.g., human labeling). Several techniques have been proposed to mitigate the oracle problem, including using pseudo-oracle and metamorphic testing. Pseudo-oracle [83] refers to several implementations addressing the same problem and verifying one another. The outcomes of all implementations are compared, and if one varies, it may indicate that the system has flaws.

Metamorphic Testing [84] is a pseudo-oracle testing technique proposed to overcome the oracle problem. The idea behind this strategy is that even if we don't know what an input's expected outcome is, we may build relationships - called metamorphic relations (MR) - between it and other inputs to establish a common behavior.

This allows for the identification of incorrect behaviors by identifying violations in validating these MRs. This technique designs MRs that connect inputs so that the connection between their corresponding outputs is known in advance, allowing the intended outputs to be predicted. Transformation (often known as metamorphic transformations (MT)) can be employed to provide new inputs for establishing these MRs. For example, a metamorphic relation for checking the implementation of the function $\sin(x)$ may be the transformation (i.e., the MT) of input x to $\pi - x$, which allows analyzing the results by detecting if $\sin(x)$ differs from $\sin(\pi - x)$.

2.3.3 Search-based Software Testing (SBST)

Search-Based Software Testing (SBST) dates from 1976 and back then, there was little interest in it. However, in the last decade, SBST has been used to solve a wide range of testing problems [85].

SBST formulates the target test criteria as a fitness function that compares and contrasts candidate solutions from the search space in terms of their adequacy. To resolve the formulated optimization problem, SBST techniques leverage metaheuristic search algorithms, as gradient-free optimizers. These latter require few to no assumptions on the properties of both the objective function and the inputs search space. However, they do not provide any guarantee of finding an optimal solution. Their applicability in input test generation is suitable as these problems frequently encounter competing constraints and require near-optimal solutions, and these metaheuristics seek solutions for combinatorial problems at a reasonable computational cost.

In our second study on DNN quantization assessment, we opt for nature-inspired population-based metaheuristics, such as Particle Swarm Optimization (PSO) [86] and Genetic Algo-

rithm (GA) [87]. These latter possess intrinsically complex routines and non-determinism that make them high-potential candidates for spotting vulnerable regions in the large, multi-dimensional input space of the Deep Learning (DL) models.

2.3.4 Differential Testing

Differential testing is a software engineering approach that aims to identify discrepancies between two or more comparable systems, which include minor variations in their implementation when subjected to identical inputs.

In recent times, Differential Testing has been employed in the field of Deep Learning Testing to identify inputs that produce behavioral discrepancies between slightly varied Deep Neural Networks (DNNs). The dissimilarities across DNNs may be categorized into two main aspects: conceptual differences and environmental differences. Conceptual differences refer to variations in training data or hyperparameters, while environmental differences pertain to disparities in the hardware environment, such as the use of 16-bit systems, 8-bit systems, and so on.

2.4 Chapter Summary

In this chapter, we describe fundamental concepts about DL, DRL, and MLOps in order to facilitate the comprehension of the studies, concepts, and approaches proposed in this thesis. In the following chapter, we review the literature about the operationalization of DL and DRL systems.

CHAPTER 3 LITERATURE REVIEW

Chapter Overview: In this chapter, we examine recent studies on DL test case generation. Following that, we highlight current research on DL and DRL deployment challenges. Finally, we discuss existing approaches for DL and DRL repair and healing. This review also explains the gaps in the literature that we tackled in our research. The outline of this chapter is as follows: Section 3.1 introduces studies on DL test case generation. Section 3.2 presents current research on DL and DRL deployment challenges. Section 3.3 discusses the existing approaches for DL and DRL repair and healing. Section 3.4 concludes the chapter.

3.1 Studies on DL Test Case Generation

Over the last few years, researchers have adapted concepts and methods from software testing to invent advanced DL testing approaches [77]. One common denominator of these new testing approaches is their focus on generating synthetic test data with a high ability to reveal erroneous behaviors of the DL models. Their test data generators consist in solving a maximization problem, i.e., finding the inputs that enhance the test adequacy criterion, relying on gradient-based optimizers [88,89], gradient-free optimizers (i.e, metaheuristics) [90, 91], and greedy search algorithms [92–94].

Although adversarial attacks [95] operate similarly by generating maliciously crafted inputs, they maximize the probabilities of other labels or the closest one against the correct label’s probability, aiming to alter the prediction of the DNN to yield a wrong label. On the other hand, DL testing approaches adapt diverse test adequacy criteria, inspired by traditional software testing domain. Pei et al. [88] proposed Neuron Coverage (NC), the ratio of newly-activated neurons by test inputs, which is inspired by conventional code coverage. Next, Ma et al. [89] generalized NC by defining multi-granularity testing criteria, i.e., neuron-level and layer-level coverage. For instance, they refine NC to target (i) the major neuronal behaviors using K-multisection Neuron Coverage (KMNC), the ratio of the covered k-multisections of neurons, and (ii) minor corner-case neuronal behaviors using Neuron Boundary Coverage (NBC), the ratio of the covered boundary region of neurons.

While most of the DL testing approaches target the search for inconsistencies in the best-fitted DNN, a few of them consider the detection of disagreements between multiple DNNs with different arithmetic precision. TensorFuzz [93] was the first method leveraging a coverage-guided fuzzing test generator to expose difference-inducing inputs between the best-fitted

DNN and its quantized counterpart. DeepHunter [92] follows the same strategy of coverage-guided fuzzing but enriches the data perturbations from only noise addition to 8 different image-based transformations, including pixel-value perturbations and geometric transformations. DeepEvolution [90] builds on top of the last work by preserving the diversity of input data transformations. Additionally, it optimizes the generation of test inputs towards the most prominent regions using a metaheuristic-based search approach and was instantiated by 9 different nature-inspired population-based metaheuristic algorithms. However, the main limitation of all these early-released research works is their empirical evaluation, which was made on simple quantized models that are usually hand-crafted and weights-only.

Recently, DiffChaser [91] challenged the state-of-the-art quantization methods supported by mainstream DL libraries, including Tensorflow Lite and CoreML. Its core approach relies on a search-based generation of distorted inputs using a genetic algorithm (GA) to perform untargeted/targeted attacks to expose behavioral divergences amongst different subjects of low/high arithmetic precision DNNs. Despite significant advances over previous approaches that were generic and less effective, DiffChaser focused on the simple detection of the unavoidable existence of divergence between two DNN versions with different arithmetic precision. None of these mentioned studies assessed the quantization-induced divergence rate (i.e., the ratio of disagreement behaviors with respect to the total number of passed tests) and compared the rates obtained by different quantization techniques.

DiffChaser, the most recent DL testing framework for quantization degradation, will serve as a baseline for our search-based framework, DiverGet, for DNN quantization assessment (Chapter 5).

3.2 Studies on DL and DRL Deployment Challenges

In recent years, ML has solved various real-world problems. Yet, the deployment of ML models remains challenging. Numerous software engineering (SE) studies have addressed these challenges to help practitioners. Breck et al. [96] provided an actionable checklist of 28 tests and monitoring criteria to assess ML system production readiness. Similarly, Paleyes et al. propose practical consideration by evaluating reports on deploying ML systems (including RL).

Recently, researchers focused on DL-specific deployment challenges across several platforms. Cummaudo et al. [97] examined developer challenges with computer vision services (i.e., APIs). Ma et al. [53] assessed seven JavaScript-based DL frameworks on Chrome and found that DL in browsers is still in its early stages. Xu et al. [98] proposed the first empirical

study on DL in Android applications. Guo et al. [99] examined the performance gap of DL models when moved to mobile devices and Web browsers, whereas Openja et al. [100] evaluated ONNX and CoreML for deploying DL models. Finally, Chen et al. [101] examined the challenges of DL deployment across three platforms.

Despite all these efforts, DRL-specific deployment challenges remain unstudied. Instead, few SE research like [102] examined DRL development challenges. Unlike these works, our empirical study (Chapter 4) focuses on DRL deployment to bridge the knowledge gap between research and practice. The current DRL deployment research is focused on addressing the problem from an academic perspective. Panzer and Bender [26] presented a systematic literature review of DRL deployment in production and provided a global overview of DRL applications in various production system domains. In addition, Dulac-Arnold et al. [60] list nine specific challenges (e.g., partial observability, and non-stationarity) to productionize RL to real-world situations. For each challenge, they propose literature-based methodologies and metrics for evaluation. These studies complement ours since they use literature to extract and understand DRL deployment challenges in production, while we focus on practitioners and the industry.

3.3 Studies on DL and DRL Repair and Healing

Software healing approaches [66] have been explored to fix software system issues “in the field”, as opposed to software repair solutions [67], which are used “in-house” for bug-fixing operations. Indeed, healing approaches respond to software failures in production by making the appropriate adjustments, at runtime, to restore the system’s normal behavior [66]. When these adjustments are fully automated without human intervention, the approaches are called software self-healing [68–70].

Due to their data-driven nature and unique architecture, specific healing, and repairing approaches have been proposed for ML systems. Rusu et al. [103] suggest freezing changes to previously trained networks and introducing additional sub-networks to learn new jobs. Shin et al. [104] train a generative model to create data with the same original data distribution. Thus, original data may be mixed with new data to preserve current knowledge and develop new skills. Next, the learning without forgetting (LwF) [105] approach limits network predictions on past tasks from changing while optimizing new task parameters. The above approaches focused on fine-tuning the model using more data. Yet, the fine-tuned model may not remove undesired behavior or efficiently learn new behaviors without catastrophic forgetting [65, 106]. Other approaches tried to address the problem without model retraining. Apricot [107] is one attempt to fix DNN iteratively through a weight-adaptation

method. Additionally, Arachne [108], a search-based DNN repair method, modified weights directly instead of retraining. Finally, Stocco and Tonella [109] proposed a novel CL method for fixing misbehavior in DNNs used in self-driving cars. When data distribution shifts, this technique updates predictors using in-field confidence metric selection and error-based weighted retraining.

Despite all these attempts, DRL-specific healing is still unstudied. Instead, few SE research such as Biagiola and Tonella [27] examined the confidence boundary of CL. Biagiola and Tonella [27] proposed AlphaTest, an approach for characterizing the DRL’s adaptability in its environment. AlphaTest takes a DRL agent that has been trained in a parameterized environment as input and samples the environment parameter space. It then trains the agent in CL mode on the sampled environment configurations, with the goal of characterizing the agent’s adaptability frontier. Leveraging AlphaTest, developers can learn about the healing boundaries of CL.

Unlike prior research, We propose, in Chapter 6, a DRL-specific self-healing approach that uses intentional forgetting [72] and CL to optimize the agent’s adaptability. The proposed *Dr. DRL* could benefit the DRL community since no advanced self-healing approaches have yet been proposed. Instead, traditional healing alternatives are used like training a new policy from scratch or using vanilla CL.

3.4 Chapter Summary

Over the last decade, academia and industries have been working increasingly to investigate DL and DRL reliability, deployment, and operationalization issues. In this chapter, we first review the DL test case generation studies. We then highlighted the DL and DRL deployment challenges evoked in the literature. Finally, we examine the DL and DRL Repair and Healing approaches reported in the literature. In all these sections, we explain the literature gaps that we identified and addressed in our studies described in the next chapters.

CHAPTER 4 DEPLOYING DEEP REINFORCEMENT LEARNING SYSTEMS: A TAXONOMY OF CHALLENGES

Reinforcement Learning (RL) is a subfield of Machine Learning (ML) concerned with autonomous learning and decision-making based on interacting with an environment [15]. RL follows a trial-and-error paradigm where an agent interacts with its environment and learns to adapt its behavior to achieve a goal by observing the outcomes (i.e., rewards) of its actions [15, 20, 21]. RL was at first unpopular since early approaches were constrained to low-dimensional tasks and lacked scalability [15, 110]. Deep Reinforcement Learning (DRL), leveraging Deep Learning (DL) in RL, sparked a rebirth in RL and revived interest in this field. It was a step toward developing autonomous systems with a deeper awareness of the surrounding world. DL is currently allowing RL to achieve human-level autonomy in previously intractable fields, such as robotics [32], computer games [33], and computer vision [26].

This potential explains the enthusiasm and rising interest in DRL in academics and industry. Frameworks and libraries like Stable Baselines [111], Keras-RL [112], and TensorForce [113] are continually being released to relieve the cost of building DRL solutions from scratch. Academia and industry are also working together to assist researchers and practitioners in handling DRL’s new challenges. For example, Nikanjam et al. [102] studied real faults that occurred while developing DRL programs and produced a taxonomy of these faults. From another perspective, the growing demand for DRL-based systems has raised new deployment concerns. For instance, these systems’ high computational and energy costs prevent their direct deployment on platforms with low processing power (e.g., drone navigation) [47, 48]. Even worse, these additional deployment concerns are generally non-trivial and more difficult compared to vanilla DL deployment. Quantization [114, 115], for instance, is more challenging in DRL and may hinder the policy’s long-term decision-making since the agent’s current action strongly affects its future states and actions [116]. However, current research focuses mostly on the development phase of DRL-based systems, with little attention paid to DRL deployment.

In this chapter, we undertake the first attempt at identifying and understanding the challenges practitioners face when deploying DRL-based software systems. We formulate our Research Questions (RQs) as follows:

- **RQ1: What is the current level of interest in deploying DRL-based systems?**
- **RQ2: What are the challenges in deploying DRL-based systems?**

- **RQ3: Which DRL deployment challenges are the most popular and difficult to answer on Stack Overflow?**

We conduct an empirical study on Stack Overflow (SO) leveraging a variety of qualitative and quantitative techniques. We investigate SO as it is the most popular Q&A platform for developers to report their challenges and issues, propose solutions, and spark discussions on various technical topics including DRL [117]. Following similar studies on DL [99, 101], we categorize relevant SO posts by deployment platforms: server/cloud, mobile/embedded system, browser, and game engine. After filtering and manual analysis to remove false positives, we examined 357 SO posts about DRL deployment. Quantitative analysis shows that the general interest in DRL deployment is growing, confirming the study’s relevance and importance. We manually review SO posts and build a comprehensive taxonomy of 31 unique challenges for deploying DRL to the selected platforms. These 31 deployment challenges can be grouped into 11 main categories. Across all platforms, RL environment-related challenges are the most popular whereas communication-related challenges are the most difficult. Also, when considering average scores and median response time as proxies for popularity and difficulty, we found that the difficult/popular challenges are significantly popular/difficult among practitioners. We found that despite increased interest from the DRL community, DRL deployment needs more work to achieve the same maturity level as traditional software systems deployment. Academia should propose more automated strategies for diagnosing and monitoring deployment issues and misconfigurations to help developers, and framework providers should enhance their tools and documentation.

We have prepared a replication package including materials used in this study, that can be used for other studies on DRL deployment [30].

Chapter Overview. The remainder of this chapter is as follows: Section 4.1 covers our methodology. Sections 4.2 to 4.4 report our empirical findings. Section 4.5 discusses the implications of our study. Section 4.6 discusses validity threats, and Section 4.7 concludes the chapter.

4.1 Methodology

To better comprehend the challenges in deploying DRL-based systems, we analyze the relevant questions posted on SO. Figure 4.1 highlights an overview of the three major steps of our methodology. We describe the steps we followed in our methodology in the rest of this section.

Step 1. Download the SO data dump. We download the SO dump from the official

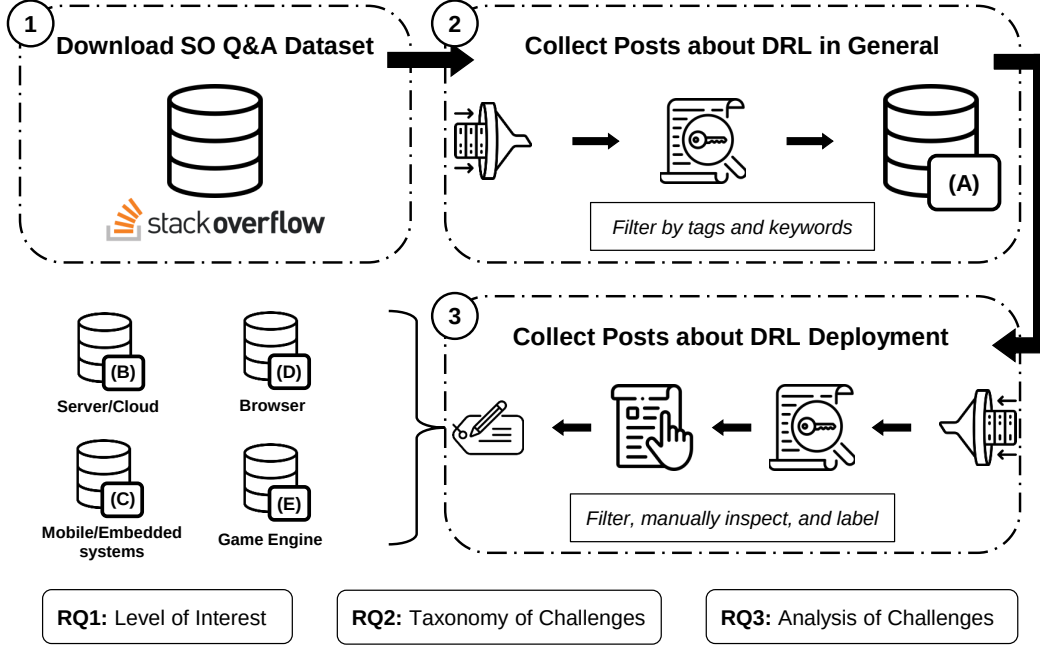


Figure 4.1 Overview of our study’s methodology.

Stack Exchange Data Dump [118] on the 11th of October 2022. The dataset includes posts from July 2008 and each post contains information such as post type (i.e., question, answer, or wiki), creation date, tags, title, body, etc. Each question has one to five tags based on its topics and can have an accepted answer (meaning that the owner of the question has found a valid response to its question in one of the answers).

Step 2. Collect relevant DRL posts. As a first step, we collect SO questions related to DRL in general. This dataset, denoted as A , constitutes the starting point of our analysis. To build A , we first extract questions tagged with “reinforcement-learning” or at least one of the top popular DRL frameworks, such as “stable-baselines” and “keras-rl” (the replication package has the full collection of tags [30]), and found 2996 relevant questions. Since many practitioners are still using classical DL frameworks such as Pytorch and TensorFlow to build DRL-based systems, we complement A with questions tagged with “Pytorch” or “TensorFlow” and having a relevant keyword of DRL (the full collection of keywords can be found in the replication package [30]) within their title or body. Finally, we remove the duplicate questions and have retained a total of 3659 questions in A .

Step 3. Collect relevant DRL deployment posts. Following similar studies [99,101], we select four representative deployment platforms of DRL systems to study, namely server/-cloud, mobile/embedded system, browser, and game engine platforms. After filtering and manual analysis, we collected a dataset of 357 posts related to DRL deployment on all plat-

forms. In the following, we detail the steps to extract these posts for each platform.

Server/Cloud Posts. We first define a vocabulary of words related to server/cloud platforms (such as, “cloud”, “server”, and “serving”). We then extract questions in A having at least one word of Server/Cloud vocabulary in their title or body. We denoted this dataset as B . To further complement B , we extract SO questions having a DRL vocabulary word within their title or body and tagged with TF Serving, Google Cloud ML Engine, Azure ML, IBM Watson, or Amazon SageMaker. These tags represent popular cloud platforms for designing, training, and deploying ML systems and are used in similar studies [101]. After manually removing duplicates and false positives (the manual analysis is detailed in the next paragraphs), we end up with 152 questions in B .

Mobile/Embedded systems Posts. We first define a vocabulary of words related to mobile/embedded system platforms, like “mobile”, and “embedded system” (the replication package has the full collection of keywords [30]). We then extract questions in A having at least one word of this vocabulary in their title or body. We denoted this dataset as C . To further complement C , we extract SO questions having a DRL vocabulary word within their title or body and tagged with at least one DL framework for mobiles/embedded systems such as TF Lite or one mobile/embedded hardware vendor like Arduino. After manually removing the duplicate and false positives, we end up with 69 questions in C .

Browser Posts. We first define a vocabulary of words related to browser platforms (such as, “browser”). We then extract questions in A having at least one word of browser vocabulary in their title or body. We denoted this dataset as D . To further complement D , we extract SO questions having a DRL vocabulary word within their title or body and tagged with tensorflow.js, brain.js, and ml5.js. These tags represent popular technologies used to deploy DL programs to browser platforms. After manually removing the duplicate and false positives, we end up with 45 questions in D .

Game Engine Posts. We first define a vocabulary of words related to game engine platforms (such as, “game engine”). We then extract questions in A having at least one word of game engine vocabulary in their title or body. We denoted this dataset as E . To further complement E , we extract SO questions having a DRL vocabulary word within their title or body and tagged with unity3d, ml-agent, barracuda, and game-engine. These tags represent popular game engine technologies. After manually removing the duplicate and false positives, we end up with 91 questions in E .

RQ1: Level of interest. Following past study [101], we start by computing the number of questions linked to the DRL deployment per year to depict the evolution pattern of DRL deployment. The metrics are derived using datasets B , C , D , and E for each of the previous

eight years (i.e., from 2015 to 2022). Second, to have an idea of the degree of difficulty of the DRL-based system deployment topic, we use two widely adopted metrics [119–121]: the percentage of questions with no accepted answer ($\%nAA$) and the response time needed to receive an accepted answer (RT). As a baseline for comparison, we used questions related to DRL but not related to deployment. To that aim, we remove the deployment-related questions (denoted as *Dep*) (i.e., questions in *B*, *C*, *D*, and *E*) from the DL-related questions (i.e., questions in *A*), and the remaining questions are referred to as non-deployment questions (denoted as *Non-Dep*). In total, we had 2822 posts in *Non-Dep*. Then, with a confidence level of 95% and a confidence interval of 5%, we randomly sampled posts from *Non-Dep*. We randomly sampled since we needed to filter out false positive posts before starting the experiment and manual analysis on the full *Non-Dep* was not practical. Our random sampling yields 339 posts in total. For the first measure (i.e., $\%nAA$), we compute and compare the proportion of questions with no accepted response in *B*, *C*, *D*, and *E*, *Dep*, and *Non-Dep*. For the second measure (i.e., the time needed for an accepted answer), we choose the questions that have obtained accepted answers and then display the distribution and the median response time (MRT) required to get an accepted answer for both deployment and non-deployment questions.

RQ2: Taxonomy of Challenges. We manually examine questions about DRL-system deployment to establish a taxonomy of challenges. The first two authors manually review all posts to eliminate duplicates and false positives. We include all posts related to DRL deployment and exclude false positive posts that address non-deployment concerns (like development). For taxonomy construction, we collected a dataset of 357 posts related to DRL deployment on all platforms. In the following, we present steps for the construction of the taxonomy.

Pilot construction and labeling. First, we randomly sample 40% of the questions used for the taxonomy for a pilot construction of the taxonomy. The taxonomy for each kind of platform is constructed individually based on its corresponding samples. We follow an open coding procedure to inductively create the categories and subcategories of our taxonomy in a bottom-up way by analyzing all questions. The two first authors reviewed and revisited all of the questions to become acquainted with them. During this process, they carefully examine all aspects of each question, including the title, body, code samples, comments, responses, and tags. They then provide brief sentences as initial labels for the questions to illustrate the challenges underlying these questions. They then proceed to categorize the labels and develop a taxonomy of challenges in a hierarchical structure. This process is iterative as they travel back and forth between categories and questions to develop the taxonomy. All problems are discussed and resolved by introducing a third person, the judge.

The judge has extensive expertise in DRL development and deployment, having published papers on DRL in top-tier journals and conferences. Finally, the judge approves all of the taxonomy categories. After constructing the pilot, the two first authors independently label the remaining questions. Questions that cannot be categorized under the present taxonomy are placed in a new category called Pending, and the two first authors discuss whether new categories should be created. Cohen’s Kappa metric for inter-rater agreement during independent labeling is 0.784, suggesting good agreement. In situations where the two first authors could not agree, the post was handed over to the judge to settle the labeling conflicts.

Taxonomy validation. To guarantee that the final taxonomy is accurate and representative of realistic DRL deployment challenges, we validated it with a survey of DRL deployment practitioners/researchers. To attract recruiters, we used two alternative methods. First, we requested candidates via personal contacts. This resulted in a list of 11 candidates who were contacted by email. We obtained 6 positive responses from 4 researchers and 2 practitioners. Second, we leveraged GitHub and SO to gather information on potential survey respondents. To identify individuals with a strong grasp of DRL, we first identified the most popular RL frameworks on GitHub. We then retrieved and ordered contributors depending on their participation in the chosen repositories. To discover SO participants, we leveraged the posts retrieved during the mining phase. Following that, we identify the users who posted the question and answers to the selected posts. We searched the web for each identified person to locate their profile and emails from other sources, such as GitHub [122] and Google Scholar [123] since SO does not display user email addresses. This resulted in a second list of 207 candidates who were contacted by email. We obtained 15 positive responses from 9 researchers and 6 practitioners. Overall, we emailed the survey to 218 individuals, and 21 individuals responded (13 researchers and 8 practitioners), resulting in a participation rate of 9.63%. The participant with the least experience had less than a year of practice in both ML/DL and DRL. The most experienced participant had more than 5 years of experience in both the ML/DL and DRL fields. The ML/DL and DRL experience medians were ‘3-5 years’ and ‘1-3 years,’ respectively. We utilized Google Forms [124], a well-known online tool for data-collecting tasks, to build our survey form. We began the survey with background questions about job titles, DL/DRL experience, and familiar frameworks. Next, we moved on to the questions about our final taxonomy. We divided the survey into sub-categories and provided written descriptions for each category, including examples of its challenges. We provided the taxonomy (a simplified illustration), the name of each category, its description, and three questions related to each category. The first question was a “yes/no” question on whether the participant had ever experienced this challenge. If the answer is yes, we ask two further Likert-scale [125] questions about the severity of the challenge and the needed effort

to resolve it. Thus, we assessed not only the challenge’s occurrence but also its severity as viewed by developers. In the final free-text question of our survey, we asked participants to name DRL deployment challenges they had and are not addressed in the taxonomy. This allowed us to determine if our taxonomy covered all developer challenges and what was missing. The survey questions are in the replication package [30].

RQ3: Analysis of Challenges. After gathering the most common DRL deployment challenges that the SO community faces, the aim is to assess these challenges to identify the ones that are getting more traction and are harder for the DRL community to answer. First, we identify the most popular DRL deployment challenges among developers. To that end, we employ two measures of popularity that have been used in previous work [119,121]: (1) The average number of views from both registered and unregistered users (*AV*) of posts within a category, and (2) the average score (*AS*) of posts within a category. *AV* assesses community interest by showing how frequently posts are visualized in one category. The rationale behind is that a post is popular among developers if many developers read it. *AS* represents posts’ recognized community value. Indeed, SO allows its users to up-vote posts that they find interesting and beneficial, and these votes are then combined to produce a score. After finding popular challenges, we assessed the difficulty of answering these challenges. Finding out if certain subjects are more difficult to answer than others will help us discover which challenges require greater community attention. It also helps us to indicate areas where improved tools/frameworks are needed to assist developers in tackling DRL deployment difficulties. To that end, we evaluate each challenge’s difficulty using the two previously mentioned metrics: (1) (*%nAA*) and (2) (*RT*). Please see section 4.1-RQ1 for further information on these two metrics.

4.2 RQ1: Level of Interest

Figure 4.2 depicts the interest in deploying DRL systems measured by the number of questions on the SO. The graph shows that general interest in this subject is growing, confirming the study’s relevance and importance.

Figure 4.2 illustrates a steady increase in posts discussing deploying DRL on servers/clouds. Moreover, the number of questions about mobile and embedded systems deployment grew considerably in 2018 compared to 2017. The reason is that some major vendors released their DL frameworks for mobile devices in 2017 (e.g., TFLite [126] and CoreML [127]). We can also notice that the number of questions for deploying DRL systems on game engines rises steadily until 2020, when it begins to decline. Finally, we notice that number of questions about DRL deployment on browsers have not fluctuated since 2018. This can be explained

by the release of TF.js [128] in the same year. However, the number of browser deployment questions remains low compared to other platforms, indicating that DRL on browsers is still in its early stages. This low interest in deploying on browsers could be due to browsers' limited resources, whereas DRL agents are resource-intensive.

Figure 4.3 and 4.4 depict the difficulty of deploying DRL systems compared to other areas of DRL development. Figure 4.3 shows the ratio of questions with no accepted answer ($\%nAA$) in DRL deployment and non-deployment-related questions. Whereas, Figure 4.4 shows the time needed to have an accepted answer (RT) for DRL deployment- and non-deployment-related questions. Overall, both figures highlight that deployment-related questions are more difficult than non-deployment-related questions. Figure 4.3 shows that $\%nAA$ for DRL system deployment and non-deployment are 74% and 68%, respectively. This suggests that questions about DRL deployment are more difficult to answer than questions about other DRL issues. More specifically, $\%nAA$ for server/cloud, mobile, and browser platforms are 78%, 75%, and 78% respectively. This suggests that deployment questions in these platforms are more difficult to answer compared to DRL non-deployment issues. However, deployment in game engines has a lower $\%nAA$ (66%) than DRL non-deployment issues.

Figure 4.4 shows the boxplot of the RT required to get an accepted answer for DRL deployment and non-deployment-related questions. The median response time (MRT) for deployment questions (22.58 hours) is 2.6 times that of non-deployment questions (8.83 hours), demonstrating that DRL deployment questions are more difficult to answer. Furthermore, the interquartile range (IQR) of RT for non-deployment questions is 48.7 compared to 156 for deployment questions, indicating a higher spread for deployment questions. More specifically, IQR of RT for server/cloud, mobile, and game engine platforms is 152.8, 247, and 261.3 respectively, as RT in these platforms is more spread than RT in DRL non-deployment questions. However, for DRL deployment on browsers, MRT and IQR are lower (3.1 and 7.4, respectively) than other platforms and non-deployment questions.

Findings: We found that questions about DRL deployment are increasing rapidly and gaining attention from the SO community. They are also more challenging to resolve than other issues of DRL system development.

4.3 RQ2: Taxonomy of Challenges

Table 4.1 shows the taxonomy of DRL deployment challenges across four platforms. The taxonomy includes four sub-taxonomies that categorize challenges of deploying DRL systems

Table 4.1 DRL deployment challenges and their categories. “Description” gives an overview of the category; “S/C”, “M/E”, “G.E”, “B” and “A” show the % of each category in Server/Cloud, Mobile/Embedded, Game Engine, Browser, and All Platforms, respectively.

Challenge	Description	S/C (%)	M/E (%)	G.E (%)	B (%)	A (%)
General questions		19.1	18.8	19.8	26.7	20.2
<i>Whole deployment process</i>	Generic concerns regarding the entire deployment phase.	5.3	5.8	5.5	6.7	5.6
<i>Conceptual questions</i>	Concerns about DRL deployment concepts/background knowledge.	9.9	13	12.1	20	12.3
<i>Limitations of platforms/frameworks</i>	Limitations observed in DRL deployment platforms or frameworks.	3.9	-	2.2	-	2.2
Deployment infrastructure		23	17.4	26.4	11.1	21.3
<i>Procedure</i>	Concerns with process accomplishment (usually "how" questions).	1.3	10.1	3.3	4.4	3.9
<i>Configuring the environment</i>	Concerns related to preparing the deployment infrastructure.	7.2	-	-	-	3.1
<i>Installing/setting libraries</i>	Concerns about installing/building key libraries for the infrastructure.	5.3	7.2	16.5	-	7.8
<i>Managing resources</i>	Concerns about managing/optimizing resources like RAM and CPU.	2.6	-	-	-	1.1
<i>Monitoring</i>	Concerns about tracking important deployment infrastructure metrics.	2	-	-	-	0.9
<i>Multithreading</i>	Concerns related to multithreading tasking in deployment.	4.6	-	-	-	2
<i>Library incompatibilities</i>	Concerns about managing compatibility between libs while deploying.	-	-	6.6	-	1.7
<i>Asynchronous behavior</i>	Concerns about platforms' async. nature, e.g., Async/Await in browser.	-	-	-	6.7	0.8
Data processing		9.9	13	9.9	20	11.8
<i>Procedure</i>	Concerns with data processing (usually "how" questions).	2	1.4	2.2	6.7	2.5
<i>Setting size/shape of input data</i>	Concerns about wrong size or shape of data during inference.	7.9	11.6	7.7	13.3	9.3
RL environment		13.8	11.6	11	24.4	14
<i>Procedure</i>	Concerns with RL environment (usually "how" questions).	0.7	4.3	5.5	8.9	3.6
<i>Configuring the RL environment</i>	Concerns related to preparing and configuring the RL environment.	5.3	-	3.3	-	3.1
<i>Installing/setting libraries</i>	Concerns about installing or building key libraries (e.g., Gym [129]).	7.9	7.2	-	15.6	6.7
<i>Library incompatibilities</i>	Concerns about managing compatibility between libs while deploying.	-	-	2.2	-	0.6
Communication		4.6	1.4	12.1	4.4	5.9
<i>Procedure</i>	Concerns with the communication process (usually "how" questions).	1.3	-	2.2	2.2	1.7
<i>Connection lost</i>	Concerns with remote DRL components losing connection.	3.3	-	5.5	-	2.8
<i>Configuring remote settings</i>	Concerns about settings up remote components (e.g., camera sensor).	-	-	3.3	2.2	1.1
<i>Client/server interaction</i>	Concerns about ensuring a remote client/server data interaction.	-	-	1.1	-	0.3
Agent loading/saving		8.6	14.5	15.4	6.7	11.1
<i>Procedure</i>	Concerns with the loading/saving process (usually "how" questions).	4.6	4.3	6.6	-	5.1
<i>Library incompatibilities</i>	Concerns about managing compatibility between libs while deploying.	3.9	5.8	3.3	-	3.6
<i>Configuring frameworks/libraries</i>	Concerns about fwk/lib configuration preventing agent loading/saving.	-	4.3	3.3	-	1.7
<i>Model conversion</i>	Concerns about converting models from one format to another.	-	-	2.2	-	0.7
Performance		6.6	10.1	5.5	-	6.2
	Category covering performance concerns while deploying DRL sys.					
Environment rendering		13.2	-	-	-	5.6
<i>Procedure</i>	Concerns with the rendering process (usually "how" questions).	3.3	-	-	-	1.4
<i>Configuring rendering components</i>	Concerns regarding configuring libraries to allow rendering.	5.9	-	-	-	2.5
<i>Installing/setting libraries</i>	Concerns about installing/building libraries for environment rendering.	3.9	-	-	-	1.7
Agent export		-	13	-	-	2.5
<i>Procedure</i>	Concerns with the agent exporting process (usually "how" questions).	-	4.3	-	-	0.8
<i>Model conversion</i>	Concerns about converting models from one format to another.	-	5.8	-	-	1.1
<i>Model quantization</i>	Concerns about quantization (e.g., quantized model in other fwks).	-	2.9	-	-	0.6
Request handling		1.3	-	-	-	0.6
	Category covering concerns about making requests in the client side.					
Continuous learning		-	-	-	6.7	0.8
	Category covering CL's concerns (learn continually without forgetting).					

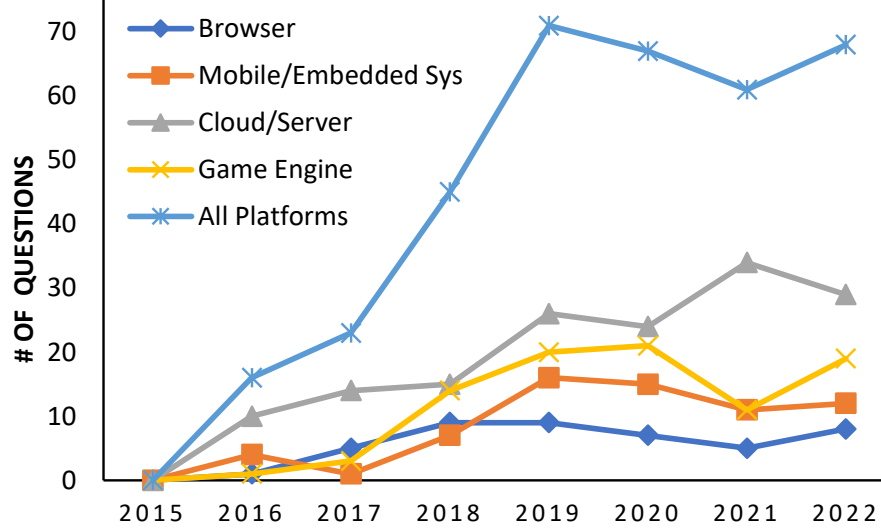


Figure 4.2 Number of Questions per Year.

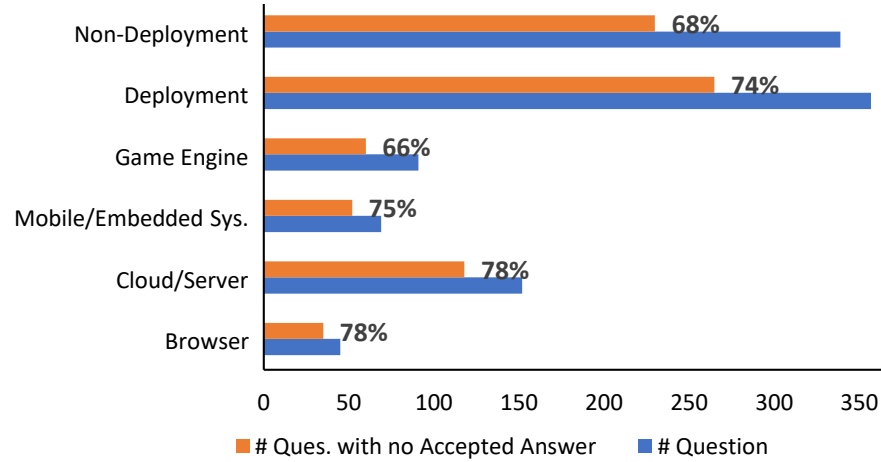


Figure 4.3 Number of Question with No Accepted Answer.

to server/cloud, mobile/embedded devices, browsers, and game engine platforms. Each sub-taxonomy has two levels: the category of challenges (e.g., deployment infrastructure), and the challenge (e.g., monitoring). The taxonomy covers 11 unique categories and 31 unique challenges over the 4 platforms. Finally, our replication package includes the tree-structured taxonomy figures.

4.3.1 Common Challenges to all platforms

To prevent repetitions, we first present the common categories over all platforms and their challenges.

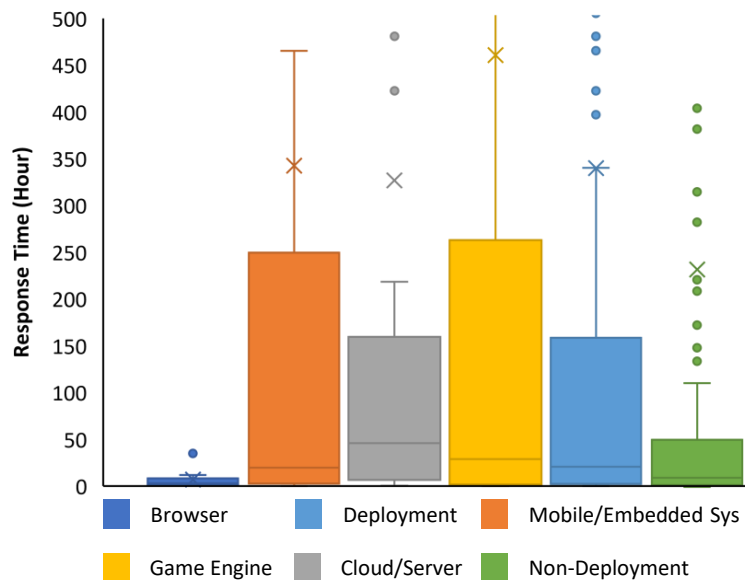


Figure 4.4 Time Needed To Receive an Accepted Answer.

General questions

This category includes general issues not limited to a particular deployment stage and includes three challenges.

Whole deployment process. This challenge describes general concerns about the entire deployment phase. These are generally “how” questions, like “How do I deploy the deep reinforcement learning neural network I coded in Pytorch to my website?” [130]. Developers often ask about general guidelines to use in a specific case (e.g., [131]). Answers usually provide tutorials, documentation-like material, or a list of generic steps to achieve the desired goal. This challenge has 5 to 7% of questions on all platforms.

Conceptual questions. This challenge includes questions on fundamental concepts or background knowledge regarding DRL deployment, such as “What is the easiest 2D game library to use with Scala?” [132]. It accounts for 9.9%, 13%, 20%, and 12.1% of questions in server/-cloud, mobile/embedded systems, browser, and game engines respectively. This suggests that even the fundamentals of DRL deployment can be difficult for developers.

Limitations of platforms/frameworks. This challenge is about the Limitations of deployment platforms and frameworks. For example, in this post [133], the SO user reported a Unity ml-agents problem. The engineer assigned to this bug apologizes for the inconsistent documentation, which was later rectified.

Data processing

This category discusses the difficulties encountered while shaping raw data into the format required by the deployed DRL system. This category accounts for 20%, 13%, 9.9%, and 9.9% of the browser, mobile/embedded system, game engine, and cloud/server questions respectively. In the following, we list common challenges in this category.

Procedure. Procedure covers questions about a specific deployment task, as opposed to the "Whole deployment process" which covers questions about the whole deployment phase. "Unity ML Agent, CameraSensor - Compression Type" [134] is an example in which the SO user asked about the model input when using the unity-trained model in a real-world setting. In the remainder of the paper, we do not duplicate the Procedure descriptions in other categories due to the page limits.

Setting size/shape of input data. Setting the size/shape of data is a typical challenge in data pre-processing. When the input data has an unexpected size/shape during the inference step, improper behavior occurs. In this SO post [135], for example, the user is attempting to take a model built by Unity ML-Agents and run inferences using Tensorflow, where she encountered a shape mismatch error.

Deployment infrastructure

This category includes concerns in preparing the deployment infrastructure for DRL systems. This category has the largest ratio of questions in game engine platforms and cloud/server platforms, with 26.4% and 23%, respectively. It also accounts for 17.4% and 11.1% of questions in mobile/embedded systems and browsers respectively. The challenges shared by all platforms in this category are:

Configuring the environment. When deploying DRL systems, developers must set up several environment variables, paths, and configuration files, all of which have numerous choices, making configuration difficult. Problems that arise during this stage are addressed under this challenge.

Installing/setting libraries. Furthermore, developers must install or set essential libraries to prepare the deployment infrastructure. This type of concern is addressed in this challenge.

Library incompatibilities. Some developers have trouble using libraries while deploying DRL systems. Indeed, the continuous growth of libraries makes version compatibility of frameworks/libraries difficult for developers. For example, a SO post reported an error that was triggered by Tensorflow 1.7.0 having an incompatible version with the Unity3d ml-agents used version [136].

RL environment

This category includes challenges in preparing the RL environment for deployment. It comprises nearly the same challenges as the Deployment infrastructure category. However, we consider a question related to the RL environment when a wrong behavior occurs during attempting to prepare the RL environment, rather than the deployment infrastructure. The Deployment Infrastructure topic covers broader issues related to the deployment ecosystem. For instance, an issue with configuring Docker containers for deployment is considered a Deployment Infrastructure challenge [137], whereas a concern with configuring the Gym environment is considered an RL environment challenge [138]. We find a noticeable variation in patterns between these two categories after establishing this distinction. With a question ratio of 24.4%, the RL environment category is the second-highest among browser platforms. It also accounted for 11.6%, 11%, and 13.8% of questions in the mobile/embedded system, game engine, and cloud/server platforms, respectively.

Communication

This category addresses issues with communication between DRL system components. For example, in this SO question titled "How to ensure proper communication between a game in C++ and an RL algorithm in Python?" the user asks about a method to ensure proper communication between a TensorFlow/Keras-based RL agent and a C++ game. This category accounts for 4.4%, 1.4%, 12.1%, and 4.6% of the browser, mobile/embedded system, game engine, and cloud/server questions respectively.

Agent loading/saving

This category contains challenges with saving a DRL agent from one platform/framework and reloading it in another platform/framework. First, developers may encounter difficulties converting models from one format to another in order to use them on a specific platform (e.g., [139]). Furthermore, incorrect setup of a certain framework or library may hinder the process of loading or storing a DRL agent to the required platform. In AWS Sagemaker, for example, a faulty setup prohibited a user from recovering an agent's trained DNN [140]. Finally, incompatibility across frameworks or libraries may pose an extra barrier when loading or storing a DRL agent [141]. The category of agent loading/saving accounts for 6.7%, 14.5%, 15.4%, and 8.6% of questions in the browser, mobile/embedded system, game engine, and cloud/server platforms respectively.

Performance

When deploying DRL systems, performance is a critical factor that developers must address. Execution time, latency, and hardware consumption are all critical factors that could affect the DRL system in production. This category includes all performance issues that may arise while deploying DRL systems. These issues might arise primarily in two places: the environment and the DRL agent. In one SO post [142], for example, the user is attempting to run a large number of simulated environments in parallel using Google Cloud Kubernetes Engine. However, the simulation on her development device is twice faster than the one on Kubernetes. In another SO post [143], the user is attempting to benchmark two Google Coral [144] devices for the deployment of a DRL agent. She was disappointed since the Coral device is significantly slower than the CPU. This category comprises 10.1%, 5.5%, and 6.6% of questions in mobile/embedded systems, game engines, and cloud/server platforms, respectively.

4.3.2 Other Challenges in Browser

Continuous Learning. Continuous Learning (CL) is the model’s ability to continually learn and adjust its behavior in real time. DRL systems’ trial-and-error nature makes CL the go-to approach for adjusting the DRL agent to rapidly changing conditions in production. This category addresses concerns regarding using CL in production with DRL systems. In this SO post, [145], for example, the user asks about a method to continually adapt a trained bot to emulate a real player. This category accounts for 6.7% of browser platform questions.

4.3.3 Other Challenges in Mobile/Embedded System

Agent Export. Agent export can be achieved by (1) directly converting its trained model into the required format or by (2) using dedicated frameworks to transform the model to a format that runs on the deployment platform. In mobile/embedded system platforms, agent export accounts for 13% of all questions. Agent exporting is essential when deploying DRL agents on edge platforms (e.g., mobiles) due to their limited resources and different operating systems. In the following, we discuss challenges within this category.

Model Conversion. It includes issues associated with any setting misbehavior or incorrect use of model conversion frameworks (e.g., ONNX) [146].

Model Quantization. Quantization lowers the accuracy of model parameters’ representation in order to minimize the memory footprint of DNNs. In this challenge, developers often struggle with (1) combining quantization frameworks like TF Lite with other frameworks or

platforms [147] or (2) working with varied precision floating points [148].

4.3.4 Other Challenges in Server/Cloud

Request handling

This category includes issues with making requests on the client side and accounts for 1.3% of cloud/server platform questions. Developers struggle mainly with customizing the request body [149] and obtaining information on serving models through request.

Environment rendering

This category discusses issues with rendering the environment while running DRL systems on a server or in the cloud. It accounts for 13.2% of cloud/server questions (third highest). Developers, in this category, struggle to configure frameworks and/or libraries to allow rendering in headless servers [150, 151].

4.3.5 Survey Results

Table 4.2 presents the validation survey findings. We show the percentage of participants who replied “yes” to whether they faced related challenges. We also show the severity of each challenge category and the reported effort needed to fix it. The survey participants have faced all sorts of challenges, proving the taxonomy’s relevance. With 81% approval, “RL environment” is the most approved category. 65% and 41% of these participants said this category has a high severity and requires high effort, respectively. 19% of survey participants had encountered “Continuous Learning” and “Agent Export”, the least-approved categories. Yet, 50% of these participants believe these categories have a high severity and require high effort. The average approval rate for common challenges to all platforms is 61.2%, indicating that the final taxonomy matches the experience of most participants. The remaining platform-specific challenges average 28.5% “yes” responses. This may be because participants only work on one platform and don’t have experience with all platforms. Finally, some participants provided challenges they thought were not in the taxonomy: *Train/production RL environment gap*: One participant highlighted the challenge of matching the training and deployment RL environment. Indeed, this problem is frequent in DRL deployment literature (e.g., the sim-to-real gap [152]). *Partial observability & non-stationarity*: Another participant commented that non-stationarity makes DRL deployment difficult [60]. For example, recommendation systems in production must adapt to changing user behavior. Finally, one participant’s comment was remarkable. According to her/him, one of the biggest deployment issues is aligning

and expressing DRL-specific challenges (e.g., sim-to-real gap [152]) with teams who don't have much RL/ML knowledge. DRL is different from common engineering practices, making team communication difficult.

Findings: We identified 31 challenges, most of which are common across deployment platforms. The most common challenges are related to the deployment infrastructure, while the RL environment challenge is confirmed by most survey participants.

4.4 RQ3: Analysis of Challenges

Table 4.3 shows the popularity and difficulty of RQ2's categories of challenges. Popularity is expressed by the average views (AV) and average scores (AS) metrics, while difficulty is expressed by the percentage of questions with no accepted answer ($\%nAA$) and the median time (in hours) to receive an accepted answer (MRT). Challenges in Table 4.3 are categorized by the deployment platforms. Platform-common and platform-specific challenges are highlighted in light and dark gray, respectively.

Common Challenges to all platforms. Across all platforms, the most difficult challenge in terms of $\%nAA$ and MRT is "Communication". "Communication" has also the second highest AV , making it a popular and difficult challenge in DRL deployment. The posts on this challenge, however, have the second-worst AS in all platform-common challenges. This high $\%nAA$ combined with low AS may indicate that these posts' questions are unclear or poorly written which decreases the likelihood of an accepted answer.

The most popular challenge in terms of AV is the "RL environment". This challenge has also the third-highest AS , which highlights the importance and popularity of this challenge. Furthermore, in our validation survey, "RL environment" was the most approved challenge. Survey participants agreed that RL environment challenges in production (e.g., sim to real gap [152], non-stationarity [60]) are prevalent.

In addition, "General Question" is the most popular challenge in terms of AS . Interestingly, this category has the lowest MRT . This suggests that even general, easy-to-answer, questions about DRL deployment are often requested.

On the other hand, "Data Processing" has the lowest AV and the lowest AS . Also, it has the second-lowest MRT and the second-lowest $\%nAA$, making this challenge the least popular and second-least difficult. To understand the reason behind this behavior, we examine the posts on this challenge and find that most posts are asking about simple tensor shaping problems. Also, Data Processing is closely related to traditional DL challenges, which could

Table 4.2 Validation Survey Results

Challenge	Resp. Yes(%)	Severity (%)			Effort Required (%)		
		Low	Med.	High	Low	Med.	High
RL Env.	81	12	24	65	12	47	41
Deployment Infra.	48	20	60	20	40	30	30
Data Processing	76	44	38	19	38	44	19
Communication	24	20	60	20	20	80	0
Agent Sav./Load.	71	40	40	20	53	27	20
Performance	67	14	43	43	21	43	36
Continuous Learn.	19	0	50	50	0	50	50
Agent Export	19	25	25	50	25	25	50
Env. Rendering	52	27	36	36	18	64	18
Req. Handling	24	0	100	0	20	60	20

Table 4.3 The popularity/difficulty per topic.

Challenge	Browser				Mobile/Emb. Sys				Game Engine				Cloud/Server				All Platforms			
	POP.		DIFF.		POP.		DIFF.		POP.		DIFF.		POP.		DIFF.		POP.		DIFF.	
	AV	AS	%nA	MRT	AV	AS	%nA	MRT	AV	AS	%nA	MRT	AV	AS	%nA	MRT	AV	AS	%nA	MRT
General Question	486	0.6	67	2.4	740	0.8	85	2.5	2713	9.1	78	25.5	354	0.7	79	8.1	1035	2.8	78	3
Data Processing	203	0.7	78	7	215	0.7	68	64.9	376	0.7	78	1.5	1362	1.1	53	21.6	657	0.8	67	14.2
Deployment Infra.	711	1	80	0.8	1986	2	92	0.1	1159	1.1	63	14.4	791	0.7	83	45.6	1091	1.1	78	37.6
RL Env.	579	0.6	91	0.2	1198	-0.1	75	10.3	533	1.4	50	326.9	2089	2.4	81	32.9	1303	1.4	76	32.9
Communication	1215	0.5	100	N.A	86	0	100	N.A	452	0.4	82	14.5	2330	2.6	71	192.9	1133	1.1	81	98
Agt Sav./Loading	189	0.3	67	34.8	817	0.9	50	340	1289	1.1	50	258.9	725	1.8	69	71.9	905	1.2	58	76
Performance	-	-	-	-	952	3.4	71	84	1391	2	60	474.1	693	1.2	70	50	975	2	68	50
Cont. Learning	235	0.7	67	6.8	-	-	-	-	-	-	-	-	-	-	-	-	235	0.7	67	6.8
Agt Export	-	-	-	-	775	0.9	78	10.4	-	-	-	-	-	-	-	-	775	0.9	78	10.4
Env. Rendering	-	-	-	-	-	-	-	-	-	-	-	-	8039	7.8	90	244.4	8039	7.8	90	244.4
Req. Handling	-	-	-	-	-	-	-	-	-	-	-	-	69	0	100	N.A	69	0	100	N.A
All Challenges	473	0.6	78	3.1	982	1.1	75	19.7	1268	2.6	66	28.8	1946	2.1	78	45.8	1401	1.9	74	20.6

explain why the SO respondents tend to answer this challenge faster.

Challenges specific to one platform. “Environment Rendering”, a specific challenge to Cloud/Server platforms, is the most popular challenge with the highest *AV* and the highest *AS*. This challenge is also the one with the highest *MRT* and the second-largest *%nAA*. This demonstrates that the “Environment Rendering” challenge is among the most popular and difficult challenges in cloud/server-related challenges. The other platform-specific challenges, like “Continuous Learning” and “Request Handling”, are less popular and less difficult compared to the other challenges. This can be explained by the fact that these challenges are very specific and are not faced by developers on a regular basis.

Correlation analysis. Finally, to have a full view of the DRL deployment challenges, we examined if there is a statistically significant correlation between the difficulty and popularity of the assessed challenges. In particular, we use the Spearman Rank Correlation Coefficient [153] to verify the correlations between the two popularity metrics (*AV*, and *AS*) and the two difficulty metrics (*%nAA* and *MRT*). We choose Spearman’s rank correlation since it does not have any assumption on the normality of the data distribution. Results show one statistically significant correlation between *AS* popularity metric and the *MRT* difficulty

metric (coeff=0.47, p-value=0.011). This demonstrates that when considering *AS* and *MRT*, the difficult challenges are popular among developers, and vice versa. The replication package [30] has further details about correlation analysis.

Findings: Across all platforms, RL environment-related challenges are the most popular whereas communication-related challenges are the most difficult. Overall, we observe a significant positive correlation between the popularity and difficulty level of the challenges.

4.5 Implications

We describe, in the following, how our findings may provide insights to practitioners, researchers, and framework developers to improve the deployment of DRL systems. Figure 4.5 illustrates a bubble plot ranking the challenges in terms of their popularity and difficulty. The bubble’s size indicates the proportion of posts for a challenge. The figure’s four quadrants show the challenge’s level of popularity and difficulty. *AV* serves as a proxy for popularity, while *%nAA* serves as a proxy for difficulty. Finally, due to the large number of challenges (31), we removed challenges with less than 10% of posts on each platform.

Implication for Practitioners. Despite being the most popular challenge, questions about “RL environment” remain mostly unresolved (see Figure 4.5). In fact, unlike vanilla DL systems, deploying DRL systems requires a focus on several components like the environment, the agent, and the communication between them. This finding should be embraced by the community to improve tutorials and documentation in order to reduce the burden of deploying DRL systems. Our findings can also assist developers in better prioritizing their effort by addressing the most challenging topics in DRL deployment. Software managers can similarly account for this by allocating more resources and time to DRL-specific tasks. Finally, DRL deployment is based on the interaction between DRL and Software Engineering. As a result, DRL deployment necessitates developers that are knowledgeable in both domains, making it a difficult task. Our taxonomy may be used as a checklist for developers, encouraging them to obtain the essential skills before deploying DRL systems.

Implication for Researchers. As revealed in our study, DRL deployment is growing in popularity among developers, but developers face a wide range of challenges, many of which are DRL-specific (e.g., RL environment challenges). Moreover, as seen in Figure 4.5, DRL-specific challenges are particularly hard to solve. Thus, researchers are urged to propose approaches and solutions to assist developers in addressing these deployment

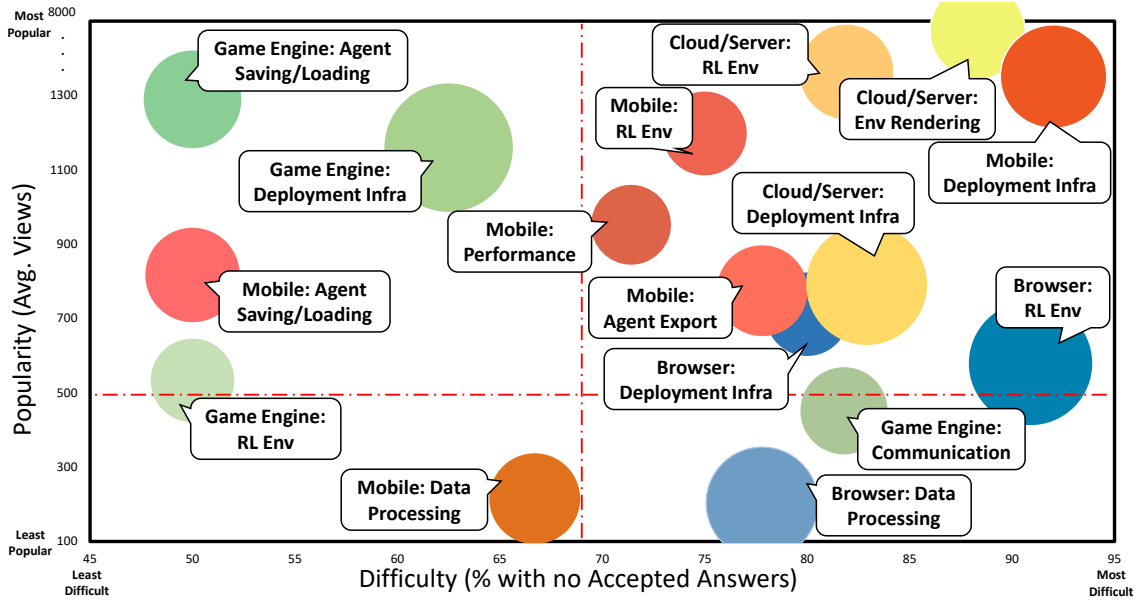


Figure 4.5 Bubble plot of topic popularity and difficulty.

challenges, such as Configuration. Several challenges in our taxonomy are connected to configuration since DRL incorporates many interacting components thus many configurable elements. This insight can inspire researchers to provide automated configuration solutions to make various deployment tasks easier for developers. In addition, rule-based verification may be introduced, as they are quite useful for detecting and diagnosing misconfigurations. Likewise, strategies for monitoring and notifying developers about potential issues throughout the deployment process might be introduced. Monitoring the deployment process is a difficult problem in DRL [60, 154, 155] that requires paying attention to the variety of potential root causes, which include hardware and software failures, as well as concept and data drifts [156].

Implication for Framework Suppliers. According to our findings, many developers struggle with the whole deployment process. Indeed, developers frequently highlight the poor quality or the lack of documentation in these questions (e.g., SO post [133]), indicating that the documentation should be enhanced. Additionally, the predominance of the “Conceptual questions” category (which hit 20% of posts in Browser) implies that framework suppliers should increase their documentation’s completeness, especially given that DRL deployment necessitates a diverse range of knowledge and expertise. Figure 4.5 also demonstrates that the “Deployment infrastructure” is the most prevalent challenge in our taxonomy in terms of number of posts. The figure also shows this challenge’s difficulty and popularity, as it appears in the figure’s most popular/difficult quadrant on three of four platforms. These findings may be used to encourage better and more intuitive end-to-end DRL frameworks. End-to-end frameworks such as MLflow [157] are gaining popularity in the ML community

and we are starting to witness initiatives aiming at “productionizing” DRL utilizing these technologies (e.g., Spark [158], MLflow [159]). Nevertheless, the majority of these frameworks do not pair well with DRL or do not support it at all. We believe that our study might push toward better and improved DRL-specific end-to-end frameworks.

4.6 Threats to Validity

One potential threat is the selection of specific tags and keywords to identify relevant posts. Our automatic collection process may be biased by the pre-selected tags and keywords. To mitigate this threat, we chose popular frameworks and platforms to ensure representativeness. However, the keyword-matching collection process may include false positives or exclude posts without pre-selected keywords.

Another potential threat is the reliance on SO as a single data source for studying developer challenges. While we retrieved a fair number of relevant posts, it is possible that we overlooked valuable insights from other sources. However, we believe that our results are still relevant owing to the diversity of SO users, including both novices and experts.

Finally, the subjectivity of the manual analysis is a potential threat to the validity of our work. To mitigate this risk, the first two authors individually inspect posts and reached an agreement. They also had the assistance of a third expert author when there is a conflict. The inter-rater agreement is substantial confirming the labeling procedure’s reliability.

4.7 Chapter Summary

This work proposes an empirical study on SO to understand the challenges practitioners faced when deploying DRL-based systems. We examined 357 SO posts about DRL deployment and found that the general interest in DRL deployment is growing. Our findings also reveal that DRL deployment is harder than other parts of DRL systems development, pushing us to investigate its specific challenges. To that end, we build a taxonomy of 31 unique challenges for deploying DRL to server/cloud, mobile/embedded systems, game engines, and browsers. We hope this study stimulates future research and helps the community solve DRL-based system deployment’s most prevalent and difficult challenges. In future work, we plan to broaden our data sources and interview experts and practitioners to further confirm and expand our findings.

CHAPTER 5 DIVERGET: A SEARCH-BASED SOFTWARE TESTING APPROACH FOR DEEP NEURAL NETWORK QUANTIZATION ASSESSMENT

After the introduction of the ImageNet dataset [160], Deep Neural Networks (DNNs) have gotten a lot of attention from academia and industry. In fact, over the last years, DNNs achieved state-of-the-art results in many tasks, such as image recognition [161], natural language processing [162], and speech recognition [163].

Recently, there has been a constant movement of Artificial Intelligence (AI) towards edge devices. The on-device inference is the process of making predictions using a trained model that is running on the device. This trend offers numerous advantages, mainly the decrease in inference latency. In fact, by bypassing the data upload to the server and the wait time for the inference result, the application is able to respond to the user’s request faster. Besides, discarding the server reliance has other benefits, like being able to operate with little to no connectivity, and lowering privacy concerns as the user’s data remains on the device. However, one limitation that hinders DNN models from being widely deployed on edge is their resources consumption. In fact, they require a lot of memory and might compromise the battery life of devices during training and inference. To that end, the scientific community and the businesses involved in AI are constantly investing in developing and implementing methods and technologies to enable high-performance on-device DNN inference.

Quantization is one of the most applied strategies to infer DNNs on edge devices. This is mainly due to its simplicity, straightforward implementation, and applicability to a wide range of scenarios, compared to hardware-specific solutions (e.g., Google Coral Edge TPU¹). In fact, quantization decreases DNN’s memory requirements by lowering the precision of the values representing weights and activation from a 32-bit floating-point format to a much smaller one. With the industry’s rising demand for quantization, the efficacy of the assessment of quantized DNNs is becoming an essential concern. To the best of our knowledge, no studies proposed dedicated techniques to assess the efficacy of these quantized models. Instead, the classical approaches that are commonly used to assess vanilla (full-size) DNN models, are also adopted to assess their quantized versions.

Classical assessment methods judge a given DNN based on its accuracy on test data. When applied to a quantized DNN, they also judge its ability to preserve the performance of its original version. Even though this test data-driven assessment method has proved its ef-

¹<https://coral.ai/products/accelerator>

fectiveness in evaluating DNNs, it might not be the best option to judge the impact of quantization on these models. In fact, test data are a set of labeled samples that give an insight on the nature, the features, and the properties of the processed data. In many domains, these samples are very hard to collect as they require expensive, time-consuming, and expert-depending procedures. This leads to smaller test datasets where the provided labeled samples might not properly represent the immensity of the input space. Besides, test data might not reflect every possible scenario that the model can encounter. In the same sense, they may not be able to trigger every possible divergent behavior between the DNN model and its quantized version. Hence, if we only rely on these traditional assessment methods, we may be led astray as two DNN versions of different arithmetic precision might show almost the same level of performance on test data, but trigger widely divergent behaviors when applied in real-world settings.

In this chapter, we address these challenges by proposing DiverGet, a customizable search-based software testing framework dedicated to quantization assessment. DiverGet aims to evaluate the real impact of quantization methods by detecting difference-inducing inputs that produce behavioral disagreements between an original DNN and its quantized version. To do so, DiverGet (i) extends the search space using semantically preserving metamorphic relations, then (ii) explores the search space following population-based metaheuristics, and (iii) uses diverse and complementary fitness functions to guide the search process and generate quality test data.

To demonstrate the effectiveness of our proposed DiverGet, we evaluate it on Hyperspectral Images (HSIs). These cubes of data are made of hundreds of bands that collect, each at a narrow interval of contiguous wavelengths, the energy emitted and/or reflected by the objects of the captured scene. Each band offers spatial information, and the stacked bands give us spectral information. This very rich information is extremely vulnerable to distortions caused by external factors. Besides, these rarely labeled data are frequently used in quantization-related applications.

Results using state-of-the-art HSI DNNs, namely the Hybrid Spectral Convolutional Neural Network (HybridSN) [1] and the Spectral–Spatial Residual Network (SSRN) [2], show that DiverGet succeeds in generating meaningful test inputs that induce behavioral disagreements among DNN versions. DiverGet also succeeds in outperforming the state-of-the-art testing framework, DiffChaser [91], in detecting divergences induced by quantization.

Chapter Overview. The remainder of this chapter is structured as follows: Section 5.1 introduces DiverGet and describes its design process. Section 5.2 reports the evaluation outcomes. Section 6.4 analyzes the threats to the validity of our proposed DiverGet. Finally,

Section 5.4 concludes the chapter.

5.1 Methodology

In this section, we introduce our proposed approach for DNN quantization assessment and describe its principal aspects, namely the domain-specific metamorphic relations and the search-based data transformation.

5.1.1 Quantization Assessment via Systematic Detection of Behavioral Divergences

DiverGet’s protocol for quantization assessment is designed to systematically search and expose behavioral divergences by showing that a quantized DNN behaves very differently on metamorphically-transformed inputs. DiverGet probes the quantized model’s erroneous behaviors w.r.t. its full-precision counterpart on practically-relevant data distortions. The main point is that our crafted synthetic inputs induce differences between the DNNs’ inference behaviors, not simply conventional adversarial attacks that provoke mis-predictions by the quantized version. This variation distinguishes difference-inducing test inputs from the more familiar fault-inducing adversarial inputs. This strongly implies that techniques generating random noisy data with no true label, maliciously-perturbed inputs with ambiguous semantics, or highly-stretched data risking out-domain distribution, are insignificant for difference-inducing test exploration. The first reason is that even state-of-the-art models still suffer from these adversarial search techniques mainly because of their over-parameterization and high sensitivity to certain well-crafted changes. Thus, the resulting disagreement would not provide novel insights on the preserved robustness following the quantization since the original model would often predict a wrong output. The second reason is that it is less interesting if the two versions of the DNN disagree on semantically-ambiguous and practically-irrelevant synthetic test inputs. In fact, the reduction of the arithmetic precision in the quantized DNN would naturally induce numerical calculation differences reaching the final output, in certain subspaces of the input data.

In response to that, we propose to elaborate domain-specific metamorphic relations, that we later introduce in Section 5.1.3. This allows realizing more task-oriented performance comparisons of the two DNN versions against input distortions that arise naturally from certain condition changes in real-world deployment settings.

The systematic aspect of our approach is established by the variation in the distortion type and its level of intensity within the permitted range. However, given the high-dimensionality

of this derived synthetic input space, the trade-off between the cost of the assessment and the coverage of behavioral divergences should be well balanced and configurable according to the safety-critical aspect of the application or the user’s preferences.

On one side, we adopt a search-based approach (please refer to Section 5.1.4) that leverages metaheuristics to optimally drive the data transformations towards the production of difference-inducing inputs. We design two alternative fitness functions that would serve as different search objectives, depending on whether fast reaching the divergences or diversifying the triggered behaviors is the priority. On the other side, we choose population-based metaheuristics as the principal type of search algorithm in our assessment because of their exploration capabilities. Their parameters regarding the size of population and the number of iterations would be chosen in advance to control the amount of distortions generated by every session.

Furthermore, our protocol supports two modes, either sampling one original datapoint or a batch of datapoints, to apply on them the generated distortions. The batch mode reduces the cost of test generations as low as we increase the size of the batch, but the search for interesting distortions on a single original datapoint can be more focused, and hence, more effective.

Last, the revealed set of behavioral divergences between the original and the quantized models can be an additional quality criterion to compare between different quantization techniques, and can influence the decision-making process on the real-world deployment of the quantized model at the edge.

Figure 5.1 shows the overview of the systematic process of the proposed quantization assessment approach.

5.1.2 Search for Transformations instead of Data

Due to the high dimensionality of HSIs, we chose to change the search space by searching for data transformations, i.e., vectors encoding the steps of transforming an original HSI into a distorted HSI, rather than searching directly for HSIs. The transition from highly-dimensioned data to lower-dimensioned transformation simplifies the search problem by narrowing the search space and simplifying validation checks. While we cannot define the search space of all valid neighbors of an original image, we can constrain the parameters of the transformations with the aim of systematically promoting valid transformed inputs as results.

The transformation vectors are the backbone of DiverGet’s workflow and can efficiently and easily help produce the ultimately needed distorted HSIs. The vectors that result in

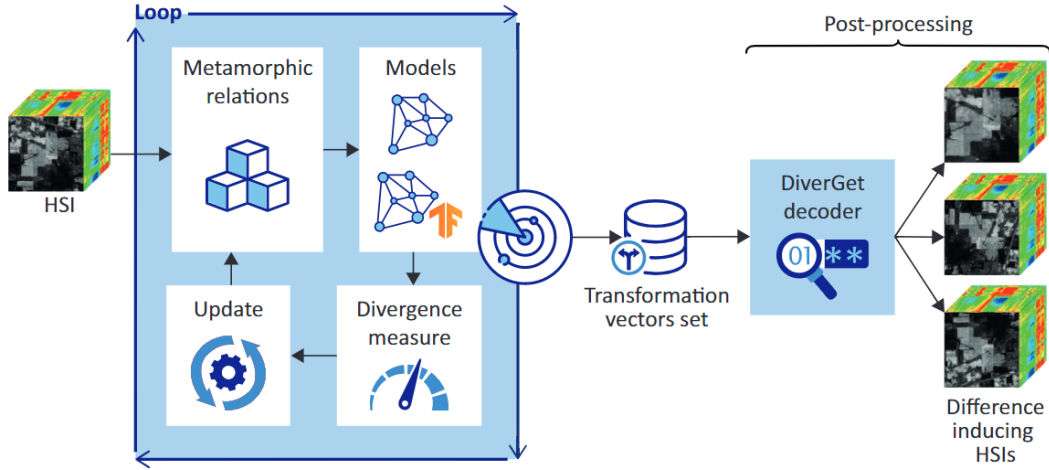


Figure 5.1 The workflow of DiverGet: starting from a given HSI (or a batch of HSIs), our proposed framework creates a set of transformations vectors. These latter can be used, in a post-processing step, to generate difference-inducing HSIs.

Difference-Inducing Inputs (DIIs) will be finally stored by DiverGet. These vectors provide all the necessary information (i.e., metadata) to generate DIIs in a post-processing stage where the user can easily utilize DiverGet’s decoder, as illustrated by Figure 5.1. Generating and exploiting DIIs might be a beneficial and rapid approach for quantized model repair in comparison to repeating the training from scratch. In fact, current research has begun to investigate several strategies that leverage DIIs to fix the divergence between the two DNN versions [164] (i.e., original and quantized versions).

The benefits of the transformation vectors are not limited to the saving of the storage space and the reconstruction of synthetic inputs. In fact, the valuable information encoded in these vectors may be exploited to gain insights into the causes of divergence between the model versions. In particular, statistics on these vectors can yield further explanations for the divergence pattern. For instance, a simple study of these vectors may indicate which transformations result in more divergences, or what correlations exist between them. These findings might be extremely useful to developers, however without transformation vectors, they would be difficult to pinpoint.

DiverGet’s purpose is to revisit the quantization assessment by adding the evaluation of the robustness of the quantized version against naturally-occurring input distortions, with respect to its original counterpart. Thus, tracking all the revealed DIIs is essential to estimate the overall divergence rate. This latter can be considered as a novel robustness measure that is complementary to other conventional metrics like the success rate that detects the exis-

tence of divergence between two DNN versions with different arithmetic precision. Searching for all possible DIIs may raise some scalability concerns. However, the continuous test of label differences (i.e., checking if the MR was violated) has a very low overhead since these labels were previously calculated to estimate the fitness values. The time complexity of this operation will be $O(1)$ and thus does not hinder the scalability of DiverGet. Regarding the scalability of the storage, our tracking module follows a periodic offloading strategy to avoid high memory usage. In fact, we save periodically the difference-inducing transformations into the hard drive after reaching a pre-fixed size. In our tests, we set the size to 10 Mb in order to balance between the I/O operations and the memory consumption.

5.1.3 Domain-specific Metamorphic Relations

Given a labeled dataset, $\mathcal{D} = (X_i, y_i) \quad \forall i \in \{1 \dots N = |\mathcal{D}|\}$, a model m_o and its quantized version m_q , we design a space of semantic-preserving Metamorphic Relations (MRs), $\mathcal{R}$. Each MR, $r \in \mathcal{R}$, would map a compound HSI distortion, $T_r \in \mathcal{T}$, to the identity of the expected predictions, formulated as $r : (T_r(\cdot, \theta_r), \mathbb{1})$, where θ_r are the parameter values of T_r within their valid set, Θ . Thus, the follow-up test consists of asserting that:

$$(m_o(T_r(X_i)) = m_q(T_r(X_i))) \wedge (m_o(X_i) = y_i), \quad \forall i \in \{1 \dots N\} \quad (5.1)$$

Hence, a failed test would yield a synthetic input, $\hat{X}_i = T_r(X_i)$, that violates a MR, r , and will be considered as a DII.

MRs have been extensively leveraged to uncover anomalies of different generations of software systems. They define a relationship between input data transformations and their corresponding expected outputs. The strength of this simple concept is that well-designed MRs help automating the large-scale generation of test cases without human intervention.

In the following, we describe the steps and principles adopted to construct MRs that relate semantically-preserving HSI transformations to the identity relationship between the expected outputs. It is worth mentioning that the design of HSI transformations and their configurations were performed in collaboration with domain experts.

- (i) Codifying naturally-occurring distortions as probabilistic data perturbations:

In contrast to traditional color images made of three bands, remote sensing HSIs are large cubes composed of hundreds of spectral bands. Each of the latter records, at a narrow interval of contiguous wavelengths, the radiations reflected and/or emitted

by the objects of the captured scene. These images offer extremely rich spatial and spectral information. Nevertheless, they are vulnerable to atmospheric changes and acquisition system’s defects that often induce partial information losses or alterations, in a chaotic or a regular way.

Through collaboration with two HSI experts who have been working on remote sensing image processing since 1994 and 2015 [165, 166], respectively, we implemented distortions that mimic naturally-occurring alterations [167] caused by instabilities or defects in acquisition systems, or by atmospheric influences. Domain experts [167] project these anticipated degradations of information, affecting HSIs, into groups, including radiometric perturbations that alter the luminance values of the pixels, and geometric transformations that change the geometric structure of the image.

For classification requirements, HSIs are cropped into multiple overlapped 3D patches constituting the data inputs for DNNs. Numerically, a 3D patch is a tensor, $\mathbf{X} = (x_{i,j,k}) \in \mathbb{R}^{d_1 \times d_2 \times d_3}$, with 2 spatial dimensions, d_1 and d_2 , and one spectral dimension, d_3 . We denote by $\mathbf{X}^{dis} = (x_{i,j,k}) \in \mathbb{R}^{d_1 \times d_2 \times d_3}$, the distorted 3D patch.

Table 5.1 summarizes the details of the distortions we implemented and used throughout this chapter. In this table, distortions are applied to randomly picked pixels, $x_{(i',j',k')}$. Furthermore, we emphasize that all the random values of any distortion parameter (e.g., mean of Gaussian noise, rotation angle), as well as all the arbitrary selections of targeted components (e.g., region, line, pixels) would be either uniformly sampled or optimally set by the optimizer, while always remaining within the possible range of values or set of choices.

(ii) Pre-define and set up the appropriate range of distortions:

The range of possible values for each transformation’s parameters would decide the resulting levels of distortions on the original data. In fact, a wide range would result in high loss of information that have high chances to produce meaningless inputs that can be filtered upfront, i.e., images that are hard to occur in real-world settings. Inversely, a narrow range would be too conservative with meaningful inputs that are very close to their original parent and have low chances to trigger uncovered behaviors of the DNN. Hence, it is important, in a pre-processing phase, to set up the appropriate range of distortions that balance between enhancing the diversity and preserving the semantic identity of the synthetically-crafted images.

To do so, we use the Peak Signal-to-Noise Ratio (PSNR) [168], a widely-used metric in the HSI field, to assess how much information loss occurs after the distortion. Thereby,

Table 5.1 Presentation of the used HSI distortions

Distortion	Variants	Short Description
Continuous Drop out	Line, Column, or Region	Replace the intensity values of <i>all</i> the pixels belonging to a random component (e.g., line, column) by either the maximum or minimum intensity value within the 3D patch.
Discontinuous Drop out	Line, Column, or Region	Replace the intensity values of <i>some</i> of the pixels belonging to a random component (e.g., line, column) by either the maximum or minimum intensity value within the 3D patch.
Stripping	Line or Column	Update the pixels belonging to a random component, $x_{i',j',k'}$, using the formula $\frac{\sigma_d}{\sigma_o} \cdot (x_{i',j',k'} - m_o + \frac{\sigma_o}{\sigma_d} m_d)$, where (m_o, σ_o) and (m_d, σ_d) are the pair (mean, standard deviation) of the original and distorted patches, respectively.
Spectral band loss	NA	Replace an arbitrary subset of band values with the mean of their successor and predecessor.
Salt and pepper noise	NA	Scatter arbitrary bright pixels (salt) or dead pixels (pepper) using, respectively, the maximum or the minimum value in the 3D patch.
Gaussian noise	Spectral or Spatial	Add a random noise ϕ that follows a Gaussian distribution ($\phi \sim N(\mu, \sigma)$), to either the spectral or spatial information of arbitrary pixels.
Rotation	NA	Rotate the patch around its central pixel, by a randomly-picked angle α .
Zoom	In or Out	Perform in or out zooming centered on the central pixel of the patch according to an arbitrary zoom factor.

we adjust the range of parameters of each codified distortion to reach or be close to their appropriate ranges. We start by the full range of possible values, then, we narrow down the range if the PSNR of the produced inputs are generally low. The significant degradation of the obtained PSNRs indicates that the distorted images are becoming out of the original data distribution.

(iii) Ensure the validity/meaningfulness of the distorted inputs:

Despite the tuning of the ranges of the distortions' parameters, there is no guarantee that the synthetic images would be in-domain distribution and preserve the semantic

identity of their genuine parents, especially when multiple distortions are applied in a compound transformation.

To ensure the meaningfulness of the generated distorted inputs, we opted for: (i) a *conservative strategy*: We adhered to expert guidelines throughout the conception of our synthetic data to ensure plausible real-world application. For instance, we constrain radiometric distortions to either be column- or row-wise due to the type of acquisition system, which collects data transversely, either by columns or by lines.

Furthermore, we set up a (ii) follow-up post-distortion validity test where we reused the PSNR for the second time. If the PSNR of distorted patches is less than 20dB (experts' recommendation), they are discarded as invalid inputs. In fact, PSNR is utilized in HSI compression to verify the image quality and ensure semantics are preserved [169, 170]. Thus, we apply the same validity check for our synthetic test inputs, using the default threshold of 20 dB as stated in Section 5 of [171].

5.1.4 Search-based Approach for Data Transformation

Once we have the domain-specific metamorphic relations, we aim to systematically search for the pairs of (original datapoint, crafted distortion) that enhance the diversity of the divergence-oriented test cases and also increase their ability to reveal disagreements between the best-fitted DNN and its corresponding quantized variants. Below, we detail the development steps that we follow to build the underlying search-based approach.

Definition of the Search Space

Given the complexity of our codified input distortions, it is difficult to formulate the space of possible distorted versions from an original datapoint, contrary to other SBST for quantization assessment that restrict the data transformation to pixel-value noise addition (i.e., the search space for an input x would be $[x - \delta, x + \delta]$, where δ is the maximum allowed pixel-value deviation). Thus, we choose to define the space of possible data transformations, then, the synthetic test inputs would be the result of the application of the crafted data transformations, and an original input data. To do that, we stack (i) the parameters in relation to all the supported data transformations, (ii) the binary variables that serve as a trigger to signal whether or not a specific distortion is active, and (iii) the integer variables that represent distorted pixel coordinates. For example, to encode rotation we first conserve the activation binary switch, followed by the coordinates of the central pixel, and finally the rotation angle. Therefore, the search space of data transformations is a multi-dimensional

vector space, $T \in \mathbb{R}^k$, where each component/axis represents either a parameter value, an activation value, or a coordinate value. (see Figure 5.2).

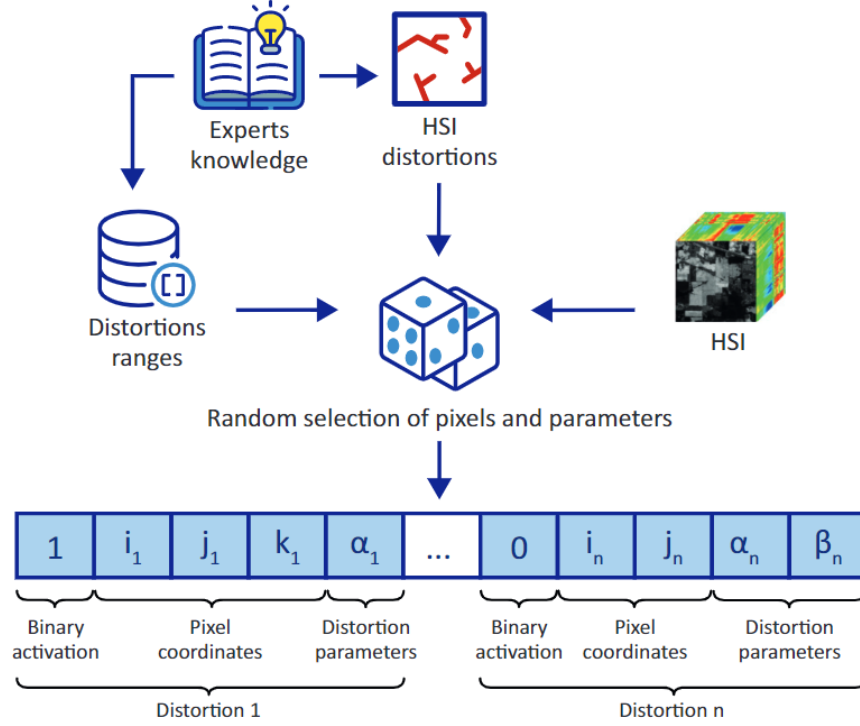


Figure 5.2 Formatting all distortions into a transformation vector

Design of the Fitness Functions

As our main objective is to expose high density and diversity of difference-inducing test inputs, we should design a fitness function, f_X that measures how much the synthetic input $\hat{x}$ induces behavioral divergences between the original DNN and its quantized version. Nevertheless, we define the data transformation space T to be our search space. The optimizer would use a fitness function, f_T , to compare and contrast the fault-revealing ability of data transformations, $t \in T$, and the connection between both fitness functions, f_X and f_T , would simply be $f_X(\hat{x}) = f_T(t)$, where $\hat{x} = t(x)$.

In the following, we discuss two specialized fitness functions that have been designed to promote the fault-detection capability and the diversity of the generated test inputs.

- (i) Divergence-based Fitness Function:

Given two DNNs, the original model, m_o , and its quantized version, m_q , we denote $\mathbf{s}_o$ and $\mathbf{s}_q$ their respective softmax activation functions. Softmax is the last layer (output) activation that transforms the logit scores into probability distribution, where its component is the probability of a class label membership. The predicted label for the input is the one with the highest probability. Given a c -classification problem and a vector of logits, $\mathbf{l} = (l_1, \dots, l_c) \in \mathbb{R}^c$, the softmax, $\mathbf{s} = (s_1, \dots, s_c) \in \mathbb{R}^c$ is computed as follows:

$$s_i = \sigma(l_i) = \frac{e^{l_i}}{\sum_{j=1}^c e^{l_j}} \quad \text{for } i = 1, \dots, c \quad (5.2)$$

We aim to design a fitness function that estimates the divergence between the two probability of class membership distributions obtained by the original and quantized DNNs. Indeed, maximization of this divergence measure would increase the chances of disagreement between the two models caused by a mismatch of their two most-probable labels for the same input. There are two common divergence measures to compare different probability distributions, Kullback–Leibler divergence (KLD) [172] and Jensen–Shannon divergence (JSD) [173]. In our divergence-based fitness function, we opt for JSD, denoted J . Considering two probability distributions, Q and R , defined on the same probability space χ ($|\chi| = c$), J can be formulated as follows:

$$J(Q||R) = \frac{1}{2}(D(Q||M) + D(R||M)) \quad (5.3)$$

where $D(Q||M) = \sum_{i=1}^c Q(i) \ln(\frac{Q(i)}{M(i)})$ and $M = \frac{1}{2}(Q + R)$.

The main reason we chose JSD over KLD is that the former is symmetric, $J(Q||R) = J(R||Q)$, and bounded, $0 \leq J(Q||R) \leq 1$. Therefore, we define, below, our divergence-based fitness function of the synthetically-produced inputs.

$$f_X^{div}(\hat{x}) = J(\mathbf{s}_o(\hat{x}), \mathbf{s}_q(\hat{x})) \quad (5.4)$$

This divergence-based fitness function can be considered as grey-box testing criterion because it has access only to the last layer softmax activations. Despite it targets directly the enhancement of the distance between the two models' predictions, it risks the phenomenon of mode collapse, where the optimizer falls on a local maximum (i.e., the transformed inputs trigger high or even the maximum divergence). Hence, all the generated inputs would be very similar or even identical. To mitigate this probable issue, we propose the following alternative fitness function.

(ii) Coverage-based Fitness Function:

We aim to design a white-box testing criterion that estimates differences in the signature of neurons activations obtained from both tested versions of the DNN. This helps compute at fine-grained level the behavioral divergences caused by the test input and ensures that the optimizer would continuously search for the synthetic inputs, inducing higher differences at all the layers' activations.

To do that, we rely on K-multisection Neuron Coverage (KMNC) that is inspired by the Neuron Coverage. The adoption of KMNC was motivated by the diversity enhancement in test input generation achieved by neuron coverage. In fact, diversifying inputs helps prevent mode collapse. Pei et al. [88] and Tian et al. [94] have demonstrated that increasing the neuron coverage is highly related to enhancing the diversity of the generated test cases. In the same vein, Yu et al. [174] have shown that coverage-guided test generation produces inputs with increasing L1 distances from their original seeds. Indeed, KMNC does not consider each neuron as active (1) or inactive (0) depending on a predefined threshold like NC, but instead, it refines this over-approximate binarization of the continuous neuron output into a multi-section discretization.

Let $N = \{n_1, n_2, \dots, n_M\}$ be the set of neurons of a given DNN and $I = \{i_1, i_2, \dots, i_M\}$, $i_m = [low_m, high_m] \ \forall m \in \{1 \dots M\}$, be the set of neurons activations' intervals observed during the training, KMNC divides all the intervals into k equal sections that should be covered by the test data, as an indication that the major behaviors of the trained DNN are triggered at least once during the testing.

As we are interested in defining a signature of neurons activations triggered by an input x , we exploit KMNC as detailed below.

Let $S_i^{n_m}$ be the set of activations from the neuron n_m in the i -th section for $1 \leq i \leq k$. Therefore, $\phi(\mathbf{x}, n_m) \in S_i^{n_m}$ indicates if the i -th section of the neuron n is covered by the input $\mathbf{x}$. Then, we define the signature S^x as the subset of all the neurons' sections covered by x , as follows:

$$S^x = \{S_i^{n_m} | \phi(\mathbf{x}, n) \in S_i^n\}, \quad \forall m \in \{1 \dots M\} \quad (5.5)$$

Given an original and quantized DNN models, we can infer their corresponding signatures of neurons' activations for the same input x , respectively, denoted as S_o^x and S_q^x . As each of them represents a set of covered sections, we can use the Jaccard similarity coefficient [175], J_{sc} , which is a well-known similarity metric for comparison between

two sets.

$$J_{sc}(S_o^x, S_q^x) = \frac{|S_o^x \cap S_q^x|}{|S_o^x| + |S_q^x| + |S_o^x \cap S_q^x|} \quad (5.6)$$

Thereby, we design the coverage-based fitness function, f_X^{cov} , that assesses the dissimilarities in the neurons activations state obtained from the two DNN versions in order to encourage the generated test inputs intensifying these dissimilarities, and hence, expose hidden disagreements between their final predictions.

$$f_X^{cov}(\hat{x}) = -J_{sc}(S_o^{\hat{x}}, S_q^{\hat{x}}) \quad (5.7)$$

The design of f^{cov} is inspired by the feature matching [176] concept from Generative Adversarial Networks (GANs). GAN suffers from mode collapse when the generator over-exploits the mistakes previously made by the discriminator and fails to explore beyond them. To stabilize GAN training and decrease mode collapse, a new objective function design was proposed to steer the generator towards matching the expected values of the discriminator's hidden layers, rather than matching only the discriminator's output.

Although the second fitness function takes into account the internal neuron activities and estimates the internal behavioral divergences at fine-grained level, there is no guarantee that the generated test inputs, emphasizing these neuron-level differences between two under test models, would provoke mismatches of their predicted labels, as evidenced by former coverage-guided approaches [92, 93]. From this perspective, we assume that the first fitness function based on the divergence of softmax outputs can be more effective in driving the test input generation towards enlarging the gaps between the two probability of class membership distributions obtained by the original and quantized DNNs, until they predict two unlike most probable labels.

Moving on to the fitness evaluation of the generated data transformation, in our approach, we support two modes. The first is the single instance mode where only one original input x^i is fixed in the loop, so that all the generated transformations $t^j \in T$ would be evaluated based on the fitness $f_T(t^j) = f_X(\hat{x}^{i,j})$, where $\hat{x}^{i,j} = t^j(x^i)$. The second is the batch mode that fixes a subset of B original inputs, X_b , at once, so that any data transformation $t^j \in T$ would be evaluated based on the average fitness $\bar{f}_T(t^j) = \frac{\sum_{i=1}^B f_X(\hat{x}^{i,j})}{B}$, where $\hat{x}^{i,j} = t^j(x^i)$, $\forall i \in$

$\{1 \dots B\}$. For the sake of simplicity, we present the formulation of our above-mentioned fitness functions on the single instance mode.

It is worth highlighting that, in the scope of this work, we define two fitness functions. Nevertheless, DiverGet provides a plug-and-play design in which users may specify their preferred fitness functions. In reality, a variety of alternative criteria might be used to determine divergence. The work of [177], in which authors attempt to evaluate a model’s robustness through adversarial attacks, might serve as an inspiration for the design of a new fitness function. The authors designed a straightforward function that assesses the model’s output score given to the target class label. In the same spirit, and with minor tweaks to binarize this fitness function, one may compute the difference between the log of the highest class probability predicted by the original model and the log of the one predicted by its quantized counterpart. Another fitness function design could be inspired by the prediction uncertainty introduced in DiffChaser’s paper [92]. The prediction uncertainty indicates that the DNN is uncertain whether to classify the input x as y or y' because their predictive probabilities are very close. Following this idea, the new fitness function could compute the distance between the two highest class probabilities that are predicted the original model when having x as input.

Implementation of Metaheuristic-based Optimizers

As introduced in Section 2.3.3, we apply SBST, using population-based metaheuristics, to drive optimally the transformation generation towards diverse prominent regions in the search space. Algorithm 2 explains the generic structure and steps of our approach to generate data transformations. Additionally, DiverGet supports divergence-based and coverage-based fitness functions as criteria for test adequacy, and it is also compatible with any population-based metaheuristic algorithm. The proposed algorithm starts with an initial random population of p transformation vectors (Line 2). The synthetic test inputs would be the result of the application of these data transformations (transformation vectors) and our original input data (Line 5). Then, the algorithm computes the fitness values for all the population individuals, depending on the selected fitness function and the evaluation mode (Line 6). Based on the obtained fitness values, the chosen metaheuristic algorithm applies its update routines on the population to derive the next generation of new candidates that are strong and likely better than their predecessors in terms of fitness (Line 13). The enhancement of the fitness values would lead to synthetic inputs that enlarge further the behavioral deviations between the two tested DNNs either at the last layer’s level or hidden layers’ level, depending on the choice of the fitness function. Hence, we have high chances to find DIIs

among the produced generations over the iterations. However, we pass validity test on the transformed inputs based on their resulting PSNR measures and only the valid ones (their PSNR values are more than 20 dB) would be considered for the difference-inducing selection criterion (Lines 7 – 12) (see Section 5.1.3 (iii)). Indeed, those lines are the major change with regards to the standard steps of population-based metaheuristics. This change alters the objective of the optimizer from searching for an optimal solution (i.e., candidate having the highest fitness value) to tracking all the revealed DIIs among the evolving valid candidates throughout the generations. Last, the generation loop for transformations on one or multiple inputs, depending on the activated fitness evaluation mode, is repeated until the maximum number of generations is reached or when the early stop condition (i.e., no changes on either fitness values or number of DIIs) is encountered (Line 14).

Algorithm 1: Data Transformation Generation

Input: x : input/ batch of input, m_o : original DNN, m_q : quantized DNN, p : population size, $maxiter$: max iterations

Output: D : Set of difference-inducing transformation vectors

```

1:  $D := \emptyset$ ;
2: initial random population  $T$ ;
   /* population of  $p$  data transformation vectors */
3: iteration := 0;
4: while iteration < maxiter do
5:    $\hat{X} := \text{GenerateHSIs}(T, x)$ ;
   /* by applying every  $t \in T$  on  $x$  */
6:    $F := \text{ComputeFitness}(\hat{X})$ ;
   /* fitness function for each  $\hat{x} \in \hat{X}$  */
7:    $\hat{X}_{valid} := \text{CheckValidationConstraint}(\hat{X}, x)$ ;
   /*  $\hat{X}_{valid} \subset \hat{X}$  */
8:   for  $\hat{x}_d \in \hat{X}_{valid}$  do
9:     if isDII( $\hat{x}_d, m_o, m_q$ ) then
10:        $D := D \cup \{t_d\}$ ;
11:     end if
12:   end for
13:    $T := \text{UpdatePopulation}(T, F)$ ;
14:   if isEarlyStopping( $D, F$ ) then
15:     break;
16:   end if
17:   iteration := iteration + 1;
18: end while

```

In line with the No Free Lunch Theorem (NFL) [178], we implement two concurrent nature-inspired population-based metaheuristics, PSO and GA, that have shown their effectiveness in high-dimensional search problem such crafting black-box adversarial attacks [177, 179]. For

each implemented metaheuristic, we tune its hyperparameters to appropriately set up its level of non-determinism and of balance between the intensification (i.e., exploitation of the best candidates found to concentrate the search on the prominent regions) and diversification (i.e., the exploration of non-visited regions to avoid missing potential interesting solutions) [180].

5.2 Experiments

We evaluate the effectiveness of DiverGet through the following research questions (RQ):

- **RQ1:** How effective is DiverGet’s main feature (i.e., the domain-specific metamorphic relations and the search-based data transformation) at finding difference-inducing inputs?
- **RQ2:** How would DiverGet perform on numerous models, datasets, quantization approaches, fitness functions, and metaheuristic algorithms?
- **RQ3:** How does DiverGet compare to state-of-the-art DiffChaser?

5.2.1 Experimental Setup

In this subsection, we detail the different elements of our experimental setup.

- **Datasets:** We consider two popular and publicly-available² hyperspectral datasets. The first is Pavia University (PU), a HSI made of 610×340 pixels with 103 spectral bands in the wavelength range of 430 to 860 nm. Its ground truth shows 9 urban land-cover classes. The second is Salinas (SA), a HSI made of 512×217 pixels and 204 spectral bands in the wavelength range of 360 to 2500 nm. Its ground truth contains 16 vegetation classes.
- **DNNs:** To perform a classification of land covers using a CNN, the HSI is cropped into multiple 3D patches, with a prefixed size, that would be used as model inputs. For a better diversity of the evaluated subjects, we studied state-of-the-art CNNs, namely SSRN [2] and HybridSN [1], that are quite popular, open-source, and previously tested on the PU and SA datasets.

SSRN is an end-to-end spectral-spatial residual CNN. It relies on 3D convolutional layers as a basic element of its architecture. It consists of two spectral and two spatial

²http://www.ehu.eus/ccwintco/index.php/Hyperspectral_Remote_Sensing_Scenes

residual blocks that learn discriminative features from spectral signatures and spatial contexts in the HSI.

HybridSN is a spectral-spatial CNN that combines both 3D and 2D convolutional layers. It uses 3D convolutional layers to learn the joint spatial-spectral feature representation on latent spatial features that are extracted by upper-level 2D convolutional layers.

Table 5.2 gives an overview of these CNNs.

Table 5.2 Details of DNNs and datasets used to evaluate DiverGet. The Overall Accuracies (OAs) are taken from [1, 2]

Dataset	DNN Model	# 3D patches	OA (%)
PU	SSRN	Train, Val: 4281; Test: 34214	99.79
	HybridSN	Train: 12832; Test: 29944	99.98
SA	SSRN	Train, Val: 5418; Test: 43293	99.98
	HybridSN	Train: 16238; Test: 37891	100

- Quantization methods: For more credibility and reproducibility, we chose to use real-world quantization methods, implemented by state-of-the-practice DL tools. More precisely, we opted for Tensorflow Lite³, Google’s open-source DL library for on-device deployment. For each original DNN, i.e., implemented in 32-bit floating precision, we performed 2 types of Tensorflow Lite’s provided quantization techniques, namely the 8-bit post-training quantization and the 8-bit quantization-aware training.

- Evaluation metrics:

- Metrics of models divergence: We use three metrics to evaluate the divergence between the model versions: (i) the number of difference-inducing inputs (#DII), (ii) the divergence rate (DiR), i.e., the percentage of difference-inducing inputs discovered over all the generated inputs for each seed, and (iii) the success rate (SR), i.e., the percentage of original 3D patches where at least one disagreement is successfully revealed among their descendant synthetic inputs.

These metrics try to assess the divergence from different angles. In fact, DiR (and consequently #DII) tries to quantify the degree of divergence between the two model versions by reporting all possible detected DIIs. Whereas, SR tries to detect the existence of divergence between the two models.

³<https://www.tensorflow.org/lite>

- Metric of input validation: As introduced in Section 5.1.3, the validity of synthetic HSI is defined by their PSNR value compared to the original HSI, to verify the image quality and ensure that semantics are preserved. If the PSNR of distorted patches is less than 20dB (experts’ recommendation based on previous work), they are discarded as invalid inputs. Therefore to inform about the quality of generated HSIs (transformations that lead to HSIs), we define the Validation Rate (VR) metric as the percentage of valid-generated transformations/HSIs over all generated transformations/HSIs, for each seed.
- Metric of execution time: To evaluate the execution time of their framework, the previous work [91] has focused on quantifying the time required to detect the first disagreement. We followed the same approach and computed for each original HSI, the time required to identify the first disagreement per patch (FDI). In addition, as indicated in the work of [91], we measured the time required to identify the first disagreement per patch only for successful cases (i.e., original HSIs that resulted in DIIs) (FDI*). This measure is applied per patch only for successful cases (i.e., HSIs that lead to DIIs) (FDI*). The FDI* measure allows reporting the time without penalization to the framework in the absence of disagreement on a particular patch.
- Metric of statistical significance and effect size: We use statistical hypothesis testing and effect size measurements to assess the statistical significance of our results. As our data are unlikely to be normally distributed, we utilized non-parametric hypothesis testing. Additionally, a paired test was chosen because of our belief in the interdependence of the two distributions. Specifically, we performed the Wilcoxon Signed Rank test [181] and Vargha-Delaney 12 [182] effect size test to determine how much two groups differ from one another.

If two groups are statistically indistinguishable, $\hat{A}_{12} = 0.5$. $\hat{A}_{12} > 0.5$ indicates that, on average, the first group outperforms the second, $\hat{A}_{12} < 0.5$ signifies that the second group outperforms the first. The magnitude of the difference between the groups can be classified into four categories based on the scaled $\hat{A}_{12}$ [183]: “negligible” ($|\hat{A}_{12}^{scaled}| < 0.147$), “small” ($0.147 \leq |\hat{A}_{12}^{scaled}| < 0.33$), “medium” ($0.33 \leq |\hat{A}_{12}^{scaled}| < 0.474$), and “large” ($|\hat{A}_{12}^{scaled}| \geq 0.474$).

- Software: We developed DiverGet in Python. It supports Tensorflow (version 2.4.1) [184] and Tensorflow Lite models. For the metaheuristics implementation, we adapted the

open-source python libraries, pyswarm⁴ and geneticalgorithm⁵, to meet our design of population-based searching algorithm specifications.

- **Hardware:** We run all experiences on B2S instances of Microsoft Azure Virtual Machines. Each of these has a 2-core 2.4 GHz Intel Xeon CPU and 4 GB of RAM, and runs on ubuntu 18.04.

The next sections describe the experiments that were carried out to answer our research questions (please consult Figure 5.3 for an overview of the experiment design).

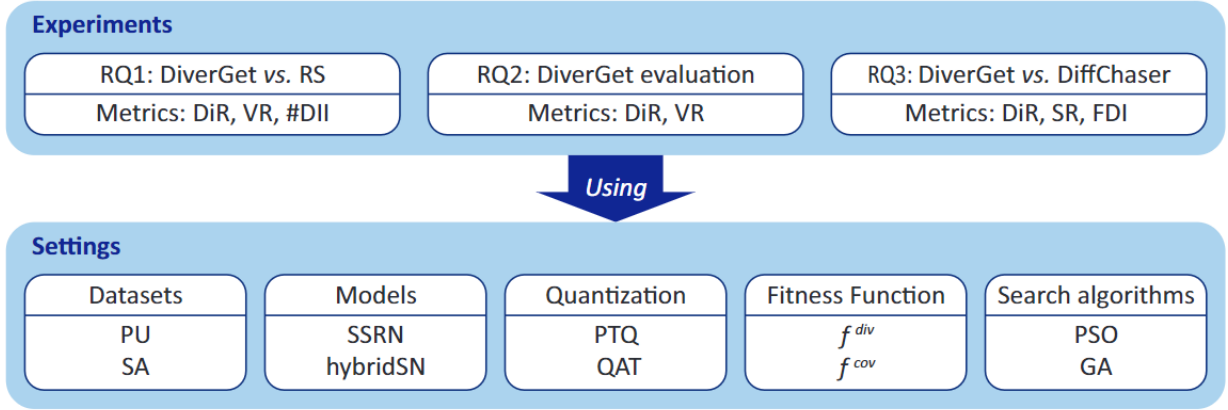


Figure 5.3 Overview of the experiment design and evaluation metrics.

5.2.2 RQ1: The Effectiveness of DiverGet

Motivation: Throughout its process, DiverGet (i) relies on domain-specific metamorphic relations to provide semantically preserving data and (ii) leverages various population-based metaheuristic algorithms to guide the search process and generate quality test data.

Method: For each dataset, we randomly select 10 seeds from a pool of 800 original inputs. All the original test inputs are sampled from the subset of test datasets, for which the original model correctly predicts their corresponding labels. For each model, we compare the original version to a quantized version using either PTQ or QAT. For the first experiment (i.e., evaluating the effectiveness of the proposed metamorphic distortions), we set a random sampler (RS) to generate synthetic data using our defined domain-specific metamorphic relations. We then compared the number of difference-inducing inputs (DII) found in the original test data and those generated by RS.

⁴<https://pythonhosted.org/pyswarm/>

⁵<https://github.com/rmsolgi/geneticalgorithm>

For the second experiment (i.e., evaluating the effectiveness of our searching strategy), we compared for each seed the RS equipped with our metamorphic distortions to DiverGet using various configurations. We compare the results of the second experiment using two metrics: i) the divergence rate (DiR), i.e., the percentage of difference-inducing inputs discovered over all the generated inputs for each seed, ii) the Validation Rate (VR), i.e., the percentage of valid-generated distortions ($\text{PSNR} > 20\text{dB}$) over all generated distortions, for each seed.

Results: Table 5.3 reports the number of DIIs revealed by the original test data and the synthetic inputs randomly sampled from the specified metamorphic relations. Tables 5.4 reports the median of the selected measurements (i.e., DiR, VR) of all used seeds when comparing DiverGet to RS. The values obtained by PSO and GA are averaged to represent the column ‘DiverGet’ in Table 5.4.

Table 5.3 Comparison of the number of DII found in the original test data and by RS using all seeds

Model	Dataset	Quantization	# DII - Original Test Data	# DII - Random sampling (RS)
SSRN	PU	PTQ	136	1609
		QAT	1	763
	SA	PTQ	0	132
		QAT	40	498
hybridSN	PU	PTQ	0	133
		QAT	1	522
	SA	PTQ	0	110
		QAT	10	506

Naturally-occurring synthetic inputs vs original test inputs: Table 5.3 shows that random samples of synthetic inputs expose more difference-inducing test cases than those of original inputs. Thus, the designed input distortions validate application-specific robustness requirements using corner-case test scenarios that cannot be assured directly from original test cases.

Finding 1: The designed domain-specific metamorphic relations expose uncovered behavioral divergences resulting from the quantization that original test data failed to shed light on.

Table 5.4 Comparison of the median values of DiR and VR between RS and DiverGet

Model	Dataset	Quantization	RS		DiverGet	
			DiR	VR	DiR	VR
SSRN	PU	PTQ	1.07	3.75	24.05	75.82
		QAT	0.48	3.89	15.68	70.28
	SA	PTQ	0.03	3.38	5.05	70.43
		QAT	0.33	3.45	18.27	70.72
hybridSN	PU	PTQ	0.08	3.66	8.43	67.77
		QAT	0.43	3.84	10.92	67.52
	SA	PTQ	0.02	2.9	3.07	67.56
		QAT	0.25	2.85	9.96	68.18

All results have a p-value $< 3.3 * 10^{-4}$ and an effect size > 0.98 (large)

Population-based metaheuristic algorithms vs Random Sampling: Table 5.4 demonstrates a comparison of the median of DiR and VR of all seeds between RS and DiverGet. Although random sampling from our synthetic inputs enables the detection of behavioral deviations, the revealed occurrences remain very low compared to search-based approach (using GA or PSO) due to the high-dimensional search space and the lack of optimization over generations.

Finding 2: DiverGet’s searching strategy using population-based metaheuristic succeed in outperforming the Random Sampling strategy into steering the generation into prominent regions.

5.2.3 RQ2: DiverGet Facing Multiple Settings

Motivation: Throughout its quantization assessment steps, DiverGet supports the instantiation with multiple settings that influence its effectiveness regarding different quality aspects of the generation process of difference-inducing data transformations.

Method: We experiment DiverGet on (i) original test samples from both HSIs, (ii) state-of-the-art DNNs, SSRN and HybridSN, and (iii) conventional 8-bit quantization methods, PTQ and QAT. We worked with 50 random seeds from a pool of 8000 original inputs. All the original test inputs are sampled from the subset of test datasets, for which the original model correctly predicts their corresponding labels. Regarding the settings of DiverGet, we

try different options for each experiment. We vary the selected metaheuristic algorithms, either PSO or GA, as well as the targeted fitness functions, f^{div} and f^{cov} . Throughout the trials, we keep track of (i) the divergence rate (DiR) and (ii) the Validation Rate (VR).

Results: Tables 5.5, and 5.7, along figures 5.5, and 5.6 report the average of the selected measurements (i.e., DiR and VR) with respect to one or multiple controlled variables of our experiments (i.e., models, datasets, quantization techniques, fitness functions, and metaheuristics).(more detailed results can be found in the supplementary materials)

Table 5.5 Comparison of the median values of DiR and VR when using DiverGet with f^{cov} and f^{div}

Data	Quantization	f^{div}		f^{cov}	
		DiR	VR	DiR	VR
PU	PTQ	22.47(*)	72.36(**)	9.57(*)	70.74(**)
	QAT	21.15(*)	71.57(*)	7.37(*)	68.02(*)
SA	PTQ	8.18(*)	70.54(**)	0.99(*)	69.29(**)
	QAT	18.17(*)	71.02(*)	9.80(*)	68.03(*)

All results have a p-value $< 7.7 * 10^{-4}$.

Effect sizes indicated as: * > 0.918 (large), ** $0.69 - 0.71$ (medium)

Divergence-based fitness vs coverage-based fitness: Table 5.5 presents the comparison of DIIs and VRs when we use metaheuristic searching with either coverage-based fitness, f^{cov} , or divergence-based fitness, f^{div} . Results show that f^{div} outperforms f^{cov} in guiding the generation of data transformation towards prominent regions. This is aligned with our expectations given that the sharp guidance of f^{div} directly targets the enlargement of the deviation between the two versions of the DNN. However, we discussed the mode collapse risk when the search process was primarily rewarded for the detection of deviations without any implicit or explicit incentives on their diversification. In the following, we aim to get more insights on the distribution of the transformation vectors that have been discovered by both search strategies. We project these vectors into a two-dimensional sub-space using t-Distributed Stochastic Neighbor Embedding (t-SNE) [185] to visualize the generated transformation distribution. t-SNE, a nonlinear dimensionality reduction method, embeds the transformation vectors into a lower-dimensional space by generating probability distributions that preserve the similarity of high dimensional data. Figure 5.4 presents the outcome of this projection. It shows that f^{cov} tends to discover more separate regions, to which difference-inducing inputs belong. In contrast, f^{div} focuses on a particular central region with a high density of

difference-inducing inputs, but it has not prevented the metaheuristic search from arbitrarily trying more distant transformations with fewer similarities that are quite dispersed in the search space.

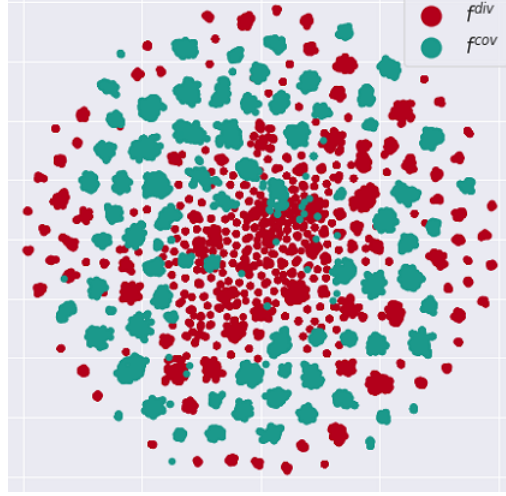


Figure 5.4 t-SNE visualization of transformation vectors induced by f^{div} and f^{cov} .

To support our statement, we performed a statistical analysis on these randomly sampled transformation vectors. More specifically, we measured the pairwise Euclidean distances in each group (i.e, the group generated using f^{cov} and f^{div}) of more than 10000 transformation vectors (the experience was repeated *3times*). We then reported the average minimum, maximum, mean, variance, and first and third quartile of the two group distances. Table 5.6 presents the obtained results. All the statistical indicators (except the maximum) demonstrate that transformation vectors generated using f^{cov} are more diverse and more spread in the search space than the ones generated using f^{div} . These results reinforce the statement that f^{cov} could be an alternative to prevent the mode collapse, since it can be used to enhance the diversity of generated transformation vectors (and ultimately DIIs).

Table 5.6 Statistics over the pairwise distances of transformation vectors generated using f^{div} and f^{cov}

Fitness Function	Min	Max	Mean	Var	Q1	Q3
f^{div}	0	1268.8	572.2	25770.21	461.72	671.1
f^{cov}	0	1268.73	602.37	29791.4	486.83	712.42

Finding 3: Divergence-based fitness function succeeds in steering the input generation towards regions of higher density of disagreement-revealing ability than its coverage-based counterpart, while preserving the test data diversification.

GA vs PSO as metaheuristic searching algorithm: Table 5.7 presents the comparison between PSO and GA as a metaheuristic for DiverGet using the same aforementioned measures. Results shows that PSO and GA succeed in improving the disagreement-revealing ability over the generations. Indeed, our in-depth investigation on each metaheuristic's optimization routines and their exposed transformation vectors, leads us to believe that issues in relation with exploitation/exploration are the main reason behind this performance gap. On the first hand, PSO tends to explore more regions since all particles have to move, even slightly, in each generation. On the other hand, GA tends to highly exploit the most-fitted candidates via breeding between them, to converge quickly towards the best-fitted solution. Hence, GA fails to explore farther candidate transformations in other regions. Besides, GA applies random mutations on the new generations of candidates aiming at reaching distant regions, but the non-determinism of this arbitrary mutation increases the occurrences of out of the space candidates, as evidenced by the obtained low valid rates (VR). For such search-based approach where the exploration has higher priority than smooth and rapid convergence, PSO might be more appropriate metaheuristic searching algorithm.

Table 5.7 Comparison of the median values of DiR and VR between PSO and GA as metaheuristics for DiverGet

Data	Quantization	DiverGet			
		PSO		GA	
		DiR	VR	DiR	VR
PU	PTQ	16.28(**)	83.04(*)	15.76(**)	60.05(*)
	QAT	12.99(*)	80.42(*)	15.53(*)	59.18(*)
SA	PTQ	3.78(*)	81.04(*)	5.40(*)	58.79(*)
	QAT	11.32(*)	79.98(*)	16.64(*)	59.07(*)

* indicates a p-value $< 8.1 \times 10^{-13}$ and an effect size > 0.92 (large)

** indicates a p-value = 0.133 and an effect size = 0.59 (small)

Finding 4: PSO outperforms GA, as a more adequate metaheuristic-based search algorithm in terms of the size/ratio of its uncovered difference-inducing inputs.

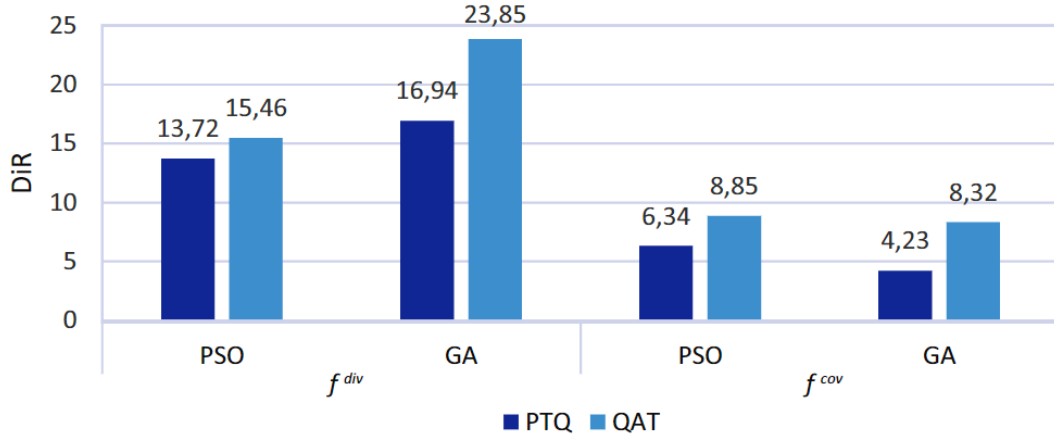


Figure 5.5 Comparison of the median values of DiR between DiverGet implementations using QAT-based and PTQ-based quantization models. All results have a p-value $< 5.8 \times 10^{-9}$ and an effect size > 0.84 (large).

PTQ vs QAT as robust quantization technique: Figure 5.5 reports the rate of generated difference-inducing inputs when assessing both of QAT-based and PTQ-based quantized models built on the subjects CNN models and HSI datasets. Results show that QAT-based spawn higher rate of behavioral divergences than its PTQ-based counterpart. This is not aligned with the test accuracy evaluations because the QAT is supposed to refine the rounding of floating-point parameters with more precise ranges estimated on the training data. This suggests that the quantization methods that rely strongly on the original data distribution at different levels, including reduction of precision and evaluation, will often be subject to widely divergent behaviors when tested on shifty data inputs that are closer to real contexts and quite dissimilar with the major situations.

Finding 5: Quantization-aware training can be less robust than post-training quantization, against naturally-occurring data distortions that can accentuate the gap between synthetic and training data distributions.

PU vs SA as complex HSI classification problem: Figure 5.6 reports the rate of generated difference-inducing inputs when assessing quantized models that solve two different HSI classification problems. Results highlight that disagreements are more likely to occur using PU than SA. This is mainly due to the nature of each scene. In fact, SA was captured over a rural location comprising large homogeneous fields, which resulted in large and rather uniformly-shaped classes with homogeneous neighborhoods. As a consequence, DNNs can

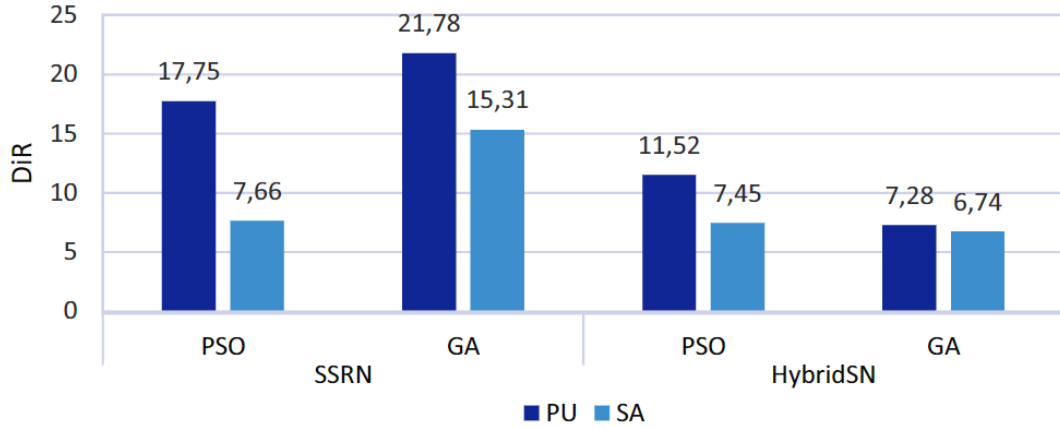


Figure 5.6 Comparison of the median values of DiR between DiverGet implementations using different datasets. All results have a p-value $< 1.5 * 10^{-11}$ and an effect size > 0.89 (large).

easily classify any 3D patch from SA, even when distorted, which makes generating difference-inducing inputs more challenging. However, PU was captured over an urban site where different land covers are represented by small and close regions that are scattered all over the image. As a result, and given PU's heterogeneous neighborhoods, distorting a 3D patch can harshly affect the spatial information, mislead the classifier, and easily generate more difference-inducing inputs. Last, in line with [91], larger models like SSRN have higher chances to generate more disagreements than hybridSN (see Figure 5.6). Indeed, SSRN is a residual network composed of more layers than hybridSN.

Finding 6: The complexity of the HSI classification problem, especially in terms of homogeneity of the pixel patch, can make the systematic assessment via naturally-occurring data distortions, more challenging to pass for quantized CNNs.

5.2.4 RQ3: DiverGet vs DiffChaser

Motivation: The goal is to compare DiverGet's performance in discovering difference-inducing inputs in comparison with DiffChaser, the state-of-the-art framework for detecting disagreement among DNNs.

Method: We slightly modified DiffChaser to add support for HSI. We used DiffChaser with its basic fitness function since it is (i) untargeted and (ii) results in more DIIs than the other DiffChaser k-uncertainty fitnesses. For a fair comparison, we chose the f^{div} as our fitness function since it exploits the last layer of the two DNNs to produce divergence which is quite

similar to DiffChaser’s basic fitness function design. We experimented on 50 random seeds from a pool of 8000 3D patches from the two test data of SA and PU. We also used the two quantization methods PTQ and QAT with the two HSI models SSRN and HybridSN. To provide the most accurate comparison, we run both frameworks with the same population size, 10, and with the same number of iterations, 25, that would yield in both evaluation the total of $10 \times 25 \times 8000$ synthetic samples. As measurements, we use the two metrics, divergences rate (DiR) and the success rate (SR), i.e., the percentage of original 3D patches where at least one disagreement is successfully revealed among their descendant synthetic inputs.

In our quantization assessment strategy, DiR estimates the expectation of quantized model divergence rates against natural perturbations by analogy with estimation of the expected model’s misclassification. This complements other metrics, such as success rate and time to find first disagreements (used in DiffChaser), that demonstrate the unavoidable existence of divergence between two DNN versions with different arithmetic-precision. To assess the effectiveness of different quantized models, we compare the likelihood that they will fail when faced with naturally-occurring distorted inputs.

To have a better understanding, we summarize the difference between the two frameworks in Table 5.8.

Table 5.8 DiverGet vs DiffChaser

Level	DiverGet		DiffChaser
Distortion design	Naturally-occurring perturbation		White noise injection
Constraints	PSNR		L_∞ norm
Search space	Transformation vector space		Input space
Search objective	Behavioral	Divergence Maximization	Targeted/Untargeted attack
Goal	DIIs generation: Exposing every possible divergence for realistic quantization assessment against corner case scenarios.		First disagreement attack: if disagreement is found DiffChaser stops or re-starts from random state.

Results: Table 5.9 summarizes the overall results. It shows that DiverGet configured with PSO successfully generates disagreements for 40.98% of 3D patches on average, with an average DiR of 14.59%. DiverGet can still generate disagreements when applied with GA

for an average of 27.27% of 3D patches with a DiR of 20.40%. For DiffChaser, the success rate is 11.25% on average, which is lower than DiverGet with both metaheuristics options. Moreover, the gap between the two frameworks is also remarkable in regards to the rates of revealed divergences. In fact, DiffChaser generates an average DiR of 2.78% which is quite low compared to DiverGet.

Table 5.9 Comparison of the median values of DiR and SR between DiffChaser, PSO-based DiverGet and GA-based DiverGet

Framework	Model	PU				SA				Average	
		PTQ		QAT		PTQ		QAT		DiR	SR
		DiR	SR	DiR	SR	DiR	SR	DiR	SR		
DiffChaser	SSRN	16.66	49.38	0.31	10.63	0.35	9.69	3.68	16.58	2.78	11.25
	Hybrid-SN	0.001	0.31	0.002	0.31	0.001	0.63	1.22	2.50		
DiverGet (PSO)	SSRN	24.96	71.25	16.42	61.25	3.60	43.75	13.92	63.44	14.56	40.98
	Hybrid-SN	16.97	24.38	20.08 (**)	28.75	9.35	14.06	11.42	20.94 (*)		
DiverGet (GA)	SSRN	35.90	58.75	28.86	37.50	14.47	20.63	31.93	43.75	20.40	27.27
	Hybrid-SN	12.06	13.75	19.22 (**)	20.94	5.32	5.63	15.46	17.16 (*)		

* indicates an effect size = 0.73 (medium)

** indicates a p-value = 0.305 and an effect size = 0.55 (negligible)

All the remaining results have a p-value $< 4.6 * 10^{-5}$ and an effect size > 0.82 (large).

This gap may be explained by the following differences: *(i) Search space encoding*: we define the transformation vector space as our search space instead of the input space. This reduces significantly the search space, enables the inclusion of diverse distortion type, and allows a better convergence of the optimizer. HSI classification problem sheds the light on the effect of our variable change' trick in reducing the high-dimensionality of HSI into a single vector of transformation metadata; *(ii) Metamorphic transformations*: DiverGet uses several radio-metric and geometric transformations to enrich the transformation space whereas DiffChaser simply applies white noise injection. This offers a great advantage to our approach and allows our search process to explore prominent regions and enhances the chances to expose hidden divergences; *(iii) First-occurrence disclosure objective*: DiffChaser was designed first to focus entirely on exploring the input space with the objective of discovering the first disagreement. This might also explain this performance disparity compared to DiverGet that

seeks to produce the most difference-inducing inputs from each 3D patch.

The time required to detect the first disagreement is a primary metric used to evaluate DiffChaser. In this experiment, we are aiming to evaluate DiverGet using this metric and compare it to DiffChaser. To that end, we used for this experiment a subset of 3 seeds (from the 50, previously mentioned, seeds) on which we run DiffChaser and DiverGet. We slightly modified DiverGet to support the first attack concept introduced in DiffChaser and computed for each original HSI the time required to identify the first disagreement per patch (FDI). In addition, as indicated in DiffChaser’s publication, we measured the time required to identify the first disagreement per patch only for successful cases (i.e., original HSIs that resulted in DIIs) (FDI*). The intuition behind this measure is that it will report the time without penalizing the framework if it does not identify a disagreement for a particular patch. The outcomes of this experiment are shown in Table 5.10. The experiment demonstrates that the results are strongly dependent on the model under test. When using SSRN, DiffChaser outperforms DiverGet configured with both metaheuristics. GA-based DiverGet generates the first DII in roughly 66 seconds on average, and 39 seconds when only successful cases are considered (FDI*). DiverGet with PSO configuration performed better. It takes 36 seconds on average to generate the first DII (26 seconds when only successful cases are considered). DiffChaser, on the other hand, is more efficient than DiverGet configured with both metaheuristics in detecting the first disagreement. In fact, given all seed inputs, it takes around 7 seconds to produce the first disagreement and 2 seconds to generate the first disagreement for successful cases only. When hybridSN is used as a model under test, the situation is mirrored. With an average FDI of 14 seconds (and an average FDI* of 5 seconds), DiffChaser becomes the slower framework. DiverGet, based on GA, performs better, with an average FDI of 11 seconds (an average FDI* of 5 seconds). Finally, with an average FDI of 7 seconds, PSO-based DiverGet has the best average FDI.

In summary, based on the two previous experiments DiverGet outperforms DiffChaser in terms of DiR and SR, with its PSO version performing a comparable FDI.

Finding 7: DiverGet outperforms DiffChaser in terms of number of revealed disagreements with a higher success rate.

5.3 Threats to Validity

In this section, we address the potential threats to the validity of our work and emphasize our prevention measures.

Table 5.10 Comparison of the median values of FDI and FDI* between DiffChaser, PSO-based DiverGet and GA-based DiverGet

Model	Framework	PU				SA				Average	
		PTQ		QAT		PTQ		QAT		FDI	FDI*
		FDI	FDI*	FDI	FDI*	FDI	FDI*	FDI	FDI*		
SSRN	DiffChaser	4.11	0.99	6.93	1.71	9.06	2.86	8.83	1.37	7.23	1.73
	DiverGet (PSO)	23.31	17.2	26.9	20.2	54.6	38.3	39.5	29.91	36.11	26.42
	DiverGet (GA)	52.38	32.99	53.71	29.85	80.99	49.62	75.49	43.51	65.64	38.99
Hybrid-SN	DiffChaser	14.0	12.8	13.6	1.99	13.5	6.07	12.81	0.63	13.51	5.39
	DiverGet (PSO)	7.27	6.46	7.57	5.48	6.82	5.49	6.16	4.14	6.96	5.39
	DiverGet (GA)	10.5	5.29	13.1	5.92	10.3	5.67	9.3	4.62	10.8	5.38

Selection of subjects: The selection of our experimental subjects like dataset, models, and quantization methods, can be a threat to validity. As a mitigation strategy, we use two variants of each evaluation subject and all of the considered variants are: (i) state-of-the-art like the tested CNNs, (ii) widely-used by the community like the HSI datasets, (iii) state-of-the-practice like the quantization techniques. Also, we use official implementations of the tested models and we rely on open-source libraries for quite delicate implementations such as quantization techniques and metaheuristic standard algorithms to prevent potential bugs.

Design choices: The design choices that we made throughout the development process can be a threat. To overcome this threat, we implement two competitive metaheuristics for SBST. Furthermore, we perform a prior phase of hyperparameters tuning to balance their configuration in regards to exploration vs exploitation, in order to be suitable for our designed optimization problem. Concerning *population_size* and *generation_number* that control the total number of generated samples, we also tried multiple combinations and we make sure that they guarantee fair comparisons with their opponents, whether it is just a random sampling or the state-of-the-art DiffChaser. Indeed, we extract the naturally-occurring distortions for HSI from remote sensing white papers and we systematically restrict the range of their parameters with regards to the source information loss (PSNR). However, we consulted HSI experts to confirm our final designed distortions, their parameters' ranges,

and the threshold of PSNR adopted in the validity test, because these choices would have an impact on the validity of our results.

Generalizability to other domains: Despite most of similar research works study conventional classification problems such as MNIST [186] and CIFAR10 [187], we opt for more complex problems, HSI classification, as study cases because of their real-world on-edge deployment challenges such as the curse of dimensionality, the prevalence of external context changes, defect-proneness of the acquisition systems. All these challenges are threatening the natural transition to on-edge devices with the conventional assessment strategy, and sheds the light on the importance of advanced quantization assessment to anticipate the risky behavioral deviations by the DNN at the edge and avoid them. Nevertheless, we describe thoroughly the methodology we follow with our domain expert collaborators to instantiate and set up different ingredients of our approach. Also, DiverGet framework is designed in a modular way to support an easy and adaptive plug-and-play configuration.

HSI Domain Selection: HSI DNNs are being employed at the edge in crucial applications related to climate changes, environmental monitoring and Astronomy. Quantization in this domain is challenged by the curse of dimensionality, the ubiquity of external-environment changes, and the defect-proneness of acquisition systems. These factors reinforce the need for advanced assessment. To the best of our knowledge, no work has addressed quantization assessment in the field of HSI. Hence, we tested DiffChaser, a general state-of-the-art computer vision testing framework that claims to be easily extendable to other domains. Its poor performance with HSI DNNs sparked the design of DiverGet, which focuses on naturally-occurring distortions and search design.

A fair comparison to RS, and DiffChaser: We used the number of generations G as metric to fix the total number of model queries allowed per testing session. For metaheuristics, we compute $G = original_test_patches \times population_size \times nb_iterations$). For RS $G = original_test_patches \times nb_samples$. Using the same original test data, we fix equal $population_size$ and $nb_iterations$ for DiverGet (GA&PSO) and DiffChaser (GA). For RS, $nb_samples = population_size \times nb_iterations$ to enable same number of queries to all testers. Also, we considered the following points for fair comparison with DiffChaser:

- Using SR in our evaluation, a metric used in DiffChaser’s paper to compare it with our work,
- Using single-mode to have same granularity as DiffChaser,
- Using f^{div} that exploits the last layer of the two DNNs to produce divergence which is

quite similar to DiffChaser’s ‘basic’ fitness function design,

- Disabling the first attack option of DiffChaser and making it generate every possible DIIs without altering its main logic.

Dealing with randomness in the search algorithms: For reliable conclusions when comparing search-based approaches, we conducted experiments that take into account the stochasticity inherent in these search algorithms. Comparisons of individual elements might compromise the validity of such experiments and hinder the validity of their outcomes. To mitigate the effects of randomness, all the results included in the empirical evaluations of our approach are the median values estimated over 50 (10 in RQ1) runs that use independently random seeds. We also used statistical hypothesis testing and effect size measurements to assess the statistical significance of our results.

5.4 Chapter Summary

In this chapter, we introduced DiverGet, a search-based software testing framework dedicated to the assessment of DNN quantization. It (i) relies on domain-specific metamorphic relations to produce semantically preserving data, (ii) leverages various population-based metaheuristic algorithms for a maximum disclosure of difference-inducing inputs, and (iii) operates with two alternative and complimentary fitness functions to guide the search. Our evaluation on HSI classification problems shows that DiverGet can successfully generate meaningful test inputs that induce disagreement among DNNs of different arithmetic precision. It substantially outperforms the state-of-the-art, DiffChaser, in detecting those quantization-induced divergences.

CHAPTER 6 AN INTENTIONAL FORGETTING, SELF-HEALING APPROACH FOR DEEP REINFORCEMENT LEARNING SYSTEMS

Deep reinforcement learning (DRL), the blend of deep learning (DL) and reinforcement learning (RL), has shown promising achievements in recent years, such as AlphaStar [22], Netflix’s customized recommendation system [23], and Facebook ReAgent [24]. This demonstrates that DRL can effectively address real-world problems including long-term planning, partial observability [60], and complicated, continuous state-action spaces. Nevertheless, DRL-based systems, as with all data-driven systems, are likely to generate incorrect or undesirable behaviors when operating in constantly changing production settings. This phenomenon is known as environmental drift [62], which is the manifestation of concept drift [156, 188] in DRL systems.

The burgeoning field of ML (and RL) operations (MLOps) (and RLOps [189]) is establishing the best practices and methods for implementing efficient, trustworthy, and maintainable self-learning systems. Among the most demanding approaches are healing strategies [68–70] that enable the ML system to recognize its malfunction after a concept shift and to adjust, accordingly, its behaviors in order to regain its performance. Conventional software healing solutions are used to identify and respond to software failures in production by making the appropriate tweaks, at runtime, to restore the system’s normal behavior [66]. The healing workflow begins with a failure detection step. Then comes a healing step, which involves executing a healing operation. Finally, a verification step ensures that the software is restored properly. Otherwise, the healing operation is repeated until the failure is resolved or no further action is possible [66]. Self-healing solutions [66, 68] describe techniques that fully automate all steps of healing without any human involvement. During a self-healing operation, a software system should adjust its own structure and behavior in production based on its interpretation of the context, the occurred failure, and its requirements [68].

Indeed, a DRL system learns its optimal policy (i.e., best actions w.r.t perceived states) by interacting with its environment [38] and by performing reward-driven learning without supervision. Hence, it can be argued that the DRL agent can adapt its policy to handle a newly shifted environment by engaging in continual learning (CL) cycles for a sufficient amount of time. In practice, various inefficiencies can adversely affect the CL ability of DRL agents, including catastrophic forgetting [65], capacity loss [190], warm-starting failure [191], and primacy bias [192]. Thus, as the environment in production drifts from its original state, CL is likely to result in slow healing cycles [193, 194] associated with prolonged system

dysfunction and downtime. Even worse, a recent study [27] reported that CL is confined within environmental parameters beyond which it is unable to heal the DRL agent. Therefore, it is of paramount importance that alternatives and improvements to vanilla CL (i.e., standard CL) be explored, in order to enlarge the adaptation frontiers and accelerate the adaptation curve for low-cost, fast, and effective self-healing DRL systems.

This chapter presents *Dr. DRL*, a self-healing method for DRL that incorporates intentional forgetting [71, 75] and continual learning in order to optimize and accelerate the agent’s healing process in the face of drifting environments. Intentional forgetting is an interdisciplinary concept that has gained traction in the software engineering community, with applications in cybersecurity [72], machine learning [71, 73, 74], and logistics supply networks [75]. Its goal is to deliberately delete unnecessary knowledge in order to focus attention on the system’s key components. To apply the intentional forgetting mechanism in DRL self-healing, we adapt the top-k neuron coverage (TKNC) [19, 31], which is a fine-grained layer coverage criterion proposed by DeepGauge [19], in order to identify the major and minor DRL agent behaviors at the level of its neural network units (i.e., neurons). Accordingly, our approach associates the DRL agent’s major behavior with hyperactive neurons (i.e., neurons with high levels of activation) and minor behavior with hypoactive neurons (i.e., neurons with low activation levels). Then, our approach deliberately suppresses the minor behaviors by assigning low-scaled random weights to their associated hypoactive neurons. As a first benefit, forgetting the minor behaviors that are likely to be non-transferable to the drifted environments, increases both the speed and the effectiveness of major behavior adaptation. As a second gain, under-scaling weight initialization as a minor behavior forgetting mechanism enables the self-healing at dual speeds, i.e., major behavior neurons (unchanged) would receive significant and more frequent updates than minor behavior neurons (under-scaled).

In order to assess the effectiveness of *Dr. DRL*, we evaluate it on deliberately drifted gym [129] settings with different drifting intensities. Results show that *Dr. DRL* succeeds in speeding up the self-healing process in terms of runtime and fine-tuning episodes, compared to vanilla CL. *Dr. DRL* extends the self-healing frontiers that CL provides by 20% on average. Finally, *Dr. DRL* can preserve or even improve the average CL reward by up to 45%.

Chapter Overview. The remainder of this chapter is structured as follows. Section 6.1 provides a concrete illustration of the problem we are addressing and how we are tackling it. Section 6.2 describes each step of our approach. Section 6.3 reports the evaluation outcomes. Section 6.4 analyzes the threats to validity. Finally, Section 6.5 concludes the chapter.

6.1 Motivation Example

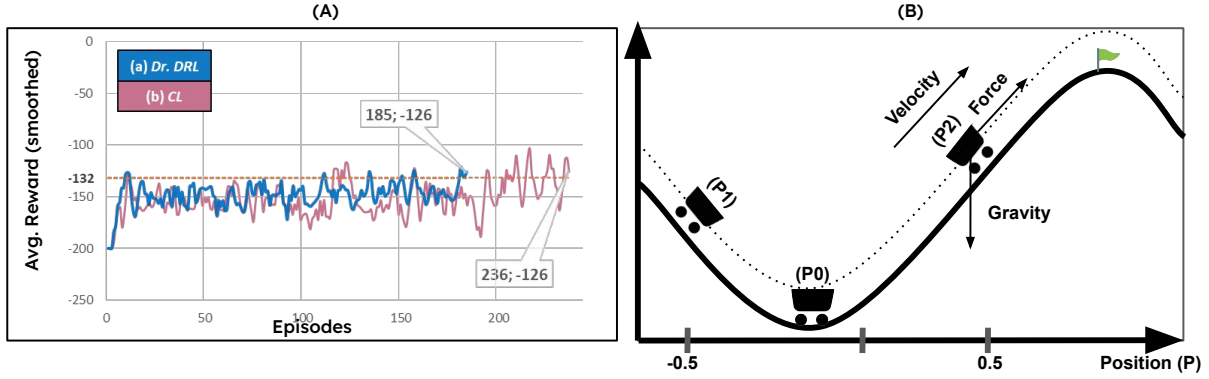


Figure 6.1 (A) The DQN agent’s average reward per episode on MountainCar adaption curves: (a) shows the average reward curve when Dr. DRL is used, whereas (b) shows the average reward curve when CL is used. (B) Graphic representation of the MountainCar environment as explained in [3].

The Mountain Car game is a simulation environment provided by the Gym library [3]. It presents a scenario where a car is positioned at the base of a valley and is restricted to one-dimensional movement down a track. The car’s sole available action is to accelerate either to the left or right direction. This setup is visually depicted in Figure 6.1. In order to successfully complete the game, the car must go towards the designated target state located at the peak of the right mountaintop, (green flag in Figure 6.1). The car begins at the valley’s lowest point, where it encounters penalties in the form of negative rewards at each time step. This design intends to promote the car to speed up its path towards the final goal. Such environments, known as deceptive reward environments, are often more challenging to solve [195]. The agent must acquire an average reward of no less than -110 across a sequence of 100 consecutive evaluation episodes in order to solve the MountainCar game. For a drifting environment, a 20% tolerance is incorporated into the calculation of the average reward; so the agent must now acquire an average reward of at least -132 during 100 consecutive episodes to solve the game.

Drifts in the MountainCar environment E can alter three parameters: force, gravity, and goal velocity, with the default values being $1e^{-3}$, $2.5e^{-3}$, and 0, respectively. In this motivation example, we first trained a DQN agent A to solve E . Then, a drift d was conducted on E , resulting in the adjustment of the force, gravity, and goal velocity parameters values of $1.2e^{-3}$, $4e^{-3}$, and 0, respectively. Consequently, A became incapable of resolving the newly-drifted environment E_d . Next, we leverage CL to fine-tune the trained A on E_d for a few

episodes.

Figure 6.1 illustrates the outcomes of vanilla CL. A required a total of 236 episodes and 436 seconds to successfully adapt to the E_d . Fortunately, when performing an intentional forgetting mechanism on the DRL agent (more details can be found in our approach, *Dr. DRL*), A was able to adapt to E_d in 185 episodes and 273 seconds. Furthermore, A reached an average reward of -122.3 when subjected to adaptation with X, in contrast to -123.7 when subjected to CL. Figure 1, also, shows that *Dr. DRL* enhanced the stability of the adaptation process through the reduction of the fluctuation width of the average reward. This motivation example showcases how improving the CL (leveraging techniques discussed in our approach *Dr. DRL*) successfully enhances the cost-effectiveness of the self-healing process.

6.2 Approach

In this section, we start by outlining the problem formulation. Then, we present our intentional forgetting method for enhanced DRL self-healing. Last, we provide an overview of the stages included in *Dr. DRL*'s workflow.

6.2.1 Problem formulation

Let E be a development RL environment, defined as a set of n parameters $\{s_1, \dots, s_n\}$, and let A_E be an agent trained on the environment E until it solves E or attains the performance goal. Next, let E' represent a new RL environment with new n parameters $\{s'_1, \dots, s'_n\}$, denoting the production RL environment. Although the A_E is trained on E , the production RL environment E' represents the same problem as E , and the A_E is expected to be able to solve E' or maintain the same level of performance, assuming that the parameters of E' do not exhibit significant deviations from E . If an important drift occurs, the performance of A_E decreases proportionally to the deviations (the greater the drift, the lower the performance). In this case, the RL agent, A_E , requires a self-healing mechanism to adapt to the drifted production environment, E' , and the healing success depends on its ability to solve E' or regain its optimal level of performance.

Conventionally, CL is the mainstream approach for self-healing against environmental drifts. Nevertheless, this approach can be inefficient due to several issues the community encounters, such as catastrophic forgetting [65] and slow adaptation [193]. In our approach, we assume that A_E in the development RL environment E learns both major and minor behaviors, where major behaviors encode critical problem-solving abilities and minor behaviors have no direct impact on problem-solving, but were necessary to achieve the desired performance on the

development environment. Structurally, the RL agent encapsulates a neural network, thus, we propose a neuronal categorization method to depict the hyperactive neurons (i.e., contribute almost to all of the predictions) and the hypoactive neurons (i.e., contribute intermittently to the predictions). The proposed neuronal categorization provides a structural representation of the problem-solving skills acquired by the optimal RL agent as a result of interaction with the development environment. Indeed, we believe that the inefficiency of CL may be partially due to the equal importance assigned to both major and minor behaviors, resulting in slow adaptation (all neurons are considered to be adapted) and catastrophic forgetting (CL may reverse the neurons’ roles from major to minor, or vice versa).

Our proposed self-healing mechanism is an improvement over vanilla CL by (i) prioritizing adjusting major behaviors to maintain the structural representation of problem-solving skills, and (ii) accelerating the behavioral adaptation using dual-speed continuous learning, in which minor behaviors are updated less frequently than major behaviors. These improvements can enhance both the speed and success of self-healing against environmental drifts, which remain unsolved by vanilla CL. In the following, we detail the main components of our approach, along with the self-healing workflow that we have adopted.

6.2.2 Intentional Forgetting for DRL Self-Healing Enhancement

The Lottery Ticket Hypothesis [196, 197] states that within deep neural networks, there exist sub-networks referred to as “winning tickets” which, when trained, can attain similar predictive performance as their original parent neural network. Winning lottery tickets is observed in the implementation of DNNs pruning strategies [198]. Indeed, these strategies involve the removal of unnecessary neurons after the training process, resulting in a reduction in the size and resource utilization of DNNs while maintaining their predictive ability.

In accordance with the aforementioned hypothesis, it is most likely that the primary behaviors of DRL agents are produced by a sub-neural network. The sub-network in question is responsible for the essential problem-solving capabilities necessary for successfully accomplishing the primary objective. This particular sub-network comprises the collection of hyperactive neurons, the primary units responsible for major behaviors. These neurons embody the fundamental skills of the agent, enabling it to handle various situations and infer appropriate actions. We hypothesize that the self-healing process should primarily focus on transferring and adapting major behaviors, which are characterized by hyperactive neural network units, to fit the drifted environment’s specificities. In contrast, minor behaviors should be learned from scratch. Minor behaviors, although lacking in transferability and direct influence on an agent’s problem-solving ability, are still essential for attaining the appropriate performance

within a particular environment.

For this purpose, we are inspired by the concept of intentional forgetting in artificially intelligent cognitive systems [71, 74], which suppress information that is deemed irrelevant and improve cognitive capabilities to focus and deal only with relevant aspects of the problem under consideration. In the next sections, we present the two key stages of our approach for improving the DRL agent's healing.

Step1: Identifying Minor/Major Behavioral Units

The initial stage of *Dr. DRL*'s self-healing process involves identifying the specific areas in the DRL system accountable for the major behavior as well as those accountable for the minor behavior. In terms of its structure, the RL agent comprises at least one neural network. Consequently, we suggest a neuronal categorization approach to represent the hyperactive neurons, which are the primary behavior units, as well as the hypoactive neurons, which are the secondary behavior units.

The activation functions in neural networks serve as the intermediary computations that facilitate the transmission of crucial information across different layers. These non-linear functions, associated with each individual neuron, decide whether a neuron should be "fired" (i.e., activated) based on the significance of its output in relation to the final prediction. As demonstrated by Equation 6.1, a representation of the DNN forward pass [199], the activations of each layer have a collective role in determining the activation of the final layer, $a^{[L]}$.

$$a^{[L]} = \sigma \left(W^{[L]} g \left(\dots g \left(W^{[1]} a^{[0]} + b^{[1]} \right) \dots \right) + b^{[L]} \right) \quad (6.1)$$

Where $W = \{W^{[l]}, \forall l \in [1, L]\}$ and $b = \{b^{[l]}, \forall l \in [1, L]\}$ are the weights and biases of a DNN.

Thus, neurons with frequent high activation scores encode features linked with primary behaviors, whereas neurons with frequent low activation scores express secondary behavior features while still contributing to the model's generalizability and confidence. These hypoactive neurons are the ones that contributed the least to prediction in the original environment. Hence, we intend to forget them before adjusting the neural network encapsulated by the DRL agent on a drifted environment.

Numerous Empirical studies [19, 31] have demonstrated that hyperactive neurons contribute to advantageous learning patterns for deep neural networks. The inspiration for the iden-

tification of hyper/hypoactive neurons stemmed from the Top-K-Neuron-Coverage (TKNC) approach [19]. The TKNC aims to evaluate the heterogeneity of activation patterns seen during DNN testing. In the context of TKNC, neurons that exhibit higher levels of activation are considered to be more prominent in the processing of incoming data and are thus assigned higher ranks. Conversely, neurons with lower activation levels are assigned lower ranks.

To distinguish between hypoactive and hyperactive neurons in *Dr. DRL*, we choose N_H hypoactive neurons per layer, i.e., those with low activation scores regularly. Using the next formula:

$$N_H^l = \text{Round}((F_r/100) * N_l) \quad (6.2)$$

N_H is a relative value obtained from a predetermined forget rate, F_r . Also, N_H^l indicates the number of hypoactive neurons in layer l and N_l represents the total number of neurons in layer l .

In the following, we discuss further the effects of weight scaling on the magnitude of gradient updates received by the weights. We also discuss how our intentional forgetting mechanism leverages the weight-gradient scaling relationship to enable dual-speed continual learning that is better suited to drifted environments.

Step 2: Dual Speed Self-Healing

Following the successful identification of the major/minor behavioral units in the DRL system, the second step of *Dr. DRL* involves deliberately resetting the minor behavioral units and engaging in a dual-speed gradient-based healing process. We hypothesize that CL’s inefficiency may be partially explained by two factors. Firstly, the equal weighting given to major/minor behavioral units may lead to prolonged adaptation periods. Secondly, the phenomenon of catastrophic forgetting may occur, where CL switches the roles of neurons between major and minor.

Our method presents two enhancements to CL. The first improvement involves prioritizing the adaptation of important behaviors above minor behaviors that are less likely to be applicable in new situations. This is anticipated to boost the efficiency and efficacy of the self-healing processes. The second improvement is the implementation of a minor behavior-forgetting mechanism, which allows dual-speed self-healing. Specifically, major behavior units undergo

more frequent and substantial updates compared to minor behavior units. However, if minor behavior units are nullified, it would have a detrimental impact on the asymmetry of the layer’s weight distribution. Additionally, the loss gradients with respect to null weights would become zero, causing the neurons associated with these weights to become frozen and cease receiving updates. In the following, we discuss further the effects of weight scaling on the magnitude of gradient updates received by the weights.

Our approach for effectively resetting minor behaviors entails re-initializing the neurons associated with these behaviors by employing the original weight initializer. Random weights are sampled in order to preserve the asymmetry across neurons and adjust the variance of the value distribution in response to the input size of the layer. In addition, the random weights created for hypoactive neurons are scaled down using a scale rate, S_r , which is less than 1.

$$\frac{\partial \mathcal{L}}{\partial z^{[1]}} = [W^{[2]}]^T \dots [W^{[L]}]^T \cdot \frac{\partial \mathcal{L}}{\partial z^{[L]}} * q^{[L-1]} * \dots * q^{[1]} \quad (6.3)$$

$$W^{(i+1)} = W^{(i)} - \eta \frac{\partial \mathcal{L}}{\partial W^{[l]}} \quad ; \quad b^{(i+1)} = b^{(i)} - \eta \frac{\partial \mathcal{L}}{\partial b^{[l]}} \quad (6.4)$$

Equations 6.3 and 6.4 illustrate the process by which the parameters of the model are updated at the iteration $i + 1$. In this context, $\mathcal{L}$ represents the loss function, while $z^{[l]}$, $W^{[l]}$, and $b^{[l]}$ denote respectively the weighted-sum outputs, weights, and biases associated with a layer l . The variable $q^{[l]} = g^{[l]'}(z^{[l]})$, where $g^{[l]'}$ denotes the gradients in the layer l . Equation 6.3 demonstrates that the updates based on gradient descent involve the multiplication of weights. Therefore, the initial magnitude of weights influences their update and determines their rate of growth and decay over the iterations. When the value of S_r approaches zero, weights would become significantly tiny, approaching zero. Consequently, the neurons associated with these weights would undergo updates at low rates. When the value of S_r approaches 1, minor behavior neurons will have unaltered scale weights and will undergo learning at a comparable pace to major behavior neurons. Therefore, the decrease in weight magnitude has a direct impact on the size of gradient updates obtained from backpropagation across layers.

On the one hand, our proposed intentional forgetting mechanism reduces the weights’ magnitude of reinitialized hypoactive neurons, ensuring gradual re-learning of minor behaviors by the DRL agent. On the other hand, it preserves the original weights’ magnitude of the hyperactive neurons, ensuring significant updates of the major behaviors of the agent. This approach promotes stable and rapid adaptation

6.2.3 *Dr. DRL*: The Self-healing Workflow

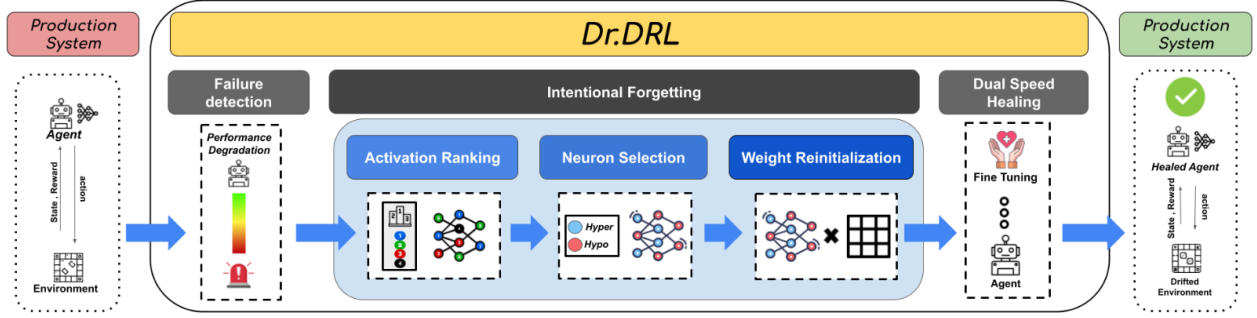


Figure 6.2 *Dr. DRL*: The Self-healing Workflow

The overview of our proposed self-healing workflow is illustrated in Figure 6.2. Below are details of the workflow’s steps.

Pre-deployment minor behavior identification. *Dr. DRL* requires a pre-deployment step in which it collects the activation traces of the agent’s DNN at the end of development. During the training process, the agent should learn how to map between states and actions properly to maximize the accumulated rewards received in the original environment [15]. Once an optimal DRL agent has been identified, we run it on samples of observations taken from the original training episodes and save the activation traces.

Failure detection. In *Dr. DRL*, we develop an event handler component that monitors the production environment and triggers the healing mechanism when an environmental drift occurs. Upon a certain level of parameter shifts, the trained agent loses its ability to predict optimal actions, which means the agent cannot reach the required minimum reward. In that case, the actual agent fails to solve the newly-drifted environment, and consequently, our proposed self-healing mechanism is triggered.

Self-Healing mechanism. *Dr. DRL* first identifies the trained agent’s minor behavior regions (i.e., hypoactive neurons) using the collected activation traces. In each layer, we rank the neurons based on their activations, and then we select the hypoactive neurons, i.e., triggering lower activations over most of the observations (lines 2 to 6). The hypoactive neurons per layer constitute the minor behavior neurons, whose counts are derived according to the prefixed F_r (See Section 6.2.2). *Dr. DRL* then intentionally erases the DRL agent’s minor behaviors and proceeds to dual-speed gradient-based healing. Algorithm 2 illustrates the steps of *Dr. DRL*’s healing mechanism. First, we reinitialize the weights associated with minor behavior neurons using the same weight initializer but under-scaled by S_r in order to prioritize the updates of major over minor behavior neurons (lines 7 to 11) (See Section

6.2.2). Second, we adjust the hyperparameters of the DRL algorithm, especially those that control how much exploration the DRL agent does over the training. We reload the replay memory obtained from previous training episodes on the original environment in order to preserve the initial performance (only for off-policy RL algorithms like DQN and SAC). As soon as the setup is complete, we start fine-tuning the partially-reinitialized DRL agent on the drifted environment (lines 12 to 14) (See Section 6.2.2). The objective is to heal the agent from the involved environmental drifts, i.e., achieve the same level of performance as the original environment.

Algorithm 2: Dr. DRL’s healing mechanism

Input: $E' = \{p'_1, \dots, p'_n\}$ drifted environment, A_E : trained agent that solves $E = \{p_1, \dots, p_n\}$, n_t : training episodes, obs : set of training observations, F_r : Forget rate, s_r : Scale rate, $init$: weights initializer, hyp : hyperparameters of the agent.

Output: $A'_{E'}$: agent adapted to the drifted environment E'

```

1:  $mask := []$ ;
   /* Detection of Minor Behavior : */
2:  $acts := \text{compute\_activation\_trace}(A_E, obs)$ 
3: for  $i$  in  $A_E.hidden\_layers$  do
4:    $hypo\_n := \text{detect\_minor\_regions}(A_E, obs, F_r)$ ;
5:    $masks.append(\text{generate\_mask}(A_E), i, hyp, hypo\_n)$ ;
6: end for
   /* Intentional Forgetting of Minor Behavior : */
7: for  $i$  in  $A_E.hidden\_layers$  do
8:    $w := \text{apply\_mask}(weights\_mask, A_E)$ ;
9:    $new\_w := \text{reinitialize}(w, masks[i], s_r, init)$ ;
10:   $A_E.set\_weights(new\_w)$ 
11: end for
   /* Dual Speed Gradient-based Healing : */
12: for  $eps$  in  $n_t$  do
13:   $A'_{E'} := \text{dual\_speed\_cl}(A_E, E', hyp)$ ;
14: end for
15: return  $A'_{E'}$ 

```

6.3 Evaluation

In this section, we introduce our evaluation subject systems, including both the environments and algorithms employed, and outline the experimental settings, metrics, and protocol, used to assess the performance of *Dr. DRL* in comparison to vanilla CL. We provide answers to research questions related to: (RQ1) cost-efficiency, (RQ2) enhancement of adaptation, and (RQ3) improvement in reward.

6.3.1 Experimental Subjects

Environments

We leverage the following 3 different environments in our empirical evaluation.

The MountainCar environment [3] was described in Section 6.1.

The CartPole environment [200] consists of an inverted pendulum attached to a cart that moves on a track controlled by a force. The agent’s goal is to keep the pole upright, receiving a reward of +1 for each successful time step. The task is episodic, and the episode ends if the pole falls $> 15^\circ$, the cart moves > 2.4 units from the center, or 200-time steps pass. To solve the task, an agent must achieve an average reward of > 195 over 100 testing episodes. The CartPole environment is parameterized by four parameters: masspole, length pole, mass cart, and friction.

The Acrobot environment [201, 202] has two links and two joints, and the goal is to raise the lower link to a specified height. If the agent achieves the goal, it gets a reward of 0, otherwise, it receives a reward of -1 at each time step. Episodes end when the goal or 500 time steps are reached, and the reward threshold for the environment is -100 over 100 evaluation episodes. The Acrobot environment is parameterized by four parameters: ‘link length 1’, ‘link compos 1’, ‘link mass 1’, and ‘link mass 2’.

Algorithms

We introduce, in this section, the subject algorithms used in our *Dr. DRL* assessment. These algorithms are extensively used and stable in most RL libraries.

Deep Q-Learning (DQN) [203, 204] is a widely utilized value-based algorithm that uses a neural network to approximate the action-value function, Q . The convergence of iterative learning of Q cannot be guaranteed, since it may be susceptible to instabilities. However, there are two strategies that may mitigate these instabilities: the utilization of a buffer to replay previous experiences and the introduction of a second target DNN.

Proximal Policy Optimization (PPO) [205] algorithm is a policy-based algorithm that aims to achieve optimal policy optimization while maintaining high-performance levels. To

ensure policy stability, the PPO algorithm employs a clipping mechanism that restricts excessive modifications to the policy. Additionally, it incorporates a hyperparameter, ϵ , which governs the extent to which the policy can deviate from its previous state.

Soft Actor Critic (SAC) [206] is a DRL hybrid algorithm. Its primary objective is to optimize returns while also considering the level of policy entropy, which refers to the degree of randomness in the decision-making process. The SAC algorithm also conducts its updates in an off-policy manner, the same as DQN.

Protocol and Metrics

Protocol: The hyperparameter values of the DRL algorithms were taken from rl-baselines-zoo [207]. We then carried out 10 separate training runs using different random seeds to account for the potential impact of implementation modifications on the performance of a DRL algorithm. The selection of the best hyperparameters for each DRL algorithm was based on the criterion of achieving the largest average reward. Six environmental drifts with varying magnitude shifts were sampled for each DRL algorithm. We use the parameter ranges provided in [27] to differentiate between the simulated drift intensities. Next, we employ either *Dr. DRL* or the conventional baseline strategy, vanilla CL, to heal the performance of the DRL agent across the six shifted environments. In order to provide an equitable comparison, we employed identical hyperparameters and the maximum number of training episodes for DRL agents. However, we tuned the particular parameters, F_r and S_r , of *Dr. DRL* for each environment by using the ranges of [50, 40, 30, 20, 10]% and [0.00001, 0.0001, 0.001, 0.01, 0.1] respectively. The optimal settings for the CartPole environment were $F_r=50\%$ and $S_r=0.1$. In the case of both the Acrobot and MountainCar, using F_r of 10% and S_r of 0.0001 yields the best performance. Each time, F_r and S_r values were kept the same across all neural network layers. We hypothesize that the depth and type of layer may have an effect on the values of F_r and S_r . We did not thoroughly examine this topic in our study, and it could serve as the focus of future studies.

In accordance with the recommendations in the previous study [27], we added an adaptation tolerance ratio of 20% for the MountainCar and Acrobot environments. Consequently, a performance decline of 20% is accepted during the healing process. In the MountainCar for example, the agent solves the drifting environment by achieving an average reward of -132 . This value is derived by subtracting 20% from -110 , the minimal average reward threshold for solving the original environment (see Figure 6.1(A)).

In order to mitigate the inherent stochasticity of these DRL algorithms, a total of ten repetitions of each experiment are conducted. The Wilcoxon statistical significance test [181] and the Vargha-Delaney A12 effect size [182] are also employed to compare the outcomes of *Dr. DRL* and vanilla CL.

Environmental Drifts: Based on the findings of [27] on the plasticity of DRL agents, two categories of shifted environments have been discerned. The first category comprises adaptable environments, wherein the agent is able to adapt. Conversely, the second category encompasses non-adaptable environments, wherein the agent is unable to adapt. In our study, we take into account both categories.

The objective regarding the adaptable category is to assess the stability of our self-healing method, as well as its efficacy in optimizing the healing of the DRL agent. In relation to the non-adaptable categories, our objective is to examine the extent to which our self-healing method enhances the healing capability of the DRL agent and facilitates its adaption in spite of CL failure.

Metrics: In the next section, we present the metrics that have been employed in our assessments.

Increase Ratio (%IR(X)) / Decrease Ratio (%DR(X)). The equation Eq. 6.5/Eq. 6.6 represents the percentage of increase/decrease in quantity X resulting from the application of DR instead of CL. Thus, positive numbers signify an enhancement in the observed variable, X, whereas negative ones imply a decline.

$$\%IR(X) = \frac{Dr. \ DRL_X - CL_X}{CL_X} \times 100 \quad (6.5)$$

$$\%DR(X) = \frac{CL_X - Dr. \ DRL_X}{CL_X} \times 100 \quad (6.6)$$

Adaptation Ratio (%AR). Equation Eq. 6.7 represents the proportion of pairs (i.e., (drifted environment, agent)) for which the method Y is capable of healing.

$$\%AR(Y) = \frac{(Env, Agent)_{adpt(Y)}}{(Env, Agent)_{All}} \times 100 \quad (6.7)$$

6.3.2 Research Questions and Answers

RQ1: Is *Dr. DRL*'s healing more cost-effective than continual learning ?

Motivation: The objective is to evaluate the cost-effectiveness of healing DRL systems in drifting environments between *Dr. DRL* and vanilla CL. Fine-tuning DRL systems to operate in a newly-drifted environment requires simulation and/or live deployment, which can be resource-intensive and costly. Consequently, delays in healing the behavior of the DRL agent can result in substantial cost penalties, and the effectiveness of healing may be hindered by a high number of required trials, in order to adapt the agent's behavior. As a result, the more successful a healing technique is, the less time it requires.

Method: To compare the cost-effectiveness of *Dr. DRL* and CL, we measured the amount of time and episodes necessary to adapt a DRL agent in a drifted environment. An episode ends when the agent has reached its goal or failed to reach it after completing the maximum number of steps allowed in each episode, which is determined by the specific environment. Hence, we track how long the agent spends adjusting to the drifted environment and how many episodes it takes for the evaluated strategy to complete the adaptation. To meet the completion criteria, the agent must achieve an average reward over 100 consecutive episodes greater or equal to the threshold set by the environment. To have a better understanding of the magnitude of speed-up gained when using *Dr. DRL* compared to CL, we compute the decrease ratio, %DR (Eq. 6.6), of both the training time and the training episodes.

Table 6.1 Comparison of Healing Time and Fine-Tuning Episodes Decreases Using *Dr. DRL* over vanilla CL.

		DQN	PPO	SAC
Episodes DR (%)	CartPole	21.2	<u>45.9</u>	0.9
	MountainCar	17.7	<u>2.9</u>	20.1
	Acrobot	3.2	9.3	<u>38.3</u>
Time DR (%)	CartPole	<u>30.3</u>	-20.5	1
	MountainCar	20.3	<u>24.4</u>	<u>18.7</u>
	Acrobot	5.3	16.2	<u>73</u>

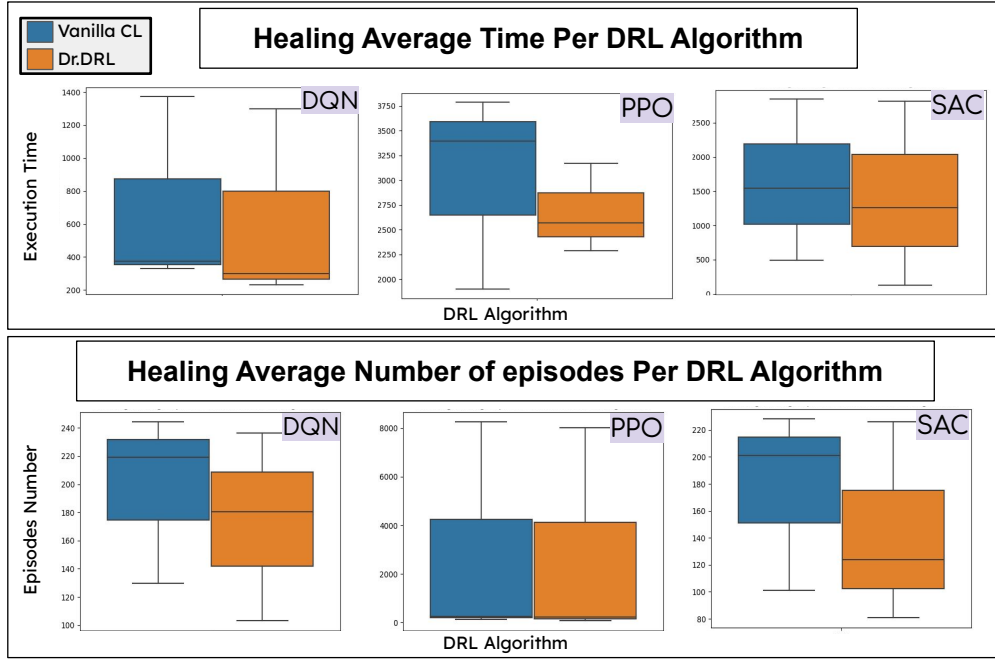


Figure 6.3 Box Plots of Healing Time and Episode Fine-Tuning Using *Dr. DRL* and vanilla CL.

Results: Table 6.1 reports the DRL agent’s running time and number of episodes required to heal its logic to the newly shifted environment using our approach, *Dr. DRL*, and vanilla CL. The bold values demonstrate that the difference between the two healing methods is statistically significant (p-value is below 0.05), while the underlined values imply medium or large effect size A12. Besides, box plots in Figure 6.3, compare the time and number of episodes required for *Dr. DRL* versus vanilla continual learning to heal the DRL agent. The median values of both time and episodes are consistently lower for *Dr. DRL* than for vanilla continual learning.

Overall, *Dr. DRL* outperforms CL in terms of cost-effectiveness. As can be seen, *Dr. DRL* provides a faster adaptation time for DRL agents than CL, except for the study case of PPO/CartPole. The difference between the two averaged values is statistically significant in 56% of the evaluated configurations and of a substantial magnitude in 33% of the configurations. In terms of fine-tuning episodes, *Dr. DRL* uses fewer episodes to adapt the DRL agent to drifted environments than CL across all the pairs of agents and environments, with the difference between the two healing approaches being statistically significant in 44% of tests with a medium or large effect size. Especially, *Dr. DRL* adapts PPO to the drifted CartPole with very few episodes, meaning the overhead of execution time was caused by longer time steps. These results demonstrate the effectiveness of our approach, *Dr. DRL*, in lowering the

cost of healing the agent’s behavior by reducing the sample complexity, i.e., the number of training episodes, in order to determine the best policy. Investigating the sample complexity of RL algorithms has been a major focus of research studies [208], and enhancing the sample efficiency is crucial to expanding DRL-based systems in real-world applications.

Finding 1. *Dr. DRL* provides a sample/cost-efficient healing of DRL agents to drifted environments, resulting in fewer fine-tuning episodes and lower healing time.

RQ2: To what extent does *Dr. DRL* enhance the self-healing of DRL agents to drifted environments?

Motivation: It was recently demonstrated [27] that CL has an environmental boundary beyond which it cannot heal the DRL agent. Our proposed approach was established with the intention of expanding this boundary. In this RQ, we aim to determine if *Dr. DRL* can heal DRL agents to drifted environments that cannot be resolved through CL and vice versa.

Method: To assess the adaptability of DRL agents resulting from different healing methods, we compute their adaptability ratios (%AR) (Eq. 6.7). Then, we expect four possible scenarios:

1. agent-environment pairs adapted by both healing methods;
2. agent-environment pairs adapted exclusively by *Dr. DRL*;
3. agent-environment pairs adapted exclusively by vanilla CL; and
4. agent-environment pairs not adapted by both methods.

Results: Figure 6.4 illustrates two components: (a) a Pie chart that presents the adaptation ratios (AR) attained by *Dr. DRL*, vanilla continuing learning, simultaneously both, or none; and (b) Pie charts that provide a breakdown of the adaptation ratios based on different environments. As seen in Figure 6.4-a, our approach effectively heals almost all agent-environment pairs that are successfully healed by CL. Furthermore, *Dr. DRL* expanded the adaptation boundary of CL.

Interestingly, our approach is able to heal about 20% of agent-environment pairs on which vanilla CL fails. This proves the effectiveness of *Dr. DRL* in the enhancement of the DRL system adaptability against a subset of previously deemed non-adaptable environmental drifts, which results in extending the frontiers of CL. Furthermore, *Dr. DRL* manages to heal almost all agent-environment pairs that are successfully healed by CL. Indeed, the cases where

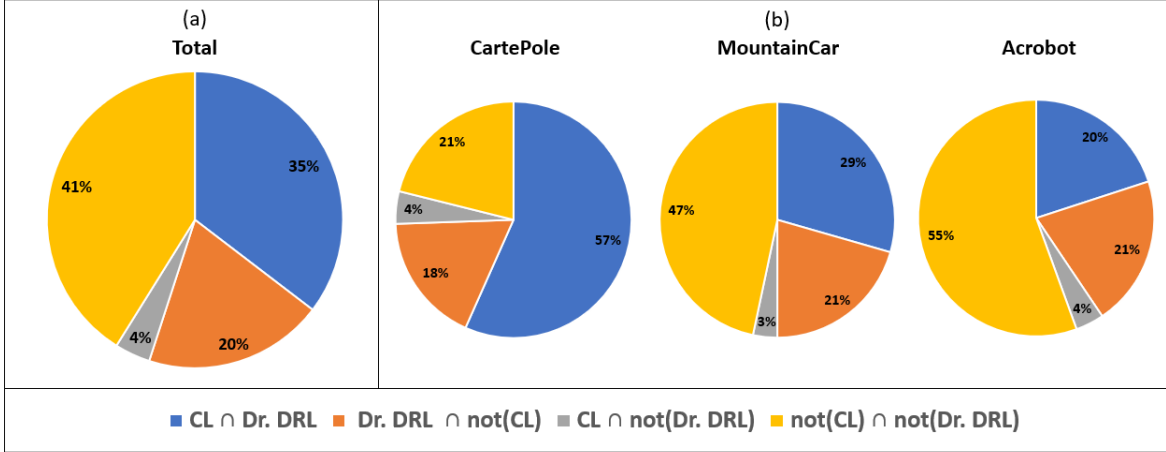


Figure 6.4 %AR distribution across the four identified groups by healing method and RL environment

Dr. DRL fails against CL are very few, about 4%. The reason is that any reduction of parameters (i.e., forgetting of hypoactive neurons), no matter how well regulated, can cause information loss, that may lead to inefficiencies in the DRL system (e.g., diminished adaptation). The findings of RQ 1 indicate that our intentional forgetting mechanism has resulted in cost-effectiveness in relation to the sample complexity. Therefore, the presence of these failures against CL, which accounts for 4%, may be eliminated by decreasing the proportion of reset minor behavior units during the *Dr. DRL*'s healing trials. In other words, if fewer minor behavior units are reset, our approach will exhibit close behavior to vanilla CL. As a result, *Dr. DRL* not only preserves but also expands CL's adaptability frontiers, enhancing the DRL system's self-healing. Consistent with the preceding study on the plasticity of DRL agents [27], there is a substantial proportion of drifted environments that are beyond the capability of vanilla CL and *Dr. DRL*. This suggests that the continuous self-healing of pre-trained in-production DRL systems to these environments with constantly shifting parameters has limits. Therefore, for some environmental drifts, the redesign of the DRL system or re-training from scratch can be more suitable to solve the problem.

Based on Figure 6.4-b, it can be observed that the portion of agent-environment pairs solely adapted by one of the two healing strategies remains quite consistent throughout the environments under study. In contrast, the portion of pairs that are adapted, or not, using both approaches varies significantly according to the specific environment under study. Therefore, although their differences are independent of the environment, both healing methods degrade in performance when exposed to challenging drifted environments, i.e., incorporating high-dimensional and complex settings, leading to less successfully healed DRL agents.

The field of healing software systems is now seeing significant activity and ongoing progress in the creation of self-healing capabilities for modern systems. This is aimed at effectively addressing the dynamic operational demands that arise from evolving needs [69]. In the field of DRL systems, our proposed approach, *Dr. DRL*, aligns with the ongoing efforts. By enhancing the success ratio of healing from 39% ($35\% + 4\%$) to 55% ($35\% + 20\%$), *Dr. DRL* offers a significant advancement in ensuring the self-healing capability of DRL systems when faced with environmental drifts commonly encountered in real-world production conditions.

Finding 2. *Dr. DRL* surpasses vanilla CL in effectively healing a greater proportion of DRL systems to drifted environments, hence extending the adaptation boundaries of the vanilla CL.

RQ3: Are *Dr. DRL*'s adapted agents more performant compared to their CL's counterparts in terms of reward?

Motivation: Predictive performance remains the main selection criterion for any DRL system. Thus, restoring or even exceeding the original performance in drifted environments after healing is an important factor in evaluating a DRL system's self-healing approach. Hence, we conduct a comparison between *Dr. DRL* and vanilla CL in terms of the resulting healed DRL system's predictive performance in drifted environments.

Method: We compute the average of rewards obtained by the healed agent over the course of 100 episodes, using the two compared self-healing approaches, *Dr. DRL* and vanilla CL. Then, we calculate *Dr. DRL*'s average reward increase ratio, %IR (Eq. 6.5), w.r.t the vanilla CL's average reward. Using the above-mentioned metrics, we solely consider the two following situations: (i) jointly adaptable environments, i.e., the agent-environment pairs on which vanilla CL and *Dr. DRL* both succeeded; (ii) jointly non-adaptable environments, i.e., the agent-environment pairs on which CL and *Dr. DRL* both failed. The remaining situations, where one approach, either vanilla CL or *Dr. DRL*, succeeds and the other fails, are irrelevant for predictive performance comparison because the gap between them will be wide due to the failure of one of them in healing the DRL system.

Results: The first row of Table 6.2 shows the reward increase ratio of *Dr. DRL* compared to vanilla CL in jointly adaptable environments. *Dr. DRL* increases the average reward obtained in 22% of experiments by at least 10%, while maintaining an almost equal average reward in the remaining experiments. The second row of Table 6.2 shows the reward increase ratio of *Dr. DRL* compared to vanilla CL in jointly non-adaptable environments. *Dr. DRL* was able to improve the average reward in 78% of the configurations by up to 45% of the

Table 6.2 Comparison of healed DRL Agent Reward Increases Using Dr.DRL over CL.

		DQN	PPO	SAC
Reward IR (%)	CartPole	-0.1	-0.06	<u>-0.2</u>
	MountainCar	0.8	-0.5	<u>10.4</u>
	Acrobot	-0.5	-1.2	<u>13.1</u>
Reward IR (%)	CartPole	<u>-34.9</u>	45.7	2.2
	MountainCar	<u>3.5</u>	<u>25.8</u>	-15.4
	Acrobot	<u>14.2</u>	13.5	<u>21.2</u>

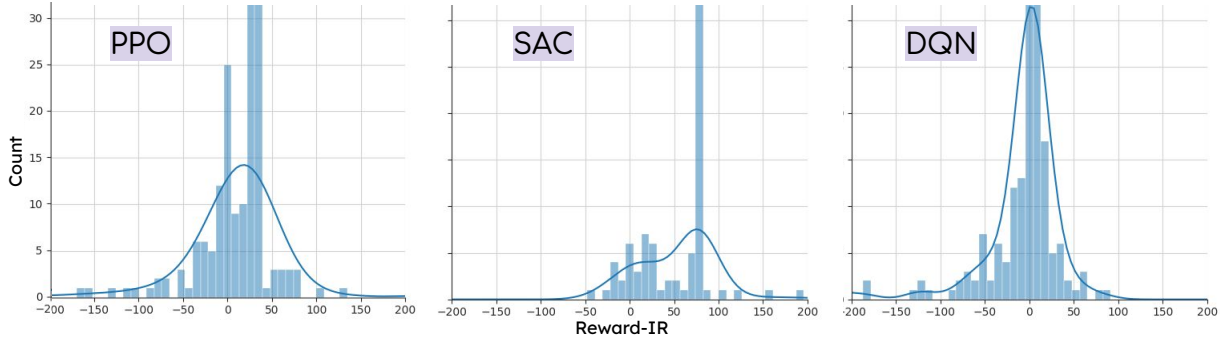


Figure 6.5 A Comparative Analysis of Average Reward Increase in Non-Adaptable Environments Utilising Dr.DRL vs CL.

increase ratio over vanilla CL's average reward. Table 6.2 demonstrates that *Dr. DRL* first succeeds in improving or at least maintaining the baseline average reward achieved by CL in jointly adaptable environments. Second, *Dr. DRL* meets average rewards better than vanilla CL in drifted environments which both self-healing approaches cannot resolve.

For a more in-depth view, we show Figure 6.5, containing the distributions of *Dr. DRL*'s reward increase ratio (averaged by DRL algorithms) over vanilla CL in jointly non-adaptable environments. Overall, distributions follow a curve that tends to skew towards the right (increase ratio ≥ 0). These left-skewed distributions (also called right-leaning curves) show that our DRL system's self-healing approach, *Dr. DRL*, is able to exceed vanilla CL's predictive performance in non-adaptable environments.

Finding 3: *Dr. DRL* outperformed vanilla CL in both adaptable and non-adaptable environments, demonstrating its superior performance in meeting requirements.

6.4 Threats to Validity

This section presents the possible threats to the validity of our study and highlights the actions we have taken to mitigate these concerns.

Generalizability to other DRL systems. The choice of our experimental subjects, such as DRL algorithms and RL environment, may threaten the generalizability of our study. While acknowledging the need for more subjects to comprehensively evaluate the generalizability of our findings, we have selected three distinct and well-established RL environments that are often employed within the DRL community. Furthermore, these environments have been incorporated into the gym library [129], further enhancing their accessibility and utility. Moreover, our approach has been evaluated on three Deep DRL algorithms that vary in type and possess unique learning methodologies.

Dealing with randomness in the experiments. The stochasticity inherent in the studied DRL algorithms and RL environments can be a threat to the validity of our findings. In order to address these risks, we conducted each experiment on both *Dr. DRL* and vanilla CL 60 times (6 RL environments with distinct parameter drifts, each repeated 10 times) using a range of optimized DRL agents. The purpose of training 10 agents for each DRL algorithm using distinct random seeds was to mitigate the inherent randomness of algorithms. Finally, to establish the statistical significance of our findings, we conducted statistical hypothesis testing and effect size evaluations utilizing the non-parametric Wilcoxon test [181] and the Vargha-Delaney effect size [182].

A fair comparison to vanilla CL. Both *Dr. DRL* and CL have been configured with the same parameters, such as the number of training episodes and the maximum allowable steps per episode, for all pairs of RL environments and DRL algorithms that were investigated.

Reproducibility. In order to guarantee the reproducibility of our evaluation, we offer a comprehensive replication package [29]. This package encompasses the complete source code of *Dr. DRL*, along with all necessary information and hyperparameters essential for replicating our assessment. This includes explicit details regarding the architecture of the tested agents and their corresponding neural networks. Moreover, section 6.3.1 provides a thorough explanation of the fine-tuning procedures undertaken and the precise values of critical hyperparameters, notably S_r and F_r , essential for accurately reproducing our evaluation process.

6.5 Chapter Summary

In this chapter, we propose, *Dr. DRL*, a novel DRL self-healing approach, which involves an intentional forgetting step to systematically prioritize the adaptation of a DRL agent’s

problem-solving skills when an environmental drift occurs in production settings. Indeed, *Dr. DRL* identifies the agent’s minor behaviors by watching the neurons with low-ranked activations over training episodes, called hypoactive neurons. If a drift in the environment happens and the agent becomes unable to solve it, *Dr. DRL* heals the DRL agent using two steps: (i) it re-initializes the weights associated with hypoactive neurons with low-scaled values, and (ii) performs continual learning updates of the patched DRL agent on the newly-drifted environment. By doing that, *Dr. DRL* enables the adaptation of the DRL agent’s behaviors at dual speeds, i.e., untouched major behavior neurons maintain larger updates compared to under-scaled minor behavior neurons. In order to evaluate our proposed improvement, we conduct self-healing experiments with different DRL agents on drifted environments with varying parameter shifts using *Dr. DRL* and vanilla continual learning. The results show that our proposed approach offers (i) faster healing of the DRL agent, (ii) a higher number of adaptable environments, and (iii) equal or more elevated average rewards than vanilla CL.

CHAPTER 7 CONCLUSION

This chapter provides an overview of our research findings and highlights potential avenues for future investigation.

7.1 Summary

Deep Learning (DL) and Deep Reinforcement Learning (DRL), the two hottest disciplines of machine learning, both play critical roles in establishing human-level or super-human AI systems. In contrast to conventional ML disciplines, DL and DRL are founded on Deep Neural Networks (DNNs), non-linear function approximators that create higher-level data representations. Using a neural network facilitates the reduction and transformation of complex high-dimensional data into a feature space of lower dimensions. DNNs enable these two ML fields to transcend the curse of dimensionality and achieve human-level autonomy in previously intractable fields like robotics, computer gaming, and computer vision. However, the barrier to ensuring reliability and efficient operationalization of DL and DRL systems continues to hinder the widespread adoption of AI across various industries. Numerous DL and DRL firms are encountering difficulties in implementing MLOps concepts and transitioning from a proof-of-concept solution to a reliable ML system suitable for production. This is primarily owing to the current slow, manual, and complex process of operationalizing ML systems, which necessitates increased commitment and focus from both academia and industry.

This thesis addresses the reliability and production readiness of DL and DRL systems. More precisely, we propose (i) an empirical study to identify DRL system deployment challenges, (ii) a search-based testing approach to evaluate DNN quantization (a post-training compression strategy), and (iii) a self-healing DRL approach to accelerate agent adaptation to environment drift. In the following, we provide an overview of the studies and approaches presented in this thesis:

An empirical study on DRL deployment challenges: We conducted an empirical study on Stack Overflow (SO) with the aim of understanding the difficulties encountered by practitioners during the deployment of DRL-based systems. The study examined 357 SO posts on DRL deployment. The findings indicate a noticeable increase in the overall level of interest surrounding DRL deployment. Our findings also show that DRL deployment is more difficult than other aspects of DRL system development, prompt-

ing us to study its unique challenges. To that purpose, we created a taxonomy of 31 distinct challenges for deploying DRL on various platforms. DRL deployment has distinct issues, such as the management of the RL environment, the loading and saving of the agent, and continuous learning. The distinctiveness of DRL arises from its specific design and learning paradigm. Unlike DL and traditional software, establishing DRL systems necessitates a focus on the environment, agents, and their interactions. We believe that this study encourages further research and assists the community in addressing the most common and challenging difficulties in DRL-based system deployment.

DiverGet: A search-based testing approach for quantization assessment: This study presents DiverGet, a search-based software testing approach for DNN quantization assessment. DiverGet leverages domain-specific metamorphic relations to generate semantically preserving data. Additionally, it employs a range of population-based metaheuristic algorithms to maximize the identification of difference-inducing inputs. Furthermore, DiverGet operates with two distinct and complementary fitness functions to effectively guide the search. The findings of our assessment on the HSI domain demonstrate that DiverGet generates significant test inputs that elicit disagreement across DNNs with varying levels of arithmetic precision. It also outperforms the state-of-the-art, DiffChaser, in identifying those quantization-induced divergences. We describe our systematic approach for creating novel domain-specific metamorphic relations. This would make it easier for the community to adapt DiverGet to evaluate on-edge quantization side effects in other safety-critical areas.

Dr.DRL: A self-healing approach for DRL systems: We propose Dr. DRL, a DRL self-healing technique that uses an intentional forgetting strategy in order to systematically prioritize the adaptation of DRL agents when an environmental drift happens in production settings. Dr. DRL effectively discerns the minor behaviors of the agent by the observation of neurons exhibiting low levels of activity during training iterations, commonly referred to as hypoactive neurons. In the event of an environmental drift that renders the agent incapable of resolving it, Dr. DRL employs a two-steps to restore the DRL agent’s functionality. The weights associated with hypoactive neurons are re-initialized with low-scaled values. Continual learning updates of the patched DRL agent are then performed on the newly-drifted environment. Dr. DRL does this by allowing the DRL agent’s behaviors to be adapted at dual speeds, i.e., undisturbed major behavior neurons preserve greater updates than under-scaled minor behavior neurons. To assess the efficacy of Dr. DRL, we conducted a series of experiments using

self-healing mechanisms applied to various DRL agents. The evaluation was performed using two approaches: Dr. DRL and vanilla continuous learning. The findings indicate that Dr. DRL exhibits (i) faster healing of the DRL agent, (ii) a greater range of adaptive environment, and (iii) comparable or higher average rewards compared to vanilla CL.

The research studies presented in this thesis jointly provide valuable approaches that can be effectively included in the MLOps pipeline to enhance the reliable operationalization of ML systems. In the first study, the empirical investigation of DRL deployment challenges on several platforms yielded actionable insights for the MLOps practitioner. Understanding the complexities and limitations of deploying DRL systems provides MLOps teams with targeted strategies to enhance the deployment processes. In the second study, DiverGet was designed as a robust mechanism for assessing model behavior alterations under quantization. By including such a testing methodology in the MLOps pipeline, quantization approaches are systematically evaluated, increasing the reliability of DL models in production. Finally, in the third study, Dr. DRL, a self-healing approach is introduced. Integrating Dr. DRL into the MLOps pipeline allows for easier monitoring and automated adaption of DRL agents to environmental drift during deployment, ensuring sustained performance. While the experiments alternate between DRL and DL, the approaches used, such as intentional forgetting and search-based software testing, show promise for generalizability and adaptability to a wide range of ML applications. Indeed, the intentional forgetting mechanism described in Dr. DRL is easily adaptable to any DNN-based system. Similarly, DiverGet’s search-based testing technique, which was originally designed for DL, can be applied to DRL systems.

7.2 Limitations and Future work

In this section, we discuss the limitations of our studies and potential directions for future research.

- The area of MLOps is now in its nascent stage and warrants further consideration from the academic community. The current emphasis within the community predominantly revolves around the development of (ML systems, with less emphasis on the deployment phase. We conduct an empirical study to provide practitioners with a comprehensive understanding of the challenges faced during the deployment of DRL systems. However, our empirical study relies only on Stack Overflow as a single data source for studying developer challenges which may introduce bias and limit the representation of developers’ issues. For future work, we may broaden our data-gathering scope and collect

developer challenges from GitHub, Reddit, and PyTorch forums. Another prospective research area is to conduct a three-staged empirical study that combines a mixed-methods approach with a sequential exploratory technique [209]. During the initial phase of the empirical study, a Systematic Literature Review (SLR) is conducted to identify the challenges encountered during the deployment of DRL systems, as well as the solutions to address these challenges. The second phase of the study encompasses semi-structured interviews, which are conducted to enhance and partially verify the data obtained during the initial phase. During the third phase of the study, a survey is conducted to collect quantitative data, in order to enhance and extend the conclusions drawn from the initial two phases. The utilization of data triangulation, which involves the integration of information from several sources, has been widely recognized in academic literature as a means to enhance the reliability and trustworthiness of research findings [209]. To summarize, our study, while instructive, is only a first step towards a more holistic understanding of challenges in deploying DRL systems in real-world circumstances.

- Post-training compression strategies can result in divergent behaviors between the original model and its compressed counterpart, resulting in issues during deployment. Ensuring the reliability of these post-compression strategies thus becomes paramount. DiverGet is designed to be extendable and may be used to evaluate any post-training compression approach like pruning and clustering; however, the current versions primarily support DNN quantization. In future iterations, we plan to expand the area of evaluation and demonstrate the relevance and efficacy of our approach across a wide range of prevalent compression techniques. DNN quantization may also be applied to DRL systems. Quantization poses a greater challenge in the context of DRL and may hinder the policy’s long-term decision-making. This is due to the significant impact of the agent’s present action on its subsequent states and actions [116]. A potential avenue for future study is examining the effects of quantization on DRL systems. In relation to the internal structure of DiverGet, our future enhancements involve two key aspects. Firstly, we aim to enhance the generation of transformation vectors in DiverGet by employing a generator based on either a Generative Adversarial Network (GAN) or a Variational Autoencoder (VAE). Secondly, we intend to incorporate naturalness metrics [210, 211] to validate the realism of DiverGet’s generated data.
- Environment drift can cause a DRL agent’s effectiveness to decline over time, drastically increasing operational costs. Recent studies [27] have demonstrated that environmental drifts can push Continuous Learning (CL) into long, or even failed, healing cycles,

necessitating the development of more efficient and effective self-healing techniques. In our study, we proposed Dr. DRL a new DRL self-healing approach leveraging an intentional forgetting mechanism. In Dr. DRL’s design, the TKNC activation-based ranking approach is leveraged to classify neurons as either hyperactive or hypoactive. Nevertheless, the process of categorizing the DRL system into binary categories may lead to an oversimplification of its behavior, resulting in a potential loss of valuable information. In our future studies, we want to investigate other approaches for categorizing neurons that extend beyond the conventional activation-based ranking. These alternative methods encompass the utilization of singular value decomposition (SVD) [212], activation graph analysis [213], and unsupervised learning techniques [214]. Through this approach, our aim is to enhance the precision in identifying the most prominent neurons, namely those that exhibit hyperactivity. In addition, it should be noted that Dr. DRL does not support multiagent DRL systems. These systems have gained significant popularity and are currently being extensively employed across many sectors. Therefore, it is imperative to cover multi-agent DRL systems in Dr. DRL’s future iterations.

REFERENCES

- [1] S. K. Roy, G. Krishna, S. R. Dubey, and B. B. Chaudhuri, “HybridSN: Exploring 3D-2D CNN Feature Hierarchy for Hyperspectral Image Classification,” *IEEE Geoscience and Remote Sensing Letters*, vol. 17, no. 2, pp. 277–281, Feb. 2020, arXiv: 1902.06701. [Online]. Available: <http://arxiv.org/abs/1902.06701>
- [2] Z. Zhong, J. Li, Z. Luo, and M. Chapman, “Spectral–Spatial Residual Network for Hyperspectral Image Classification: A 3-D Deep Learning Framework,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 56, no. 2, pp. 847–858, Feb. 2018.
- [3] A. W. Moore, “Efficient memory-based learning for robot control,” 1990.
- [4] “The electric light system,” 2015, publisher: National Historical Park New Jersey. [Online]. Available: <https://www.nps.gov/edis/learn/kidsyouth/the-electric-light-system-phonograph-motion-pictures.htm>
- [5] “Andrew ng: why ai is the new electricity,” Feb. 2023. [Online]. Available: <https://www.gsb.stanford.edu/insights/andrew-ng-why-ai-new-electricity>
- [6] “Henry kissinger warns that ai will fundamentally alter human consciousness,” Nov. 2019. [Online]. Available: <https://gizmodo.com/henry-kissinger-warns-that-ai-will-fundamentally-alter-1839642809>
- [7] D. Zhang, S. Mishra, E. Brynjolfsson, J. Etchemendy, D. Ganguli, B. Grosz, T. Lyons, J. Manyika, J. C. Niebles, M. Sellitto *et al.*, “The ai index 2021 annual report,” *arXiv preprint arXiv:2103.06312*, 2021.
- [8] J. Bajorath, S. Kearnes, W. P. Walters, N. A. Meanwell, G. I. Georg, and S. Wang, “Artificial intelligence in drug discovery: into the great wide open,” pp. 8651–8652, 2020.
- [9] J. Marchant, “Powerful antibiotics discovered using ai.” *Nature*, 2020.
- [10] “The data that transformed AI research—and possibly the world,” Jul. 2017. [Online]. Available: <https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world>

- [11] C. Gitau, “Success stories of reinforcement learning,” Nov. 2018. [Online]. Available: <https://categorgitau.medium.com/success-stories-of-reinforcement-learning-9b4064171668>
- [12] M. C. Horowitz, G. C. Allen, E. B. Kania, and P. Scharre, *Strategic competition in an era of artificial intelligence*. Center for a New American Security., 2018.
- [13] A. H. Yahmed, H. B. Braiek, F. Khomh, S. Bouzidi, and R. Zaatour, “Diverget: a search-based software testing approach for deep neural network quantization assessment,” *Empirical Software Engineering*, vol. 27, no. 7, p. 193, 2022.
- [14] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, pp. 436–444, 2015.
- [15] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [16] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [17] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent trends in deep learning based natural language processing,” *ieee Computational intelligence magazine*, vol. 13, no. 3, pp. 55–75, 2018.
- [18] H. Geoffrey, D. Li, Y. Dong, E. D. George, and A.-r. Mohamed, “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” *IEEE Signal processing magazine*, vol. 29, no. 6, pp. 82–97, 2012.
- [19] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, “Deepgauge: Multi-granularity testing criteria for deep learning systems,” in *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering*, 2018, pp. 120–131.
- [20] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.
- [21] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

- [22] K. Arulkumaran, A. Cully, and J. Togelius, “Alphastar: An evolutionary computation perspective,” in *Proceedings of the genetic and evolutionary computation conference companion*, 2019, pp. 314–315.
- [23] F. Amat, A. Chandrashekar, T. Jebara, and J. Basilico, “Artwork personalization at netflix,” in *Proceedings of the 12th ACM conference on recommender systems*, 2018, pp. 487–488.
- [24] J. Gauci, E. Conti, Y. Liang, K. Virochsiri, Y. He, Z. Kaden, V. Narayanan, X. Ye, Z. Chen, and S. Fujimoto, “Horizon: Facebook’s open source applied reinforcement learning platform,” *arXiv preprint arXiv:1811.00260*, 2018.
- [25] A. H. Yahmed, A. A. Abbassi, A. Nikanjam, H. Li, and F. Khomh, “Deploying deep reinforcement learning systems: A taxonomy of challenges,” *arXiv preprint arXiv:2308.12438*, 2023.
- [26] M. Panzer and B. Bender, “Deep reinforcement learning in production systems: a systematic literature review,” *International Journal of Production Research*, vol. 60, no. 13, pp. 4316–4341, 2022.
- [27] M. Biagiola and P. Tonella, “Testing the plasticity of reinforcement learning based systems,” *ACM Transactions on Software Engineering and Methodology*, 2022.
- [28] A. H. Yahmed, R. Bouchoucha, H. B. Braiek, and F. Khomh, “An intentional forgetting-driven self-healing method for deep reinforcement learning systems,” *arXiv preprint arXiv:2308.12445*, 2023.
- [29] “An intentional forgetting-driven self-healing method for deep reinforcement learning systems,” 2023. [Online]. Available: <https://github.com/ahmedhajyahmed/DrDRL>
- [30] “Replication package,” <https://github.com/DRLDeploymentChallenges-ICSME2023/ReplicationPackage>.
- [31] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, “Deephunter: a coverage-guided fuzz testing framework for deep neural networks,” in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019, pp. 146–157.
- [32] H. Nguyen and H. La, “Review of deep reinforcement learning for robot manipulation,” in *2019 Third IEEE International Conference on Robotic Computing (IRC)*, 2019, pp. 590–595.

- [33] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, “A survey of deep reinforcement learning in video games,” *CoRR*, vol. abs/1912.10944, 2019. [Online]. Available: <http://arxiv.org/abs/1912.10944>
- [34] D. W. Otter, J. R. Medina, and J. K. Kalita, “A survey of the usages of deep learning for natural language processing,” *IEEE transactions on neural networks and learning systems*, vol. 32, no. 2, pp. 604–624, 2020.
- [35] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis *et al.*, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.
- [36] H. Purwins, B. Li, T. Virtanen, J. Schlüter, S.-Y. Chang, and T. Sainath, “Deep learning for audio signal processing,” *IEEE Journal of Selected Topics in Signal Processing*, vol. 13, no. 2, pp. 206–219, 2019.
- [37] Q. Le, L. Miralles-Pechuán, S. Kulkarni, J. Su, and O. Boydell, “An overview of deep learning in industry,” *Data Analytics and AI*, pp. 65–98, 2020.
- [38] R. S. Sutton and A. G. Barto, “Reinforcement learning: an introduction mit press,” *Cambridge, MA*, vol. 22447, 1998.
- [39] R. Bellman, “On the theory of dynamic programming,” *Proceedings of the national Academy of Sciences*, vol. 38, no. 8, pp. 716–719, 1952.
- [40] P. Kurrek, F. Zoghلامي, M. Jocas, M. Stoelen, and V. Salehi, “Reinforcement learning lifecycle for the design of advanced robotic systems,” in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1. IEEE, 2020, pp. 230–235.
- [41] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen *et al.*, “Massively parallel methods for deep reinforcement learning,” *arXiv preprint arXiv:1507.04296*, 2015.
- [42] B. Liu, L. Wang, and M. Liu, “Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems,” *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4555–4562, 2019.
- [43] H. Mao, M. Schwarzkopf, H. He, and M. Alizadeh, “Towards safe online reinforcement learning in computer systems,” in *NeurIPS Machine Learning for Systems Workshop*, 2019.

- [44] B. Balaji and A. Luo, “Deploying reinforcement learning in production using ray and amazon sagemaker, aws machine learning blog,” Nov. 2020.
- [45] C. Olston, N. Fiedel, K. Gorovoy, J. Harmsen, L. Lao, F. Li, V. Rajashekhar, S. Ramesh, and J. Soyke, “Tensorflow-serving: Flexible, high-performance ml serving,” *arXiv preprint arXiv:1712.06139*, 2017.
- [46] E. Liberty, Z. Karnin, B. Xiang, L. Rouesnel, B. Coskun, R. Nallapati, J. Delgado, A. Sadoughi, Y. Astashonok, P. Das *et al.*, “Elastic machine learning algorithms in amazon sagemaker,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020, pp. 731–737.
- [47] A. Shiri, U. Kallakuri, H.-A. Rashid, B. Prakash, N. R. Waytowich, T. Oates, and T. Mohsenin, “E2hrl: An energy-efficient hardware accelerator for hierarchical deep reinforcement learning,” *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 27, no. 5, pp. 1–19, 2022.
- [48] J. Jagannath, K. Hamedani, C. Farquhar, K. Ramezanpour, and A. Jagannath, “Mr-inet gym: Framework for edge deployment of deep reinforcement learning on embedded software defined radio,” in *Proceedings of the 2022 ACM Workshop on Wireless Security and Machine Learning*, 2022, pp. 51–56.
- [49] R. David, J. Duke, A. Jain, V. Janapa Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang *et al.*, “Tensorflow lite micro: Embedded machine learning for tinyml systems,” *Proceedings of Machine Learning and Systems*, vol. 3, pp. 800–811, 2021.
- [50] “Apple developer documentation.” [Online]. Available: <https://developer.apple.com/documentation/coreml>
- [51] A. Lall and S. Tallur, “Deep reinforcement learning-based pairwise dna sequence alignment method compatible with embedded edge devices,” *Scientific Reports*, vol. 13, no. 1, p. 2773, 2023.
- [52] C. Bamford, M. Jiang, M. Samvelyan, and T. Rocktäschel, “Griddlyjs: A web ide for reinforcement learning,” *arXiv preprint arXiv:2207.06105*, 2022.
- [53] Y. Ma, D. Xiang, S. Zheng, D. Tian, and X. Liu, “Moving deep learning into web browser: How far can we go?” in *The World Wide Web Conference*, 2019, pp. 1234–1244.

- [54] A. Tucker, A. Gleave, and S. Russell, “Inverse reinforcement learning for video games,” *arXiv preprint arXiv:1810.10593*, 2018.
- [55] U. Technologies, “Barracuda release,” <https://github.com/Unity-Technologies/barracuda-release>, 2021.
- [56] Y. Guo, “A Survey on Methods and Theories of Quantized Neural Networks,” *arXiv:1808.04752 [cs, stat]*, Dec. 2018, arXiv: 1808.04752. [Online]. Available: <http://arxiv.org/abs/1808.04752>
- [57] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” *arXiv:1806.08342 [cs, stat]*, Jun. 2018, arXiv: 1806.08342. [Online]. Available: <http://arxiv.org/abs/1806.08342>
- [58] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A survey of quantization methods for efficient neural network inference,” *arXiv preprint arXiv:2103.13630*, 2021.
- [59] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, “Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation,” *arXiv:2004.09602 [cs, stat]*, Apr. 2020, arXiv: 2004.09602. [Online]. Available: <http://arxiv.org/abs/2004.09602>
- [60] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of real-world reinforcement learning,” *arXiv preprint arXiv:1904.12901*, 2019.
- [61] Z. Wang, C. Chen, H.-X. Li, D. Dong, and T.-J. Tarn, “A novel incremental learning scheme for reinforcement learning in dynamic environments,” in *2016 12th World Congress on Intelligent Control and Automation (WCICA)*. IEEE, 2016, pp. 2426–2431.
- [62] —, “Incremental reinforcement learning with prioritized sweeping for dynamic environments,” *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 2, pp. 621–632, 2019.
- [63] M. B. Ring *et al.*, “Continual learning in reinforcement environments,” 1994.
- [64] R. Hadsell, D. Rao, A. A. Rusu, and R. Pascanu, “Embracing change: Continual learning in deep neural networks,” *Trends in cognitive sciences*, vol. 24, no. 12, pp. 1028–1040, 2020.

- [65] R. M. French, “Catastrophic forgetting in connectionist networks,” *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.
- [66] L. Gazzola, D. Micucci, and L. Mariani, “Automatic software repair: A survey,” in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 1219–1219.
- [67] M. Monperrus, “Automatic software repair: a bibliography,” *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–24, 2018.
- [68] H. Psaiar and S. Dustdar, “A survey on self-healing systems: approaches and systems,” *Computing*, vol. 91, pp. 43–73, 2011.
- [69] N. Perino, “A framework for self-healing software systems,” in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 1397–1400.
- [70] D. Ghosh, R. Sharman, H. R. Rao, and S. Upadhyaya, “Self-healing systems—survey and synthesis,” *Decision support systems*, vol. 42, no. 4, pp. 2164–2185, 2007.
- [71] C. Beierle and I. J. Timm, “Intentional forgetting: An emerging field in ai and beyond,” pp. 5–8, 2019.
- [72] D. Shands and C. Talcott, “Intentional forgetting,” *arXiv preprint arXiv:2106.09802*, 2021.
- [73] L. Reuter, J. O. Berndt, A.-S. Ulfert, C. H. Antoni, T. Ellwart, and I. J. Timm, “Intentional forgetting in distributed artificial intelligence,” *KI-Künstliche Intelligenz*, vol. 33, pp. 69–77, 2019.
- [74] I. J. Timm, S. Staab, M. Siebers, C. Schon, U. Schmid, K. Sauerwald, L. Reuter, M. Ragni, C. Niederée, H. Maus *et al.*, “Intentional forgetting in artificial intelligence systems: Perspectives and challenges,” in *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer, 2018, pp. 357–365.
- [75] I. J. Timm, O. Herzog, J. O. Berndt, and C. Beierle, “Intentional forgetting must be part of the functionality: Interview with prof. otthein herzog, jacobs university bremen, university of bremen, and tongji university, shanghai,” *KI-Künstliche Intelligenz*, vol. 33, pp. 89–91, 2019.
- [76] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, “Machine learning testing: Survey, landscapes and horizons,” *IEEE Transactions on Software Engineering*, 2020.

- [77] H. B. Braiek and F. Khomh, “On testing machine learning programs,” *Journal of Systems and Software*, vol. 164, p. 110542, 2020.
- [78] S. Galhotra, Y. Brun, and A. Meliou, “Fairness testing: testing software for discrimination,” in *Proceedings of the 2017 11th Joint meeting on foundations of software engineering*, 2017, pp. 498–510.
- [79] E. J. Weyuker, “On testing non-testable programs,” *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982.
- [80] N. Li and J. Offutt, “Test oracle strategies for model-based testing,” *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 372–395, 2016.
- [81] J. Mayer, R. Guderlei *et al.*, “Test oracles using statistical methods.” in *SOQUA/TECOS*, 2004, pp. 179–189.
- [82] L. Padgham, Z. Zhang, J. Thangarajah, and T. Miller, “Model-based test oracle generation for automated unit testing of agent systems,” *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1230–1244, 2013.
- [83] M. D. Davis and E. J. Weyuker, “Pseudo-oracles for non-testable programs,” in *Proceedings of the ACM’81 Conference*, 1981, pp. 254–257.
- [84] T. Y. Chen, S. C. Cheung, and S. M. Yiu, “Metamorphic testing: a new approach for generating next test cases,” *arXiv preprint arXiv:2002.12543*, 2020.
- [85] P. McMinn, “Search-Based Software Testing: Past, Present and Future,” in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, Mar. 2011, pp. 153–163.
- [86] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *MHS’95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Oct. 1995, pp. 39–43.
- [87] M. Mitchell, *An introduction to genetic algorithms*, 7th ed., ser. Complex adaptive systems, Cambridge, Mass., 2001.
- [88] K. Pei, Y. Cao, J. Yang, and S. Jana, “DeepXplore: Automated Whitebox Testing of Deep Learning Systems,” *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, Oct. 2017, arXiv: 1705.06640. [Online]. Available: <http://arxiv.org/abs/1705.06640>

- [89] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, “DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems,” *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 120–131, Sep. 2018, arXiv: 1803.07519. [Online]. Available: <http://arxiv.org/abs/1803.07519>
- [90] H. B. Braiek and F. Khomh, “DeepEvolution: A Search-Based Testing Approach for Deep Neural Networks,” *arXiv:1909.02563 [cs, stat]*, Sep. 2019, arXiv: 1909.02563. [Online]. Available: <http://arxiv.org/abs/1909.02563>
- [91] X. Xie, L. Ma, H. Wang, Y. Li, Y. Liu, and X. Li, “Diffchaser: Detecting disagreements for deep neural networks.” in *IJCAI*, 2019, pp. 5772–5778.
- [92] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, “DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing,” *arXiv:1809.01266 [cs]*, Nov. 2018, arXiv: 1809.01266. [Online]. Available: <http://arxiv.org/abs/1809.01266>
- [93] A. Odena and I. Goodfellow, “TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing,” *arXiv:1807.10875 [cs, stat]*, Jul. 2018, arXiv: 1807.10875. [Online]. Available: <http://arxiv.org/abs/1807.10875>
- [94] Y. Tian, K. Pei, S. Jana, and B. Ray, “DeepTest: Automated Testing of Deep-Neural-Network-driven Autonomous Cars,” *arXiv:1708.08559 [cs]*, Mar. 2018, arXiv: 1708.08559. [Online]. Available: <http://arxiv.org/abs/1708.08559>
- [95] B. Biggio and F. Roli, “Wild patterns: Ten years after the rise of adversarial machine learning half-day tutorial,” in *25th ACM Conference on Computer and Communications Security, CCS 2018*. Association for Computing Machinery, 2018, pp. 2154–2156.
- [96] E. Breck, S. Cai, E. Nielsen, M. Salib, and D. Sculley, “The ml test score: A rubric for ml production readiness and technical debt reduction,” in *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 2017, pp. 1123–1132.
- [97] A. Cummaudo, R. Vasa, S. Barnett, J. Grundy, and M. Abdelrazek, “Interpreting cloud computer vision pain-points: A mining study of stack overflow,” in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, 2020, pp. 1584–1596.
- [98] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, “A first look at deep learning apps on smartphones,” in *The World Wide Web Conference*, 2019, pp. 2125–2136.

- [99] Q. Guo, S. Chen, X. Xie, L. Ma, Q. Hu, H. Liu, Y. Liu, J. Zhao, and X. Li, “An empirical study towards characterizing deep learning development and deployment across different frameworks and platforms,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 810–822.
- [100] M. Openja, A. Nikanjam, A. H. Yahmed, F. Khomh, and Z. M. J. Jiang, “An empirical study of challenges in converting deep learning models,” in *2022 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2022, pp. 13–23.
- [101] Z. Chen, Y. Cao, Y. Liu, H. Wang, T. Xie, and X. Liu, “A comprehensive study on challenges in deploying deep learning based software,” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 750–762.
- [102] A. Nikanjam, M. M. Morovati, F. Khomh, and H. Ben Braiek, “Faults in deep reinforcement learning programs: a taxonomy and a detection approach,” *Automated Software Engineering*, vol. 29, no. 1, pp. 1–32, 2022.
- [103] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, “Progressive neural networks,” *arXiv preprint arXiv:1606.04671*, 2016.
- [104] H. Shin, J. K. Lee, J. Kim, and J. Kim, “Continual learning with deep generative replay,” *Advances in neural information processing systems*, vol. 30, 2017.
- [105] Z. Li and D. Hoiem, “Learning without forgetting,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.
- [106] D. Rolnick, A. Ahuja, J. Schwarz, T. Lillicrap, and G. Wayne, “Experience replay for continual learning,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.
- [107] H. Zhang and W. Chan, “Apricot: A weight-adaptation approach to fixing deep learning models,” in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2019, pp. 376–387.
- [108] J. Sohn, S. Kang, and S. Yoo, “Search based repair of deep neural networks,” *arXiv preprint arXiv:1912.12463*, 2019.
- [109] A. Stocco and P. Tonella, “Confidence-driven weighted retraining for predicting safety-critical failures in autonomous driving systems,” *Journal of Software: Evolution and Process*, vol. 34, no. 10, p. e2386, 2022.

- [110] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, “Pac model-free reinforcement learning,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 881–888.
- [111] “Stable baselines3,” Dec. 2022, original-date: 2020-05-05T05:52:26Z. [Online]. Available: <https://github.com/DLR-RM/stable-baselines3>
- [112] M. Plappert, “keras-rl,” <https://github.com/keras-rl/keras-rl>, 2016.
- [113] A. Kuhnle, M. Schaarschmidt, and K. Fricke, “Tensorforce: a tensorflow library for applied reinforcement learning,” Web page, 2017. [Online]. Available: <https://github.com/tensorforce/tensorforce>
- [114] Z. Cai, X. He, J. Sun, and N. Vasconcelos, “Deep learning with low precision by half-wave gaussian quantization,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 5918–5926.
- [115] Y. Guo, “A survey on methods and theories of quantized neural networks,” *arXiv preprint arXiv:1808.04752*, 2018.
- [116] S. Krishnan, M. Lam, S. Chitlangia, Z. Wan, G. Barth-Maron, A. Faust, and V. Janapa Reddi, “Quarl: Quantization for sustainable reinforcement learning,” *arXiv e-prints*, pp. arXiv–1910, 2019.
- [117] R. Abdalkareem, E. Shihab, and J. Rilling, “What do developers use the crowd for? a study using stack overflow,” *IEEE Software*, vol. 34, no. 2, pp. 53–60, 2017.
- [118] “Stack exchange data dump,” <https://archive.org/details/stackexchange>.
- [119] A. Abdellatif, D. Costa, K. Badran, R. Abdalkareem, and E. Shihab, “Challenges in chatbot development: A study of stack overflow posts,” in *Proceedings of the 17th international conference on mining software repositories*, 2020, pp. 174–185.
- [120] M. Alshangiti, H. Sapkota, P. K. Murukannaiah, X. Liu, and Q. Yu, “Why is developing machine learning applications challenging? a study on stack overflow posts,” in *2019 acm/ieee international symposium on empirical software engineering and measurement (esem)*. IEEE, 2019, pp. 1–11.
- [121] S. Ahmed and M. Bagherzadeh, “What do concurrency developers ask about? a large-scale study using stack overflow,” in *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, 2018, pp. 1–10.

- [122] GitHub, “Github,” 2008. [Online]. Available: <https://github.com/>
- [123] Google, “Google scholar,” 2004. [Online]. Available: <https://scholar.google.com/>
- [124] Google, “Google forms,” 2008. [Online]. Available: <https://www.google.com/forms/about/>
- [125] A. Joshi, S. Kale, S. Chandel, and D. K. Pal, “Likert scale: Explored and explained,” *British journal of applied science & technology*, vol. 7, no. 4, p. 396, 2015.
- [126] Google, “Tensorflow lite,” 2019. [Online]. Available: <https://www.tensorflow.org/lite>
- [127] Apple Inc., “Core ml,” 2017. [Online]. Available: <https://developer.apple.com/documentation/coreml>
- [128] Google, “Tensorflow.js: Machine learning for the web and beyond,” 2018. [Online]. Available: <https://www.tensorflow.org/js>
- [129] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [130] Rex.Gatling, “How do I deploy deep reinforcement learning neural network I coded in Pytorch to my website?” Jul. 2019. [Online]. Available: <https://stackoverflow.com/q/57206464>
- [131] ak, “Reinforcement learning - custom environment implementation in java for python rl framework,” Sep. 2021. [Online]. Available: <https://stackoverflow.com/q/69035664>
- [132] placeybordeaux, “What is the easiest 2D game library to use with Scala?” Dec. 2012. [Online]. Available: <https://stackoverflow.com/q/10888207>
- [133] escribano101, “ML-agents cooperative push block not returning rewards,” Oct. 2021. [Online]. Available: <https://stackoverflow.com/q/69432159>
- [134] nokdoo, “Unity ml agent, camerasensor - compression type,” Aug. 2022. [Online]. Available: <https://stackoverflow.com/q/73481946>
- [135] Billwaa, “Running ml-agent trained frozen graph in tensorflow or keras,” Jul. 2020. [Online]. Available: <https://stackoverflow.com/q/62839784>
- [136] A. G. Prasad, “ImportError: cannot import name 'cluster_resolver’,” Jul. 2019. [Online]. Available: <https://stackoverflow.com/q/56911525>

- [137] A. Chakraborti, “Pyglet no standard config is available, when trying to run openai gym inside a docker container,” Mar. 2018. [Online]. Available: <https://stackoverflow.com/q/49121200>
- [138] dhanush-ai1990, “Cannot install atari-py on RedHat Enterprise server with c compiler broken error,” Aug. 2018. [Online]. Available: <https://stackoverflow.com/q/52119002>
- [139] P. Borges, “Convert stable-baselines tensorflow model to tensorflowjs,” Sep. 2019. [Online]. Available: <https://stackoverflow.com/q/57934683>
- [140] 16Aghnar, “Reinforcement Learning coach : Saver fails to restore agent’s checkpoint,” Oct. 2020. [Online]. Available: <https://stackoverflow.com/q/64349126>
- [141] H. Lemon, “Can’t use trained model on older version of ML-agents from Unity,” Jun. 2021. [Online]. Available: <https://stackoverflow.com/q/68053354>
- [142] A. Jones, “Using integrated graphics for headless rendering inside a cloud docker container,” May 2019. [Online]. Available: <https://stackoverflow.com/q/56341582>
- [143] G. B, “Google Coral performance isn’t amazing,” Dec. 2021. [Online]. Available: <https://stackoverflow.com/q/70417439>
- [144] Google, “Coral,” <https://coral.ai/>, 2019.
- [145] J. Szatmari, “Is it possible to play against a trained bot from the keyboard using gym (Pong)?” Nov. 2019. [Online]. Available: <https://stackoverflow.com/q/59040990>
- [146] FindersKeeper, “Type error while tracing for exporting pytorch RL agents to ONNX,” Mar. 2022. [Online]. Available: <https://stackoverflow.com/q/71459368>
- [147] Welius, “How to use a tensorflow lite model in unity,” Nov. 2019. [Online]. Available: <https://stackoverflow.com/q/56711706>
- [148] YoavEtzioni, “Is it possible to train with tensorflow 1 using float16?” Feb. 2020. [Online]. Available: <https://stackoverflow.com/q/60171887>
- [149] J. Hager, “Aws lambda function with layer works perfectly in testing on aws, but doesn’t return body to front end fetch,” Sep. 2022. [Online]. Available: <https://stackoverflow.com/q/73603895>
- [150] AwokeKnowing, “How can I properly run OpenAI gym with nvidia-docker and see the environments,” Sep. 2017. [Online]. Available: <https://stackoverflow.com/q/45475658>

- [151] H. Gupta, “Headless servers Opengym AI rendering Error while using ray,” Mar. 2021. [Online]. Available: <https://stackoverflow.com/q/66772492>
- [152] W. Zhao, J. P. Queralta, and T. Westerlund, “Sim-to-real transfer in deep reinforcement learning for robotics: a survey,” in *2020 IEEE symposium series on computational intelligence (SSCI)*. IEEE, 2020, pp. 737–744.
- [153] J. H. Zar, “Significance testing of the spearman rank correlation coefficient,” *Journal of the American Statistical Association*, vol. 67, no. 339, pp. 578–580, 1972.
- [154] F. Feit, A. Metzger, and K. Pohl, “Explaining online reinforcement learning decisions of self-adaptive systems,” in *2022 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2022, pp. 51–60.
- [155] J. Garcia and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.
- [156] S. Padakandla, P. KJ, and S. Bhatnagar, “Reinforcement learning algorithm for non-stationary environments,” *Applied Intelligence*, vol. 50, pp. 3590–3606, 2020.
- [157] Databricks, “Mlflow: Open source platform for the complete machine learning lifecycle,” 2018. [Online]. Available: <https://mlflow.org/>
- [158] Z. Inc., “Rl-bakery: build production, large scale, deep reinforcement learning applications,” <https://github.com/zynga/rl-bakery>, 2021.
- [159] D. Inc., “Productionizing deep reinforcement learning with spark and mlflow,” <https://www.conferencecast.tv/talk-31961-productionizing-deep-reinforcement-learning-with-spark-and-mlflow>, 2020.
- [160] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255, iSSN: 1063-6919.
- [161] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *arXiv:1512.03385 [cs]*, Dec. 2015, arXiv: 1512.03385. [Online]. Available: <http://arxiv.org/abs/1512.03385>
- [162] T. Young, D. Hazarika, S. Poria, and E. Cambria, “Recent Trends in Deep Learning Based Natural Language Processing,” *arXiv:1708.02709 [cs]*, Nov. 2018, arXiv: 1708.02709. [Online]. Available: <http://arxiv.org/abs/1708.02709>

- [163] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups,” *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
- [164] Q. Hu, Y. Guo, M. Cordy, X. Xie, W. Ma, M. Papadakis, and Y. L. Traon, “Characterizing and understanding the behavior of quantized models for reliable deployment,” *arXiv preprint arXiv:2204.04220*, 2022.
- [165] R. Zaatour, S. Bouzidi, and E. Zagrouba, “Unsupervised image-adapted local fisher discriminant analysis to reduce hyperspectral images without ground truth,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 58, no. 11, pp. 7931–7941, 2020.
- [166] S. Bouzidi, “Parallel and distributed implementation on spark of a spectral-spatial classifier for hyperspectral images,” *Journal of Applied Remote Sensing*, vol. 13, no. 3, p. 034501, 2019.
- [167] H. Agili, S. Daniel, and K. Chokmani, “Revue des méthodes de prétraitement des données d’imagerie hyperspectrale acquises depuis un drone.” *Geomatica*, vol. 68, no. 4, pp. 331–343, 2014. [Online]. Available: <http://espace.inrs.ca/id/eprint/4391/>
- [168] Y. Q. Shi and H. Sun, *Image and video compression for multimedia engineering: Fundamentals, algorithms, and standards*. CRC press, 2017.
- [169] G. Yang, N. Zheng, and S. Guo, “Optimal wavelet filter design for remote sensing image compression,” *Journal of Electronics (China)*, vol. 24, no. 2, pp. 276–284, 2007.
- [170] Y. Dua, V. Kumar, and R. S. Singh, “Comprehensive review of hyperspectral image compression algorithms,” *Optical Engineering*, vol. 59, no. 9, p. 090902, 2020.
- [171] N. Thomos, N. V. Boulgouris, and M. G. Strintzis, “Optimized transmission of jpeg2000 streams over wireless channels,” *IEEE Transactions on image processing*, vol. 15, no. 1, pp. 54–67, 2005.
- [172] S. Kullback, “Letter to the editor: The kullback-leibler distance,” 1987.
- [173] T. M. Cover and J. A. Thomas, *Elements of information theory*, ser. Wiley series in telecommunications. New York: Wiley, 1991.
- [174] J. Yu, Y. Fu, Y. Zheng, Z. Wang, and X. Ye, “Test4deep: an effective white-box testing for deep neural networks,” in *2019 IEEE International Conference on Computational*

Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC). IEEE, 2019, pp. 16–23.

- [175] P. Jaccard, “The Distribution of the Flora in the Alpine Zone.1,” *New Phytologist*, vol. 11, no. 2, pp. 37–50, 1912. [Online]. Available: <https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x>
- [176] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, “Improved techniques for training gans,” *Advances in neural information processing systems*, vol. 29, 2016.
- [177] M. Alzantot, Y. Sharma, S. Chakraborty, H. Zhang, C.-J. Hsieh, and M. Srivastava, “GenAttack: Practical Black-box Attacks with Gradient-Free Optimization,” *arXiv:1805.11090 [cs]*, Jun. 2019, arXiv: 1805.11090. [Online]. Available: <http://arxiv.org/abs/1805.11090>
- [178] Y.-C. Ho and D. L. Pepyne, “Simple explanation of the no-free-lunch theorem and its implications,” *Journal of optimization theory and applications*, vol. 115, no. 3, pp. 549–570, 2002.
- [179] R. Mosli, M. Wright, B. Yuan, and Y. Pan, “They Might NOT Be Giants: Crafting Black-Box Adversarial Examples with Fewer Queries Using Particle Swarm Optimization,” *arXiv:1909.07490 [cs, stat]*, Sep. 2019, arXiv: 1909.07490. [Online]. Available: <http://arxiv.org/abs/1909.07490>
- [180] S. K. Joshi and J. C. Bansal, “Parameter tuning for meta-heuristics,” *Knowledge-Based Systems*, vol. 189, p. 105094, 2020.
- [181] F. Wilcoxon, “Individual comparisons by ranking methods,” *Biometrics Bulletin*, vol. 1, no. 6, p. 80, Dec. 1945. [Online]. Available: <https://www.jstor.org/stable/10.2307/3001968?origin=crossref>
- [182] A. Vargha and H. D. Delaney, “A critique and improvement of the cl common language effect size statistics of mcgraw and wong,” *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.
- [183] M. R. Hess and J. D. Kromrey, “Robust confidence intervals for effect sizes: A comparative study of cohen’sd and cliff’s delta under non-normality and heterogeneous variances,” in *annual meeting of the American Educational Research Association*, vol. 1. Citeseer, 2004.

- [184] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: A system for large-scale machine learning,” *arXiv:1605.08695 [cs]*, May 2016, arXiv: 1605.08695. [Online]. Available: <http://arxiv.org/abs/1605.08695>
- [185] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.” *Journal of machine learning research*, vol. 9, no. 11, 2008.
- [186] Y. LeCun, “The mnist database of handwritten digits,” <http://yann.lecun.com/exdb/mnist/>, 1998.
- [187] A. Krizhevsky, V. Nair, and G. Hinton, “The cifar-10 dataset,” <http://www.cs.toronto.edu/kriz/cifar.html>, 2014.
- [188] G. I. Webb, R. Hyde, H. Cao, H. L. Nguyen, and F. Petitjean, “Characterizing concept drift,” *Data Mining and Knowledge Discovery*, vol. 30, no. 4, pp. 964–994, 2016.
- [189] P. Li, J. Thomas, X. Wang, A. Khalil, A. Ahmad, R. Inacio, S. Kapoor, A. Parekh, A. Doufexi, A. Shojaeifard *et al.*, “Rlops: Development life-cycle of reinforcement learning aided open ran,” *IEEE Access*, vol. 10, pp. 113 808–113 826, 2022.
- [190] C. Lyle, M. Rowland, and W. Dabney, “Understanding and preventing capacity loss in reinforcement learning,” *arXiv preprint arXiv:2204.09560*, 2022.
- [191] B. Wexler, E. Sarafian, and S. Kraus, “Analyzing and overcoming degradation in warm-start reinforcement learning,” in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4048–4055.
- [192] E. Nikishin, M. Schwarzer, P. D’Oro, P.-L. Bacon, and A. Courville, “The primacy bias in deep reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2022, pp. 16 828–16 847.
- [193] S. I. Mirzadeh, A. Chaudhry, D. Yin, T. Nguyen, R. Pascanu, D. Gorur, and M. Farajtabar, “Architecture matters in continual learning,” *arXiv preprint arXiv:2202.00275*, 2022.
- [194] K. Khetarpal, M. Riemer, I. Rish, and D. Precup, “Towards continual reinforcement learning: A review and perspectives,” *Journal of Artificial Intelligence Research*, vol. 75, pp. 1401–1476, 2022.

- [195] C. Li, X. Wei, Y. Zhao, and X. Geng, “An effective maximum entropy exploration approach for deceptive game in reinforcement learning,” *Neurocomputing*, vol. 403, pp. 98–108, 2020.
- [196] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” *arXiv preprint arXiv:1803.03635*, 2018.
- [197] E. Malach, G. Yehudai, S. Shalev-Schwartz, and O. Shamir, “Proving the lottery ticket hypothesis: Pruning is all you need,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 6682–6691.
- [198] A. E. Siswanto, “Block sparsity and weight initialization in neural network pruning,” Ph.D. dissertation, Massachusetts Institute of Technology, 2021.
- [199] C. M. Bishop, “Neural networks and their applications,” *Review of scientific instruments*, vol. 65, no. 6, pp. 1803–1832, 1994.
- [200] A. G. Barto, R. S. Sutton, and C. W. Anderson, “Neuronlike adaptive elements that can solve difficult learning control problems,” *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.
- [201] A. Geramifard, C. Dann, R. H. Klein, W. Dabney, and J. P. How, “Rlpy: a value-function-based reinforcement learning framework for education and research.” *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 1573–1578, 2015.
- [202] R. S. Sutton, “Generalization in reinforcement learning: Successful examples using sparse coarse coding,” *Advances in neural information processing systems*, vol. 8, 1995.
- [203] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [204] E. Urtans and A. Nikitenko, “Survey of deep q-network variants in pygame learning environment,” in *Proceedings of the 2018 2nd International Conference on Deep Learning Technologies*, 2018, pp. 27–36.
- [205] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
- [206] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

- [207] “Welcome to stable baselines docs! - rl baselines made easy — stable baselines 2.10.2 documentation.” [Online]. Available: <https://stable-baselines.readthedocs.io/en/master/>
- [208] T. Xie, N. Jiang, H. Wang, C. Xiong, and Y. Bai, “Policy finetuning: Bridging sample-efficient offline and online reinforcement learning,” *Advances in neural information processing systems*, vol. 34, pp. 27 395–27 407, 2021.
- [209] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian, “Selecting empirical methods for software engineering research,” *Guide to advanced empirical software engineering*, pp. 285–311, 2008.
- [210] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, “Is neuron coverage a meaningful measure for testing deep neural networks?” in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 851–862.
- [211] B. C. Hu, L. Marsso, K. Czarnecki, R. Salay, H. Shen, and M. Chechik, “If a human can see it, so should your system: Reliability requirements for machine vision components,” in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 1145–1156.
- [212] S. Swaminathan, D. Garg, R. Kannan, and F. Andres, “Sparse low rank factorization for deep neural network compression,” *Neurocomputing*, vol. 398, pp. 185–196, 2020.
- [213] J. Chen, J. Ge, and H. Zheng, “Actgraph: Prioritization of test cases based on deep neural network activation graph,” *Automated Software Engineering*, vol. 30, no. 2, p. 28, 2023.
- [214] A. Bau, Y. Belinkov, H. Sajjad, N. Durrani, F. Dalvi, and J. Glass, “Identifying and controlling important neurons in neural machine translation,” *arXiv preprint arXiv:1811.01157*, 2018.

APPENDIX A CO-AUTHORSHIP

The research studies in this thesis were published as follows:

- Deploying deep reinforcement learning systems: A taxonomy of challenges, **Ahmed Haj Yahmed**, Altaf Allah Abbassi, Amin Nikanjam, Heng Li, Foutse Khomh, in *The 39th IEEE International Conference on Software Maintenance and Evolution (IC-SME2023)*
- DiverGet: A Search-Based Software Testing Approach for Deep Neural Network Quantization Assessment, **Ahmed Haj Yahmed**, Housseem Ben Braiek, Foutse Khomh, Sonia Bouzidi, Rania Zaatour, in *Journal of Empirical Software Engineering (EMSE)*, 2022, Springer
- An Intentional Forgetting-Driven Self-Healing Method For Deep Reinforcement Learning Systems, **Ahmed Haj Yahmed***, Rached Bouchoucha*, Housseem Ben Braiek, Foutse Khomh, in *The 38th IEEE International Conference on Automated Software Engineering (ASE2023)*

The following publications/submissions were produced during my Master's degree studies but is not directly related to the content of the thesis:

- An empirical study of challenges in converting deep learning models, Moses Openja, Amin Nikanjam, **Ahmed Haj Yahmed**, Foutse Khomh, Zhen Ming Jack Jiang, in *The 38th IEEE International Conference on Software Maintenance and Evolution (IC-SME2022)*
- DeepEvolution: A Search-based Data Transformation for Deep Learning Model Testing, Housseem Ben braiek, **Ahmed Haj Yahmed***, Rached Bouchoucha*, Foutse Khomh, Sonia Bouzidi, submitted to *Transactions on Software Engineering and Methodology (TOSEM)*.