Common Challenges of Deep Reinforcement Learning Applications Development: An Empirical Study

Mohammad Mehdi Morovati, Florian Tambon, Mina Taraghi, Amin Nikanjam, Foutse Khomh

Noname manuscript No.
(will be inserted by the editor)

Common Challenges of Deep Reinforcement Learning Applications Development: An Empirical Study

Mohammad Mehdi Morovati · Florian Tambon · Mina Taraghi · Amin Nikanjam · Foutse Khomh

Received: date / Accepted: date

Abstract Machine Learning (ML) is increasingly being adopted in different industries. Deep Reinforcement Learning (DRL) is a subdomain of ML used to produce intelligent agents. Despite recent developments in DRL technology, the main challenges that developers face in the development of DRL applications are still unknown. To fill this gap, in this paper, we conduct a large-scale empirical study of **927** DRL-related posts extracted from Stack Overflow, the most popular Q&A platform in the software community. Through the process of labeling and categorizing extracted posts, we created a taxonomy of common challenges encountered in the development of DRL applications, along with their corresponding popularity levels. This taxonomy has been validated through a survey involving 59 DRL developers. Results show that at least 45% of developers experienced 18 of the 21 challenges identified in the taxonomy. The most frequent source of difficulty during the development of DRL applications are *Comprehension*, *API usage*, and *Design problems*, while *Parallel processing*, and *DRL libraries/frameworks* are classified as the most difficult challenges to address, with respect to the time required to receive an accepted answer. We hope that the research community will leverage this taxonomy to develop efficient strategies to address the identified challenges and improve the quality of DRL applications.

This work was supported by: Fonds de Recherche du Québec (FRQ), the Canadian Institute for Advanced Research (CIFAR) as well as the DEEL project CRDPJ 537462-18 funded by the National Science and Engineering Research Council of Canada (NSERC) and the Consortium for Research and Innovation in Aerospace in Québec (CRIAQ), together with its industrial partners Thales Canada inc, Bell Textron Canada Limited, CAE inc and Bombardier inc.

Mohammad Mehdi Morovati · Florian Tambon · Mina Taraghi · Amin Nikanjam · Foutse Khomh
SWAT Lab., Polytechnique Montréal, Montréal, Canada
E-mail: {mehdi.morovati,florian-2.tambon,mina.taraghi,amin.nikanjam,foutse.khomh}@polymtl.ca

Keywords Deep Reinforcement Learning · Machine Learning · Deep Learning · Stack Overflow · Programming Issues · Software Reliability · Empirical Study

1 Introduction

Reinforcement Learning (RL) has begun making its mark across a range of industrial sectors, from autonomous vehicles [2] and traffic engineering [56] to healthcare systems [59]. Recently we have been also witnessing an increasing adoption of RL to solve different software engineering tasks, from automatic code improvement [55], to test case prioritization [6], and program debloating [22]. Reinforcement Learning differs significantly from other subcategories of Machine Learning (ML) such as supervised and unsupervised learning, as it includes an agent that interacts with an environment to learn how to perform a sequence of actions leading to the best cumulative final rewards [41]. In other words, in RL, an agent learns to act in a way that modifies its behavior gradually to achieve the best final result; which makes traditional software quality assurance techniques inadequate for RL.

Deep Reinforcement Learning (DRL) is an integration of Deep Learning (DL) and RL, also known as Deep RL, to address challenges, such as high-dimension input data [4]. Combining DL and RL enables DRL to discover compact low-dimensional representations of high-dimensional data automatically [4].

Although there exist studies on testing and debugging RL programs [64, 53], the main challenges and obstacles that developers face while developing RL applications are still unclear. Moreover, because of basic differences between the paradigm of traditional software applications and ML applications [37, 26], it is expected that developers of ML applications face different types of challenges in the implementation process of such applications. Thus, DRL developers may face different challenges from other types of software systems (including traditional software systems as well as other subcategories of ML applications). As an example, Listing.1 shows a SO post (70562317) related to a DRL application, representing a challenge in implementing the method to choose an optimal action which is specific to DRL development and differs from ML and DL development challenges.

Although there exist some studies regarding challenges in the development of DL [61, 47], ML applications [33, 51], to the best of our knowledge there is no study on the challenges that developers face when developing DRL applications. The study by Yahmed et al. [58] is the most closely related work to this research. It examines the challenges that developers face during the deployment process of DRL systems but does not consider the challenges occurring in the early development phases prior to deployment. In this study, we examine the following research questions:

RQ1. *What are the common challenges of DRL application development?*
RQ2. *How are the identified challenges perceived by DRL practitioners?*

```
def act(self, some_input, state):
    mu, var, state_value = self.model(some_input, state)
    # mu contains info required for gradient
    mu = mu.data.cpu().numpy()
    # mu is detached and now has forgot all the operations
    performed in "self.action_head"
    sigma = torch.sqrt(var).data.cpu().numpy()
    action = np.random.normal(mu, sigma)
    action = np.clip(action, 0, 1)
    action = torch.from_numpy(action/1000)
    return action, state_value
```

Listing 1: SO post (70562317) showing a challenge in the development of the RL action.

RQ3. *Are DRL application development challenges language- and/or framework-specific?*

To answer these research questions, we manually examined and categorized 927 Stack Overflow (SO) posts that are related to DRL development. We report our results as a taxonomy of challenges in DRL application development. Besides, we conducted a survey of DRL developers/practitioners to validate our findings. Moreover, we investigated the dependency of the identified challenges on programming languages and libraries/frameworks used for DRL development. The contributions of this study are summarized as follows.

- We provide the first large-scale empirical study of the challenges in the development of DRL applications,
- We categorize challenges in DRL application development and propose a taxonomy,
- We conduct a survey with DRL practitioners to validate the identified common challenges of DRL application development,
- We examine the relationship between the identified challenges and the programming languages and libraries/frameworks used to develop DRL applications.

The rest of the paper is as follows. We describe the methodology of our study in Section 2. In Section 3, we report our findings including the taxonomy of DRL development challenges. Section 4 discusses the implications of the highlighted findings. Afterward, we review related works in Section 5. Threats to the validity of our research, and conclusion/future works are discussed in Section 6 and Section 7, respectively.

2 Methodology

This section describes the methodology we follow in this study. This methodology is illustrated in Fig.1.

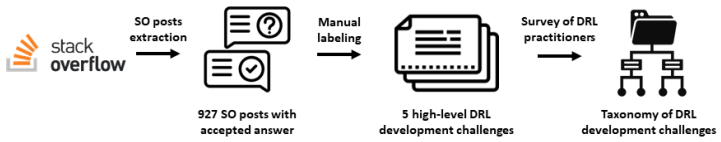


Fig. 1: High-level view of the used methodology.

2.1 Extracting Posts from Stack Overflow (SO)

We rely on Stack Overflow (SO) as the main source of information in this study; similar to several previous studies which utilized data exclusively obtained from SO for analyses [61, 1, 18]. SO is known as the largest technical question and answer (Q&A) website creating a public knowledge base in various areas [63], with 23.4 million questions and 19.6 million users as of December 2022 [52]. In the software development community, SO provides a platform for developers to exchange about coding issues; improving their coding knowledge. To extract SO posts related to DRL, we use Stack Exchange Data Explorer[1] which provides access to up-to-date SO data. Overall, we use a list of DRL-related tags and keywords to collect DRL-related SO posts. Listing.2 presents an example of a query used to collect SO posts containing *'deep-learning'* and *'reinforcement-learning'* tags at the same time.

```
SELECT * FROM Posts
WHERE Tags LIKE '%<deep-learning>%'
AND Tags LIKE '%<reinforcement-learning>%'
```

Listing 2: Sample query to extract SO posts that have *'deep-learning'* and *'reinforcement-learning'* tags simultaneously.

To gather the list of DRL-related tags and keywords, we follow previous study [1] using a snowballing approach in which we start with posts that have *'deep-learning'* and *'reinforcement-learning'* tags, simultaneously. In the next step, we collect all tags assigned to the SO posts gathered in the previous step. Then, we include DRL-related tags to our list of tags (e.g., *'dqn'*). We continue this process and expand the list of DRL-related tags until we are unable to add any new tags to our list. Besides, we create a list of DRL-related keywords based on the list of collected DRL-related tags. Firstly, we include all DRL-related tags (such as 'reinforcement learning') to the list of DRL-related keywords. Moreover, we add expanded forms of DRL-related tags which are acronyms. As an example, we add *'Deep Q-Learning'* which is the expanded form of *'dqn'*. The complete list of tags and keywords used to extract SO posts are available in our replication package [38]. In summary, we collected SO posts that meet at least one of the following criteria:

- Posts having one of the identified tags (e.g., '*drl*', '*dqn*', etc.)
- Posts with a combination of identified tags (e.g., combination of '*deep-learning*' and '*reinforcement-learning*')

[1] https://data.stackexchange.com/stackoverflow/query

- Posts with a combination of identified tags and keywords in their title or body (e.g., '*reinforcement-learning*' tag and '*deep*' in the post title)
- Posts including identified keywords in their title or body (e.g., '*drl*')

After extracting all posts and removing duplicates, we obtained 3083 posts. Then, we filtered out the posts without an accepted answer which leaves us with 927 posts. We chose to remove posts without an accepted answer, similar to previous studies [41, 21] because the correctness of any of such responses would not be inferable, which could potentially bias our results. We also collected information about the time taken by each post to receive an accepted answer and used it as an indicator of the level of difficulty of the question, similar to the approach employed in previous works [20, 60].

2.2 Manual Inspection

During this step, a team of four raters (three Ph.D. candidates and a senior research staff who are practitioners of DRL development) is responsible for labeling the collected SO posts. Following a methodology similar to prior research conducted by [24, 25], we split the collected SO posts into 10 parts each inspected in a dedicated labeling round. All the discussions and referenced source codes in each post were thoroughly reviewed. The raters use an open coding method [32] to label SO posts and categorize them. Each SO post is reviewed by two raters. We use the *Google Sheet* platform [17] to save all extracted labels in an online environment. That is, all raters put generated labels in the shared document, but they do not have access to the labels that other raters assigned to each post. After labeling SO posts in each round, the raters meet to discuss disagreements and resolve conflicts. In case they failed to resolve a disagreement, a third rater reviews the SO post and makes a decision about its label, acting as a tie-breaker. Besides, the raters review the generated labels in each meeting to ensure their comprehensiveness and granularity (combining similar labels generated by different raters or dividing a label into separate ones). In case of changing existing labels, raters re-review the previously labeled posts to ensure that assigned labels are in synchronization with newly generated labels. We made this decision to allow for continuous improvement of the labeling process. This way raters have the opportunity to resolve their conflicts at the end of each round, similar to the technique used in previous studies [24, 25]. Besides, in the case that any of the raters suggests generating a new label, all raters meet to discuss and reach an agreement on that new label. After completing labeling all 927 posts, all raters meet to finalize the generated labels, categorize them, and create the taxonomy. Then, the first two authors review all of the labeled posts again to ensure that their assigned labels are in sync with the final generated taxonomy. Regarding the posts in which the questioner asks more than one question belonging to two different categories, we repeat that record in our dataset and assign a different label to each record. Although we could not report inter-rater agreement level due to the lack of prior defined categories, we calculate inter-rater agreement

Table 1: Detailed information on the manual labeling process.

Round	Analyzed SO posts	Conflicts	Relevant to DRL	New categories
1	100	11	95	15
2	100	7	93	1
3	72	10	67	–
4	100	16	96	4
5	100	14	90	1
6	100	9	96	–
7	100	5	95	–
8	100	7	93	1
9	100	5	94	–
10	55	3	51	–
Total	**927**	**87**	**870**	**22**

between the pair of raters who investigate each SO post after finalizing the labels using Cohen's Kappa [34] and obtained an 86% agreement level. Table. 1 presents detailed information on the labeling procedure.

During the manual inspection of SO posts, we filtered out 57 posts that were not related to DRL development. Generally, some questioners may add DRL-related tags to their posts by mistake or because of unfamiliarity with DRL and its real capabilities to solve their considered problems (such as #60958362). We also filtered out posts that are too general and which could not be considered as reporting about a challenge in DRL development (e.g., #3972812).

It is worth mentioning that 70% of the DRL-related questions in SO still remain without any accepted answer. This is consistent with previous findings by Alshangiti et al. [1] that 61% of ML-related SO posts remain without any accepted answer. Multiple factors could explain this finding. In some cases, the person who asks the question responds to the question after a while, but she does not assign the accepted answer badge to the post (e.g., #45364837). Some users also ask basic questions irrelevant to DRL but assign DRL-related tags to them. These questions receive negative scores and remain without any accepted answer (e.g., #50544568). We also observed posts where the person asking the question forgot to assign the accepted answer badge to an answer, based on upvotes to the response, comments of the person who asked the question, or other people with the same problem under the response (e.g., #63250935). About the posts with accepted answers, it should be mentioned that 16% of them have been answered by the user who published the question. This usually happens when a user asks a very specific question that remains unanswered for a long time, and then the same user finds the response elsewhere and adds it to his original post (e.g., #2723999).

2.3 Taxonomy Construction and Validation

Similar to other studies [54, 24], we use a bottom-up methodology to create the taxonomy. To this end, we first categorize labels belonging to the same theme

into groups. Next, we build up parent nodes in a way that makes sure that categories and their subcategories adhere to a *'is a'* relationship. Each update on the taxonomy (e.g., adding a new category, subcategory, or combining two categories/subcategories) that generated a new version of the taxonomy was debated by all authors at a meeting. After integrating all suggested updates, all the authors of the paper met again to inspect the final categories and subcategories of the taxonomy and finalize the taxonomy.

The comprehensiveness and representativeness of the obtained taxonomy are assessed using a survey with DRL developers/practitioners (not involved in the construction of the taxonomy). We collect a list of survey participants from collaborators of GitHub repositories related to DRL. Specifically, we extract GitHub repositories mentioning *'deep reinforcement learning'* in their description using GitHub's search API V3[2]; a rest API that receives a query and returns a list of repositories that satisfy conditions stated in the query. From this search, we obtained 7244 repositories.

Given that GitHub search API limits access to only the first 1000 results, we follow the methodology used in [37] and run several different queries to achieve less than 1000 repositories for each query. That is, we divide the duration of the search for repository creation date between Jan 1, 2010, and Jan 31, 2023 (the date of running queries) into snapshots of 1 month. Thus, we execute 157 GitHub search requests to collect 7244 repositories. We then filter out forked and disabled repositories. In the next step, we check the repositories' contributors and collect the contributors mentioning their email addresses, obtaining 2531 unique email addresses of developers. We use Qualtrics [46], an online survey tool for designing and conducting surveys, to create survey forms. Table. 2 presents the structure of survey questionnaires provided to DRL application developers. We start the survey with general questions regarding the participant's current position and experience in DRL development. Next, we continue with specific questions about the finalized taxonomy. Since representing the whole taxonomy in the survey as a figure might be complicated and hardly understandable, we split it into its main categories. We also provide a description for each subcategory (challenges) to give participants a clear picture of each challenge. For each identified challenge, we ask three questions including 1) a 'yes/no' question identifying whether the answerer has faced the identified challenge, 2) the severity of the challenge, and 3) the amount of effort required to address the challenge. If a participant answers *'yes'* to the first question, s/he will have the next two Likert-scale questions which are related to the severity of challenges and the required effort to address them. We also provide a free-text question in the final part of the survey asking the participants about any challenges in developing DRL applications not listed in our provided taxonomy. These free-text questions allow us to collect possible challenges that we may have missed in the taxonomy. The full questionnaire of the survey is available in our replication package [38].

[2] https://docs.github.com/en/rest

Table 2: Survey structure

Sec.1	**General questions**								
	Email address (optional)								
	Years of experience on DRL								
	...								
Sec.2	**DRL development challenges review**								
	DRL Challenges	***Facing this challenge***		***Challenge severity***			***Needed effort to address***		
	Comprehension	◯*Yes*	◯*No*	◯*Minor*	◯*Major*	◯*Critical*	◯*Low*	◯*Medium*	◯*High*
	Reward	◯*Yes*	◯*No*	◯*Minor*	◯*Major*	◯*Critical*	◯*Low*	◯*Medium*	◯*High*
	...								
Sec.3	**Free-text question**								
	Any challenge that we have missed								
	Any suggestion or comment								

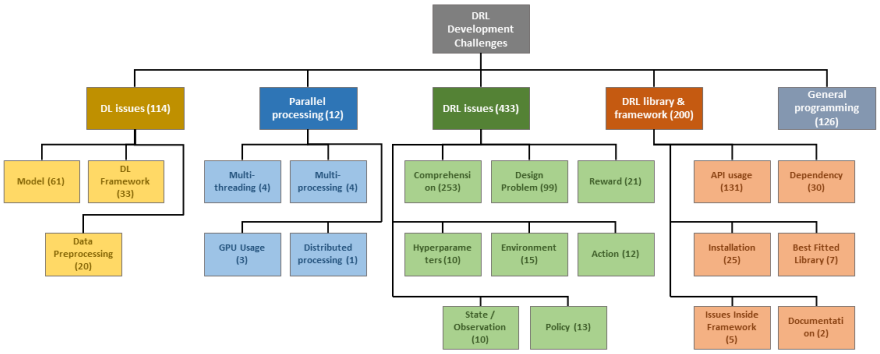


Fig. 2: Taxonomy of common challenges in DRL development.

3 Results

This section presents and discusses the results of our study. All the materials used in this study, including the collected data, are publicly available online in our replication package at [38].

RQ1: What are the common challenges of DRL application development?

The final taxonomy of challenges of DRL application development is arranged into a tree structure with five high-level categories in which leaves (subcategories) refer to the challenges. Fig.2 shows the taxonomy of common challenges in the development of DRL applications. The number in parentheses presented next to each category/subcategory is the absolute number of identified SO posts that are categorized into that category (the absolute frequency of each challenge). To give a better understanding of identified challenges, a brief description of each category/subcategory is provided in the following.

DRL Issues. This category focuses on the challenges that developers may face while developing the DRL part of their applications. So, all challenges in this category are specific to DRL applications.

a. *Design Problem:* Instances where the user asks for advice for designing a solution and implementing a DRL application for their specific problem or scenario. For example, developers asked for recommendations to implement different parts of DRL applications for Curve Fever or mini-golf games. Another such challenge in this category is related to designing the properties of each object in a tank game.
b. *Comprehension:* Challenges about the meaning or details of theoretical concepts in DRL, i.e., misunderstanding about basic formulas of different DRL algorithms. For instance, a developer mentioned *"I'm trying to make a learning football game from scratch using Deep Q-learning algorithm (without convolutional network though). I just couldn't figure out what does Φ stand for in this algorithm."*, or one which is related to the difference between SARSA and Q-learning algorithms in terms of collecting the next policy value.
c. *Policy's Loss:* This category refers to challenges about DRL learning policy's loss. As an example, questions regarding implementing a customized loss function or any problems in loss calculation methods are categorized in this group.
d. *Reward:* Challenges in the implementation of reward, e.g., not using the negative reward to penalize each added time step. Another such example of this category is a post asking *"I am implementing the basic RL algorithm to play the game Flappy Bird. I want to be able to process the screen and recognize whether a point has been scored or the bird has died. Processing the screen returns a stacked numpy array. The reward function then needs to assign a reward to the provided array, but I have no idea how to go about this"*.
e. *Action:* Questions/Challenges related to the action(s). e.g., possible actions in a specific game or implementing a *'chooseAction'* method for a PacMan bot.
f. *State/Observation:* Questions/Problems regarding the state(s) or the agent observation(s), e.g., handling large state spaces. An example of such instance is a user asking *"I implemented a 3x3 OX game by q-learning (it works perfectly in AI v.s AI and AI v.s Human), but I can't go one step further to 4x4 OX game since it will eat up all my PC memory and crash... Since I need to calculate each Q value (for each state, each action), I need such a large number of array, is it expected? any way to avoid it?"* and the accepted answer provided them with suggestions for reducing their state space size by considering symmetries and other tricks.
g. *Environment:* Questions/Problems pertinent to the environment, e.g., designing a custom environment. For example a user asked *"I'm very new to Ray RLlib and have an issue with using a custom simulator my team made. We're trying to integrate a custom Python-based simulator into Ray RLlib to do a single-agent DQN training. However, I'm uncertain about how to integrate the simulator into RLlib as an environment"*.
h. *Hyperparameters:* Questions/Problems related to the hyperparameters of the RL algorithm, e.g., setting the discount factor too high. A good example

of this group is demonstrated in a post where the user shared the code for their learning algorithm and reported that the loss keeps increasing and the model is not learning. The accepted answer states that *"The main problem I think is the discount factor, gamma. You are setting it to 1.0, which means that you are giving the same weight to the future rewards as the current one"*.

DRL Libraries/Frameworks. This category refers to the challenges that developers face when they are trying to use DRL-specific libraries/frameworks (e.g., KerasRL [44], RLlib [30], Tensorforce [49], etc.). Challenges that software developers face when using libraries/frameworks have been extensively studied for traditional software systems development [13, 39] and also for DL applications development [3]. However, despite the large number of SO posts related to the usage of DRL libraries/frameworks (i.e., 200 SO posts), these challenges are yet to be examined for DRL application development.

a. *Installation:* Questions/problems regarding installing/uninstalling DRL-related libraries/frameworks or missing libraries. Issues categorized in this subcategory can often stem from an incompatibility between DRL-related frameworks/libraries and other libraries. For example, a user described her issue as *"when I try to install gym[box2d] I get the following error: I tried: pip install gym[box2d]. on anaconda prompt I installed swig and gym[box2d] but I code in python3.9 env and it still not working (my text editor is pycharm) gym is already installed"*..
b. *Dependency:* This subcategory includes questions/problems about the mismatch between versions of installed libraries/frameworks and problems in installed versions of libraries, e.g., when the version of the installed *OpenAI Gym* is not compatible with *Python*. An instance of this is a user reporting getting an error installing OpenAIGym and the answer pointed out that *"the error means that the package has dependency requirements that conflict with one another"*.
c. *API usage:* This subcategory includes questions about the usage of arguments, attributes, methods, etc. of an API. It also includes questions about the default values, implemented method, existence of attributes, or methods in an API. An example from this group of issues is a user reporting not knowing how to get the weights of the network using the correct API methods: *"I'm using RLlib to train a reinforcement learning policy (PPO algorithm). I want to see the weights in the neural network underlying the policy. After digging through RLlib's PPO object, I found the TensorFlow Graph object. I thought that I would find the weights of the neural network there. But I can't find them"*.
d. *Documentation (using newly added features):* This subcategory of issues occurs when a developer wants to use a feature of a DRL library/framework, but there is no documentation for it. For example, a user who could not find the required documentation for the *Neural Network Approximator* in *ReinforcementLearning.jl*: *"I have decided to use a Neural Network Approximator. But the docs do not discuss much about it, nor are there any*

examples where a neural network approximator is used. I am stuck on how to figure out how to use such an approximator".

e. *Best fitted library for a special task (library suitability):* Instances where the user asks about the best DRL libraries/frameworks for customizing agents, based on the requirements of the problem. An instance of this group was observed in a post where a user had a customized state space and was looking for a library that supports it: *"I've had some luck training an agent using keras-rl, specifically the DQNAgent, however, keras-rl is under-supported and very poorly documented. Any recommendations for RL packages that can handle this type of observation space? It doesn't appear that openai baselines, nor stable-baselines can handle it at present".*

f. *Problems inside DRL frameworks:* Including issues that are encountered because of internal faults, i.e., bugs in the DRL frameworks. For example, a user kept getting a *numpy* error when calling the *model.learn()* function and it was found to be an official bug in the used library *Stable Baselines3*.

DL Issues. This category represents the challenges that arise specifically from the DL part of DRL applications. As the challenges belonging to this category are shared by both DL and DRL applications, we use the high-level categories of the taxonomy provided by Humbatova et al. [24] for DL applications.

a. *Model:* Questions regarding the DL model including model layer, activation function, load/save model, etc. For instance, a user who was implementing a DRL model asked for advice on back propagation: *"I am struggling with the implementation of the back propagation. Since the rewards are so big, the error values are huge, which creates huge weights. After a few training rounds, the weights to the hidden layer are so big, my nodes in the hidden layer are only creating the values -1 or 1".*

b. *Data Preprocessing:* Questions about preparing data to be fed into the DL model, e.g., the shape of the input matrix. As an example there was a user that asked *"I am learning how to use Gym environments to train deep learning models built with TFLearn. At the moment my array of observations has the following shape: (210, 160, 3). Any recommendations on what is the best way to reshape this array so it can be used in a TensorFlow classification model?"*

c. *DL framework:* Questions about the usage of DL frameworks (e.g., Keras, TensorFlow, PyTorch, etc.) in development of DL part in DRL applications. For instance, a user who wanted to use *Huber loss* in their model which was written using *Keras*, but this loss function is not readily available in this library, and the accepted answer implements the *Huber loss* function.

Parallel Processing & Multi-threading. This category focuses on the challenges associated with running DRL applications as parallel or distributed applications. While running different types of DL applications in a parallel style is a common practice (e.g., using GPU or multi-core CPU), it is important to note that the architecture of DRL applications differs from other types of DL applications. As a result, running a DRL application as a parallel or

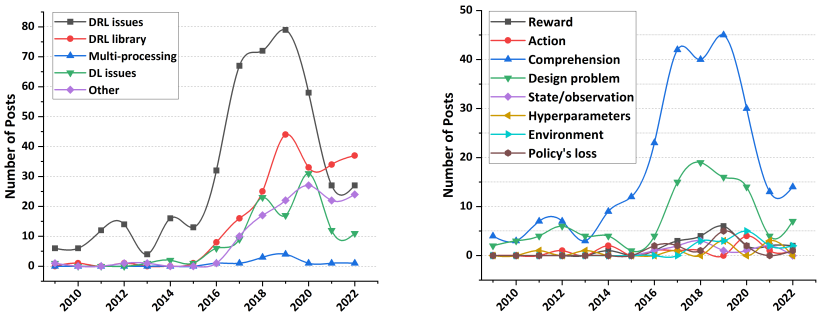
distributed application introduces new challenges that may not occur in DL application development.

a. *GPU usage:* Questions/Problems regarding utilizing GPU for running DRL applications. For example, a user was facing performance issues when training a DQN on GPU: *"I am try to train a DQN model with the following code. The GPU (cuda) usage is always lower than 25 percent. I know the tensorflow backend is consulting the GPU resources, but the usage is low. Is there any way I can improve the utilization of the GPU (When I train a CNN network, the GPU (cude) utilization is around 70 percent)?"*
b. *Distributed processing:* Questions/Problems about running DRL applications as a distributed software. For example when a user wanted to implement the *Asynchronous Advantage Actor Critic (A3C)* model for reinforcement learning in their local machine, they posed a question about the possibility of implementing it in a distributed manner: *"Would it be easier/faster/better to implement this using the distributed TensorFlow API? In the documentation and talks, they always make explicit mention of using it in multi-device environments. I don't know if it's an overkill to use it in a local async algorithm"*.
c. *Multi-threading:* Questions/Problems about running DRL applications as a multi-threaded software. An example of this category was observed in a post where the user asked about the possibility of reducing the training time by running it on multiple threads concurrently: *"My friend and I are training a DDQN for learning 2D soccer. I trained the model about 40.000 episodes but it tooks 6 days. Is there a way for training this model concurrently? For example, I have 4 core and 4 thread and each thread trains the model 10.000 times concurrently. Therefore, time to training 40.000 episodes are reduced 6 days to 1,5 days like parallelism of for loop"*.
d. *Multi-processing:* This subcategory refers to challenges stemming from running DRL applications on multiple processing units (e.g., multiple CPUs). As an example, a user asked a question about using Ray[3] in a multi-processing style.

General Programming Issues. This category contains programming and coding mistakes occurring when developing DRL applications. The challenges in this category could not be classified in any of the other categories described above. For example, a user had a question about how to slice a 3D *numpy* array in an RL finite MDP application (#67089715).

Overall, the majority of the analyzed SO have been assigned to categories *Comprehension*, *Design problem*, *Model*, and *API Usage*. Aside from *Design problem* related questions, which are often quite specific (i.e., related to particular implementations of DRL), the majority of questions asked by DRL developers concern issues that apply to DRL applications in general.

[3] https://www.ray.io/



(a) (b)

Fig. 3: Distribution of DRL related SO posts per years (a) high-level categories of the provided taxonomy, (b) subcategories of *'DRL issues'* category.

Finding 1: The taxonomy of challenges in DRL development is structured into five main categories including *DRL issues*, *DL issues*, *DRL libraries/frameworks*, *parallel processing & Multi-threading*, and *general programming issues*, where *DRL issues* (48.9%) and *DRL libraries/frameworks* (22.6%) categories are the most popular. Among the challenges, *comprehension* (28.6%), *API usage* (14.8%), and *design problem* (11.2%) are the most prevalent DRL development challenges.

Fig.3 presents the distribution of SO posts related to DRL application development for a period of 13 years, from 2009 to 2022. From Fig.3a, we observe a substantial surge of inquiries about DRL development in 2016; reaching a peak in 2019 and 2020. Additionally, as can be seen on Fig.3b, *comprehension* and *design problem* questions dominated posts about DRL application development challenges. It is also noticeable that *API usage*, the second most common DRL application development challenge, was at its peak in 2018.

Fig.4 presents the distribution of time taken by SO posts from different categories to receive an accepted answer. This duration is an indicator of the level of difficulty of the questions contained in the SO posts [20, 60]. *Parallel Processing* is the category with the highest average time taken before receiving an accepted answer. This can be explained by the fact that using multi-processing or distributed processing in DRL is not necessarily widespread and also requires particular knowledge and expertise. The remaining categories need a nearly similar average time frame to receive an accepted answer, with the *general programming issues* category having the shortest average duration. We attribute the shortest average time of the *general programming issues* category to the fact that this category contains generic challenges that do not require expertise in DL or DRL.

Although we note a high proportion of outliers in Fig.4, we have a relatively low median, first, and third quartiles, for all categories compared to

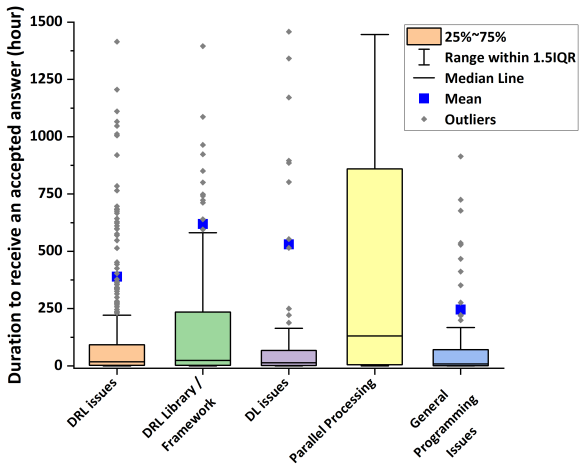
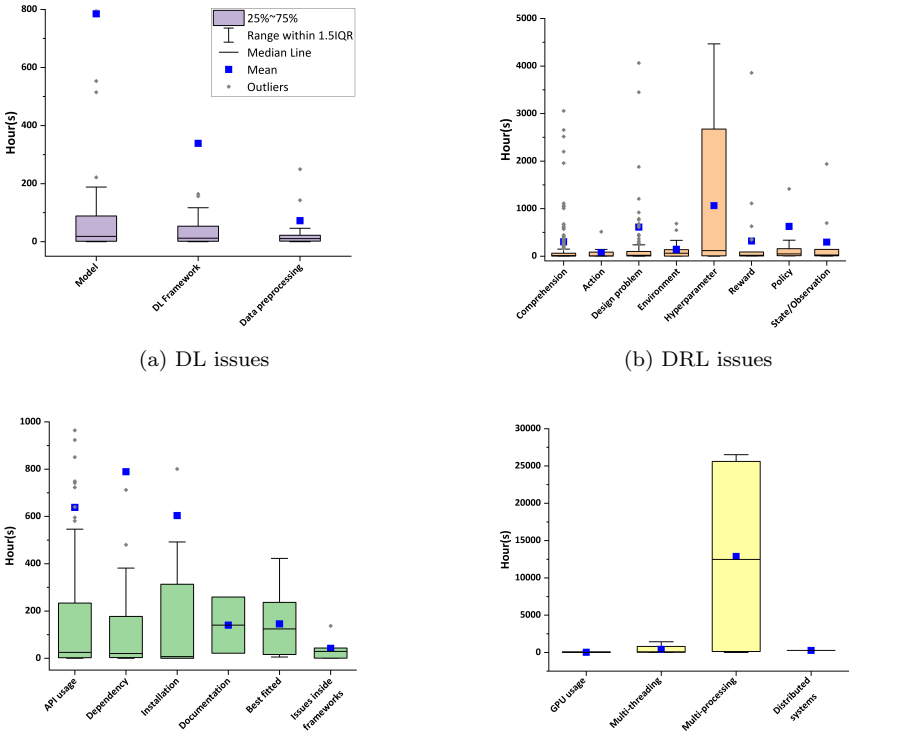


Fig. 4: Duration to receive an accepted answer (hours).

the average. Meaning that, although the majority of posts received an accepted answer in a relatively short period of time (in general less than 10 days), a sizable number of posts required a longer time (more than 20 days). Therefore, even within categories, there is quite some discrepancy inside the subcategories (challenges) themselves. As can be seen on Fig.5a, SO posts categorized as *Model* have a higher average waiting time to receive an accepted answer than the other subcategories of the *DL Issues* category. Among subcategories in *DRL Issues*, *hyperparameters* includes SO posts with the highest average required time for receiving an accepted answer (Fig.5b). For *DRL libraries/frameworks*, SO posts belonging to *API usage*, *Dependency* and *Installation* subcategories require the longest average time before receiving an accepted answer (Fig.5c). Regarding subcategories within *parallel Processing & multi-threading*, it is notable that the average duration required to receive an accepted answer for SO posts classified under *multi-processing* subcategory exceeds one year. It should be also taken into account that the small number of SO posts in this subcategory might bias the results, with respect to the fact that small data may not represent the distribution of classes in the population adequately [10].



(a) DL issues

(b) DRL issues

(c) DRL libraries/frameworks

(d) Parallel processing & multi-threading

Fig. 5: SO posts' required time to receive an accepted answer.

Finding 2: The proportion of SO posts containing questions related to DRL application development increased significantly during the period (2016 - 2021), reaching a peak between 2019 and 2020. In terms of average time before the reception of an accepted answer, *Parallel processing* is the category that required the longest period of time, in comparison to the other categories. Overall, we observe a high variance in the time to answer of the posts from the different categories of challenges identified in our taxonomy, suggesting that their impact on the development process of DRL applications is not equal.

RQ2: How are the identified challenges perceived by DRL practitioners?

We cross-checked the taxonomy generated based on SO posts using a validation survey. **59** ML practitioners participated in our survey to assess our identified challenges, including 62% researchers (Master's and Ph.D. students, research assistants, and professors), 19% ML/SE engineers, 11% developers, and 8%

Table 3: Result of the survey of DRL practitioners.

Challenges		Response		Severity			Required effort		
		Yes	No	Minor	Major	Critical	Low	Medium	High
DRL Issues	Comprehension	**46%**	54%	68%	24%	**8%**	59%	24%	**17%**
	Reward	**86%**	14%	29%	39%	**32%**	24%	39%	**47%**
	Action	47%	53%	71%	12%	17%	63%	22%	15%
	Environment	**81%**	19%	31%	37%	**32%**	30%	31%	**39%**
	Hyperparameters	**78%**	22%	48%	27%	**25%**	41%	22%	**37%**
	Design problem	**76%**	24%	39%	36%	**25%**	34%	29%	**37%**
	Policy	59%	41%	54%	36%	10%	56%	29%	15%
	State/Observation	**64%**	36%	54%	31%	**15%**	53%	32%	**15%**
DRL libraries/ frameworks	Installation	41%	59%	83%	14%	3%	76%	17%	7%
	Dependency	**53%**	47%	76%	15%	**9%**	75%	20%	**5%**
	API usage	**37%**	63%	86%	9%	**5%**	78%	14%	**8%**
	Documentation	49%	51%	70%	22%	8%	73%	19%	8%
	Bugs inside framework	49%	51%	75%	13%	12%	70%	15%	15%
	Best fitted library	51%	49%	71%	25%	4%	64%	26%	10%
Parallel processing	GPU usage	56%	44%	70%	22%	8%	68%	20%	12%
	Multi-threading	46%	54%	68%	25%	7%	66%	19%	15%
	Multi-processing	51%	49%	56%	32%	12%	51%	24%	25%
	Distributed systems	42%	58%	59%	27%	14%	58%	13%	**29%**
DL Issues	Model	51%	49%	64%	29%	7%	64%	27%	9%
	Data prepossessing	49%	51%	58%	27%	15%	64%	24%	12%
	DL framework	46%	54%	66%	24%	10%	64%	27%	9%

data scientists. Among our respondents, 84% have at least 1 year of DRL development experience and 44% have more than 3 years of experience. Table. 3 summarizes the responses of participants for each DRL development challenge contained in the taxonomy. For each challenge, we provide the percentage of developers who reported having experienced that challenge (based on the answers to the *'yes'* or *'no'* question). Besides, we asked the participants about the severity of each challenge and the effort required to address them. Results show that all challenges presented in our taxonomy were encountered by the survey respondents. Moreover, no additional challenges were proposed by the survey participants through the open-text questions. This indicates that our provided taxonomy is representative of challenges faced by developers during the DRL application development.

According to the survey results, the majority of our respondents have been confronted with challenges classified as *DRL issues* (67.1% average over all subcategories in this category). This observation is aligned with the proportion of SO posts categorized as *DRL issues* (see Fig.2). Conversely, *DRL libraries/frameworks* challenges have been experienced the least with only 46.7% of respondents (which is the lowest proportion among all categories) reporting having faced challenges leveraging DRL libraries. This finding is reflected by the results of our quantitative analysis of SO posts, which show that only 21.6% of posts contained questions related to the *DRL libraries/frameworks* category.

Among the challenges identified in our taxonomy, *reward* (86%), *environment* (81%), *hyperparameters* (78%) and *design problem* (76%) are the most common challenges in DRL development, from the viewpoint of the developers who participated in our survey. Although only 14.8% of SO posts contained

questions about DRL *API usage*, 37% of survey respondents identified it as a challenging issue in DRL development. Given that *reward, environment, hyperparameters*, and *design problem* are fundamental components of the development of an RL application [31], it is expected that survey participants reported them as the most encountered challenges. That is, defining an RL environment is a crucial step that affects the convergence of an agent's behavior significantly [48]. This however contrasts with the number of SO posts identified as belonging to the *DRL environment* category (i.e., 1.6% of SO posts). We also note that *comprehension*, the most frequent challenge in terms of the number of SO posts in our taxonomy (28.6% of SO posts categorized as *DRL issues*), has been reported by 46% of survey participants as a non-challenging issue. The explanation for this variance lies in the experience level of the survey respondents. Indeed, it is indicated that the survey participants possess a medium to high level of experience in DRL development (84% of the survey participants have at least 1 year of experience in DRL development). That is to say, experienced practitioners are less likely to seek help for understanding the fundamental DRL concepts because they have already mastered these basics. Therefore, in order to fulfill the specific requirements of various DRL developers with different experience levels, it is important to acknowledge that DRL developers have unique needs at different stages of DRL development journey. Moreover, the survey was conducted in 2023; nearly a decade after DRL started to become mainstream [35, 29]. According to our 13 years analysis of the distribution of DRL-related SO posts, there has been a drastic drop in the number of SO posts categorized as *DRL issues* after its peak in 2019 (Fig.3a and Fig.3b). This phenomenon may indicate that DRL developers have progressively mastered the fundamental principles of DRL over these years leading to a reduction in challenges faced by them and a decrease in the number of DRL-related posts on SO.

Finding 3: Survey respondents encountered all the challenges included in our taxonomy. They identified *reward, design problem, environment* and *hyperparameters* as the most common challenges in the development of DRL applications. However, this observation highlights a notable contrast with the findings from the analysis of SO posts, wherein *comprehension* with 28.6% of SO posts is the most frequent challenge. This variance can be explained by the fact that more experienced practitioners may encounter different challenges and have different requirements compared to the developers who are newer to DRL.

We also ask survey respondents about the severity and needed effort to address the challenges identified in the provided taxonomy (Table.3). In general, the majority (exceeding 52%) of the survey respondents indicated that the most frequent challenges from their viewpoint (i.e., *Reward*, *Environment*, *Design problem* and *Hyperparameters*) are critical. Moreover, at least 59% of the survey participants considered the level of effort required to address these

challenges, to be "Medium" or "High". The majority (more than 54%) of the survey participants consider the other identified challenges to be of *Minor* severity level, and to require a *Low* level of effort. In general, the participants consider that *Installation* and *API usage* challenges require a low level of effort, which might signal that DRL libraries/frameworks have good documentation and usability in general [36].

We compared the time-to-answer of the posts (from different challenges categories) with the effort reported by our survey participants for the different challenges categories and made the following observations:

- *Hyperparameters*, and *design problems* are the subcategories of *DRL issues* that took longer time before receiving an accepted answer. Survey respondents also reported them as severe and requiring a high effort from ML developers.
- The average time required to receive an accepted answer for *State/observation* and *comprehension* SO questions are comparable (even though some SO posts within *comprehension* subcategory took a bit longer to receive an accepted answer than posts belonging to *State/observation*). The survey participants also assessed these two subcategories of challenges (i.e., *comprehension* and *State/observation*) as easy to resolve in general.
- *API usage* and *Dependency* challenges are the groups of challenges that took the longest time before their questions received an answer. This result is in contrast with the survey participants' estimation of the effort required to fix them.

The severity and required effort reported by the survey participants for each challenge are strongly correlated (using kendall's tau [16]) in a positive direction [14] and statistically significant ($P - value < 0.05$). Hence, more severe challenges necessitate more effort from developers.

Finding 4: Survey respondents highlighted *Reward, Environment, Hyperparameters*, and *Design problem* as the most severe and intricate-to-address challenges in DRL development.

RQ3: Are DRL application development challenges language- and/or framework-specific?

We extract information about the programming languages used to develop DRL applications from the collected SO posts. It should be mentioned that the posts were collected without any distinction on the programming language and frameworks used. Fig.6 presents the proportion of *Python* posts for the different identified categories of challenges. As can be seen, *Python* is by far the dominant programming language for all categories of challenges. However, the proportion of posts related to other programming languages and containing *DRL issues* is non-negligible (i.e., 20.2%). This high ratio is mostly attributable to *Java* (5.2% of all posts), *C++* (4.7% of all posts), and *R* (4.7%

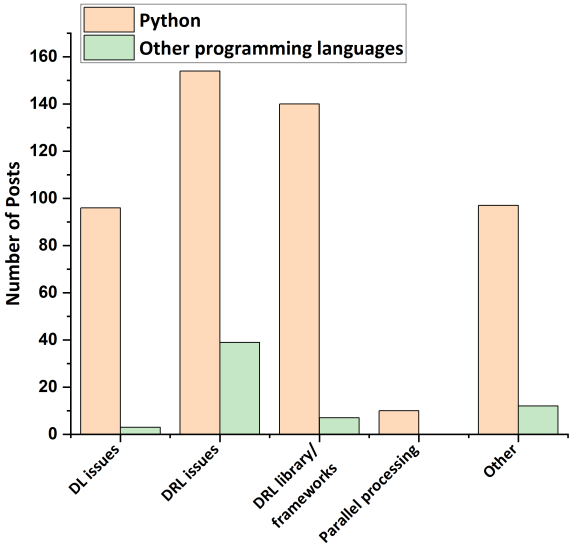


Fig. 6: Programming languages mentioned in the posts belong to various categories of challenges.

of all posts) programming languages. It is also noteworthy that investigating the relationship between used programming languages and the challenges (subcategories) within each category reveals that *Python* stands out as the predominant programming language across all DRL development challenges. Based on these results, we conclude that there is no relationship between DRL development challenges and used programming languages. This finding is in accordance with prior research [37, 24] which reported that *Python* is the most popular programming language for ML-enabled applications. These results about programming languages used in DRL applications development are also supported by our validation survey where all participants mentioned *Python* as the programming language they use for developing DRL applications. Besides, 20% of participants reported *C/C++*, and 12% mentioned other programming languages in addition to *Python* (e.g., *C#* 4%, and *Java* 3%).

We examined the types of frameworks and libraries mentioned in posts related to different challenge categories and found that *TensorFlow*, *Keras*, and *PyTorch* are the most popular libraries/frameworks as shown in Fig.7a. The majority of reported DRL challenges are related to *gym* [9], which is one of the most used DRL libraries. Fig.7b presents the libraries/frameworks frequently mentioned in DRL challenges from the *DL issues* category. Results show that *TensorFlow*, *Keras*, and *PyTorch* are the most popular libraries/frameworks in all subcategories of this category. Given that the challenges within the *DL issues* category pertain to the DL parts of DRL applications, it is not surprising to see *TensorFlow*, *Keras*, and *PyTorch* mentioned since they are the most used libraries in the development of DL-enabled applications [37, 24]. Subsequent to

these three libraries/frameworks, *ray* holds a more substantial part than other libraries/frameworks. This situation can be attributed to the fact that *ray* encompasses not only DRL-related libraries (e.g., RLlib [30]) but also various other libraries for a wide range of ML-related tasks, including scalable datasets, model training, and hyperparameter tuning [45]. About the most frequently referenced libraries/frameworks in the *DRL libraries/frameworks* category, as illustrated in Fig.7c, *gym* received the largest number of questions, especially on the topics of *API usage*, *installation*, and *dependency* challenges. Regarding parallelization and multi-threading, as can be seen on Fig.7d, the majority of issues are reported against *TensorFlow*, in particular regarding *GPU usage* and *distributed processing*. Most of the *multi-processing* challenges relate to the *ray* framework.

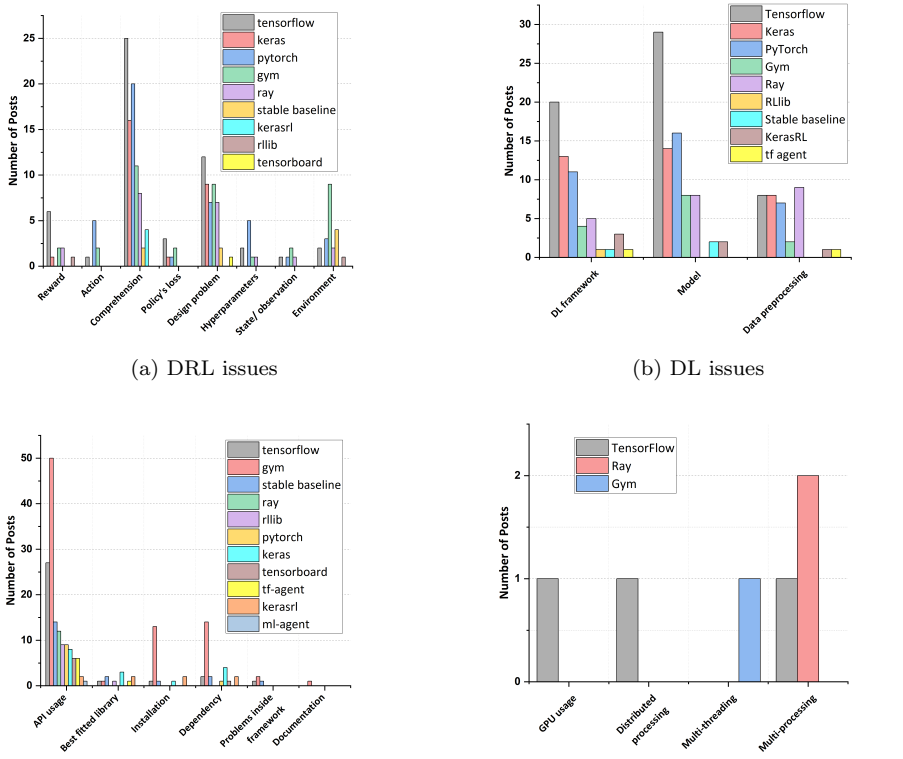
The respondents of our survey corroborated these findings; with 86% of them reporting *PyTorch* as their preferred framework, followed by *TensorFlow* (50%), and *Keras* (39%). Some participants also mentioned *KerasRL* (6%) and *JAX* (6%). We note that *PyTorch* was cited more than *Tensorflow* and *Keras* by participants of our validation survey compared to SO posts. This can be explained by the fact that SO posts do not necessarily include information about the frameworks used by users asking questions. It is also worth mentioning that we collected SO posts over a period of 13 years (from 2009 to 2022), while *PyTorch* was introduced only in 2016 (in comparison to *Tensorflow* and *Keras* released in 2015).

Finding 5: *Python* is by far the dominant programming language used for DRL development. While *TensorFlow*, *Keras*, and *PyTorch* are frequently mentioned in posts reporting challenges faced by DRL developers, *gym* is the most challenging libraries to install (see *installation*, and *dependency* challenges) and use (see *API usage* challenges). Our results also show that *ray* is the most challenging library when dealing with *multi-processing* and *data preprocessing* operations.

4 Discussion

Based on the findings of our study, in this section, we discuss the state of DRL application development and highlight some research avenues for researchers and practitioners.

Through this study, we gained a thorough understanding of frequently asked questions regarding DRL development to enable the community to explore potential approaches for mitigating these challenges, minimizing errors, and enhancing the reliability of DRL applications. Based on our provided taxonomy, one can see that some challenges faced in the development of DRL applications are common to all types of DL applications. For example, managing dependencies when using DL libraries/frameworks is a prevalent challenge in DL applications. However, dependency management can be more complex



(a) DRL issues

(b) DL issues

(c) DRL libraries/frameworks

(d) Parallel processing & multi-threading

Fig. 7: Common libraries/frameworks used for developing DRL applications.

in DRL, because of the need for synchronization among a larger number of libraries in the development of DRL applications (e.g., aligning the Python version with the DRL libraries/frameworks and the library that manages RL environment). Similarly to what was suggested by Huang et al. [23] to tackle dependency management challenges in DL applications, DRL researchers can provide a dependency knowledge graph for DRL libraries/frameworks to mitigate this challenge.

Finding 2 revealed that 27.3% of DRL development challenges categorized as *comprehension* are related to the lack of sound understanding of basic DRL concepts. In other words, 58.4% of posts belonging to *DRL issues* category (DRL-specific category) are about *comprehension* challenge. This finding highlights the need for documentation and tutorials to help DRL practitioners who are not experts in DRL, in the development of DRL applications. A roadmap for the development of DRL applications would also help developers navigate through the implementation of DRL applications with fewer misunderstandings of DRL concepts. The need for such material is emphasized by a post[4]

[4] https://stackoverflow.com/questions/37973108

asking questions about the difference between RL and DRL. By providing a roadmap that systematically expands DRL developers' understanding, developers will be supported in overcoming the most common challenge in DRL application development. An illustration of such guidance is the work conducted by Garg et al. [15] on creating a roadmap for DL development. The need for good documentation and guidance is also emphasized by the survey participants who mentioned that *'although there are a number of tutorials to start working on DRL, a few issues are shared between many of them'*. The participants also noted that many of the DRL tutorial documents cover only a specific domain of DRL. Participants also lamented the poor usability of DRL-related tutorials, claiming that they often contain a lot of unnecessary materials.

Leveraging our Findings 1 and 2, researchers can develop debugging tools to help developers identify the issues early on during DRL application development. Debugging tools can significantly reduce DRL development and maintenance costs. For instance, considering the limited documentation available for most of the DRL libraries/frameworks in comparison with DL libraries for example (e.g., *TensorFlow*), a helpful approach would be proposing techniques and tools to assist DRL developers when using different DRL APIs. This could help mitigate DRL API issues. An example of such techniques focusing on the challenges of software API usage, is the work by Xie et al. [57] which proposes an approach to automatically extract the API parameter constraints of DL libraries/frameworks.

Finding 2 of this study regarding challenges associated with installation and dependency management of DRL libraries/frameworks is aligned with previous studies on dependency management in software development [11] in general, as well as in DL applications [19]. Considering the complex nature of DRL application development (due to the communication of several libraries), challenges regarding libraries/frameworks installation and their dependency management become more intricate, compared to other types of DL applications. This highlights the need for tools (e.g., package manager) to support dependency management. As an example, researchers can provide a tool (such as Maven[5] for Java) that automates the identification and installation of DRL libraries/frameworks that are best suited for a specific Operating System (OS) and a specific version of *Python*. Additionally, such tools can assist DRL developers in synchronizing the installed DRL-related libraries/frameworks when they need to update some of them.

Our results in Finding 2 also stress the need for supporting tools and documentation, for parallel and distributed DRL application developments. Questions related to this topic took a long time before receiving an accepted answer on Stack Overflow. DRL experts could consider developing pre-configured packages to support parallel/distributed DRL application developments. In fact, Openja et al. [42] examined ML application deployment practices on

[5] https://maven.apache.org/

Docker and reported that a significant number of ML developers use Docker to manage dependencies, environment, and the execution of ML applications.

5 Related Works

We now report and discuss the related literature.

5.1 SO Posts Analysis

Beyer et al. [8] investigated the automatic classification of SO posts. They manually labeled 1000 posts, and identified 7 categories of questions: 1) API changes, 2) API usage, 3) Conceptual, 4) Discrepancy, 5) Learning, 6) Errors, and 7) Review. Leveraging the labeled dataset, they developed two approaches for the automatic classification of SO posts. In the first approach, they used the labeled dataset to extract some regular expression patterns and used these patterns to predict the category of other posts; achieving a performance of 0.91 for both precision and recall. In the second approach, they trained Random Forest and Support Vector Machine (SVM) classifiers using the labeled dataset. The best results were obtained using the Random Forest classifier, i.e., a precision of 0.88 and a recall of 0.87.

Alshangiti et al. [1] investigated SO posts related to ML development. They used a tag-based snowball sampling approach to extract SO posts related to ML, starting with the 'machine-learning' tag. Their results revealed that a higher number of ML-related posts remain without any accepted answer (61%), in comparison with general domain questions (48%). They also reported that ML-related questions need 10 times longer to receive an answer, compared to general domain questions. Next, they compared the ratio of expert users in ML and web development (the most popular domain of programming in SO), showing that the number of ML experts is significantly less than that in web development. Afterward, they reviewed the most challenging ML development phases revealing that data preprocessing and manipulation, and model deployment and environment setup are the two most error-prone phases.

Bangash et al. [7] conducted an empirical study of SO posts related to ML. They used the 'machine_learning' tag to extract 28,010 posts published between 2008 and 2018. Next, they used Latent Dirichlet allocation (LDA) [27] to categorize the extracted posts into 44 detailed topics. Then, they showed that *code errors*, *Algorithms*, and *Labeling* are the most discussed topics in ML. They then combined the 44 topics identified using LDA into 4 main groups including frameworks, implementation, sub-domain (RE), and algorithms. They reported that nearly 51% of all ML-related SO posts belong to the implementation group. Afterward, two of the researchers manually examined 230 sampled posts and reported that most of the questions stem from the fact that ML novice developers try to use ML in their software systems. They extracted

information about the number of questions with accepted answers and concluded that ML-related questions are harder to answer than general domain SO questions. They also observed that only 65.6% of ML-related SO posts have appropriate tags; which might suggest that many users are not knowledgeable enough to assign the proper ML-related tags to their posts.

Hamidi et al. [18] examined the challenges that developers may face in the development of ML systems, based on their discussion in SO. They studied 43,950 ML-related SO posts submitted between 2008 and 2020. First, they showed that *Python* is the most popular programming language for ML development, and C# and C/C++ are the least popular programming languages for ML development. Then, they reported that model building and model evaluation are the two most challenging steps in ML development while model monitoring is the least questioned phase. They also report that questions regarding model requirements, data collection/processing, and model-building steps receive less accepted answers than others. This may stem from the fact that questions about these steps are more difficult to answer or the lack of active knowledgeable developers on SO to answer questions related to these steps.

Although these previous works investigated SO questions related to ML/DL application development, to the best of our knowledge, none of them examined the challenges of DRL development specifically.

5.2 RL and DRL Quality Assurance

In this section, we report on studies about the quality assurance of RL and DRL applications.

Zhang et al. [62] proposed strategies to help DL and DRL developers detect and resolve quality issues in their applications. Nikanjam et al. [40] proposed a methodology for automatically detecting faults in DL applications, using graph transformations. Schoop et al. [50] also proposed a strategy for detecting common issues during the development of DL models. These studies primarily targeted issues occurring in the training program of DL models. The issues considered in these aforementioned study fall within the *'DL issues'* category of our provided taxonomy, which constitutes only a small portion of DRL issues. Nikanjam et al. [41] also investigated challenges categorized as *'DRL issues'* in our proposed taxonomy. They examined questions/discussions about four popular DRL frameworks (including *gym* [9], *Tensorforce* [28], *Dopamine* [12], and *Keras-rl* [44] on GitHub and SO) and extracted 329 SO posts about DRL. They categorized these posts into six groups: basic concepts, without acknowledgment, implementation issues, answered by the owner, relative questions, and others. They reported that 'without acknowledgment' and 'implementation' questions are the most common DRL-related questions in SO, accounting for 32% and 27%, respectively. They also showed that in 2% of their studied SO posts, the answer has been posted by the questioner. They report that DRL-related SO posts take an average and median time of 2.07 days and 13

hours, respectively, before receiving an accepted answer. This time period is longer than the time taken by DL-related SO posts to receive an accepted answer; which is 5 hours on average. This finding implies that DRL-related questions might be more difficult to answer than DL-related questions. Nikanjam et al. [41] also proposed a taxonomy of faults in DRL models. The most significant difference between their study and ours is their focus on the DRL model only and the fact that they only collected data about faults mentioned in SO; disregarding all the other types of questions.

Yahmed et al.[58] conducted a study on the challenges of deploying DRL systems, based on the questions that developers ask on SO. In the first step, they extracted 357 SO posts related to the deployment of DRL systems. Next, they categorized collected SO posts into 4 categories with respect to their deployment platform, including 'server/cloud', 'mobile/embedded system', 'browser', and 'game engine'. Their results showed that the number of SO posts regarding deployment has grown over the 7 studied years. They manually examined the extracted SO posts and identified 31 challenges related to DRL deployment. They grouped these challenges into 11 categories (and proposed a taxonomy): general questions, deployment infrastructure, data preprocessing, RL environment, communication, agent load/save, performance, environment rendering, agent export, request handling, and continuous learning. The proposed taxonomy has been evaluated via a survey with practitioners. Their results show that DRL developers struggle the most with deployment infrastructurs and RL environments. Their results also show that communication-related challenges (procedure, connection loss, configuration of remote setting, and model convergence) are the most difficult challenges to address, in terms of the time to an accepted answer. The main difference between the work of Yahmed et al. and our study is the focus of the study. Yahmed et al. examined DRL deployment challenges while we examined challenges faced by DRL developers during the application development phase, i.e., prior to deployment.

6 Threats to Validity

We now discuss threats to the validity of our study.

Construction validity. Our methodology and labeling process can be a potential validity threat. We have thoroughly described our process and the tags used to collect the posts. As no previous taxonomy on this subject exists, we used an open coding approach with multiple rounds and cross-checking to ensure continuous improvement and consistency of the labeling. We further validate our results via a survey with 59 DRL practitioners.

Internal validity. As users do not necessarily provide suitable tags for their questions, our search might have missed some DRL challenges. For example, post *37973108* is related to DRL, but it does not have any specific tag mentioning DRL. Nonetheless, tag usage was necessary for a consistent methodology and we believe that the number of posts gathered and analyzed (927) is sizable enough to provide a good representation of the challenges faced by DRL

developers. Moreover, we used a snowballing approach to expand our basic set of DRL-related tags used to extract DRL-related SO posts, similar to previous studies [5]. In addition to extracting posts based on the DRL-related tags, we used a set of keywords to extract DRL-related SO posts without any DRL-related tags to address this issue, as other researchers followed a similar methodology [43].

External validity. While there might exist other DRL challenges that practitioners are facing, we conducted our study using SO which is the largest technical Q&A platform in the software development community. Moreover, all the challenges identified in our provided taxonomy have been validated by our survey participants. Also, respondents of the survey did not report any other challenge that is not included in our provided taxonomy; which is a good result, with respect to completeness.

Reliability validity. We described our methodology in detail and provided a replication package [38] to allow others to replicate our results and expand our study.

7 Conclusion and Future Works

In this study, we conducted a large-scale empirical study of 927 DRL-related posts extracted from SO. We examined all posts manually to identify the challenges that developers face when developing DRL applications. We found that *Python* is by far the most popular programming language and TensorFlow, OpenAI Gym, and Keras are the most frequently used libraries/frameworks for developing DRL applications. We categorized DRL development challenges into five groups including DRL issues, parallel processing & multi-threading, DRL libraries/frameworks, DL issues, and general programming. An analysis of the response received by posts reporting about the challenges shows that DRL comprehension, DRL libraries/frameworks API usage, and designing a problem using DRL algorithms are the most challenging parts of DRL applications' development. Furthermore, parallel processing/multi-threading and DRL libraries/frameworks challenges required a longer time before receiving an accepted answer. We proposed a taxonomy of challenges and validated it using a survey of 59 DRL developers. The developers confirmed the frequency, severity, and required effort to address identified challenges. We hope that the results reported in this paper will stimulate the development of DRL quality assurance tools and guide the research community toward solving the identified challenges.

8 Data Availability Statement

The dataset generated during the current study is available in the replication package, which is accessible via [38].

References

1. Alshangiti, M., Sapkota, H., Murukannaiah, P.K., Liu, X., Yu, Q.: 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), pp. 1–11. IEEE (2019)
2. Aradi, S.: Survey of deep reinforcement learning for motion planning of autonomous vehicles. IEEE Transactions on Intelligent Transportation Systems **23**(2), 740–759 (2020)
3. Arpteg, A., Brinne, B., Crnkovic-Friis, L., Bosch, J.: Software engineering challenges of deep learning. In: 2018 44th euromicro conference on software engineering and advanced applications (SEAA), pp. 50–59. IEEE (2018)
4. Arulkumaran, K., Deisenroth, M.P., Brundage, M., Bharath, A.A.: Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine **34**(6), 26–38 (2017)
5. Ayman, A., Aziz, A., Alipour, A., Laszka, A.: Smart contract development in practice: trends, issues, and discussions on stack overflow. arXiv preprint arXiv:1905.08833 (2019)
6. Bagherzadeh, M., Kahani, N., Briand, L.: Reinforcement learning for test case prioritization. IEEE Transactions on Software Engineering **48**(8), 2836–2856 (2021)
7. Bangash, A.A., Sahar, H., Chowdhury, S., Wong, A.W., Hindle, A., Ali, K.: What do developers know about machine learning: a study of ml discussions on stackoverflow. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), pp. 260–264. IEEE (2019)
8. Beyer, S., Macho, C., Di Penta, M., Pinzger, M.: What kind of questions do developers ask on stack overflow? a comparison of automated approaches to classify posts into question categories. Empirical Software Engineering **25**, 2258–2301 (2020)
9. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)
10. Bruer, J.J., Tropp, J.A., Cevher, V., Becker, S.R.: Designing statistical estimators that balance sample size, risk, and computational cost. IEEE Journal of Selected Topics in Signal Processing **9**(4), 612–624 (2015)
11. Cao, Y., Chen, L., Ma, W., Li, Y., Zhou, Y., Wang, L.: Towards better dependency management: A first look at dependency smells in python projects. IEEE Transactions on Software Engineering (2022)
12. Castro, P.S., Moitra, S., Gelada, C., Kumar, S., Bellemare, M.G.: Dopamine: A research framework for deep reinforcement learning. arXiv preprint arXiv:1812.06110 (2018)
13. Decan, A., Mens, T., Grosjean, P.: An empirical comparison of dependency network evolution in seven software packaging ecosystems. Empirical Software Engineering **24**, 381–416 (2019)
14. Frost, J.: Introduction to Statistics: An Intuitive Guide for Analyzing Data and (2019)

15. Garg, D., Goel, P., Kandaswamy, G., Ganatra, A., Kotecha, K.: A roadmap to deep learning: a state-of-the-art step towards machine learning. In: Advanced Informatics for Computing Research: Second International Conference, ICAICR 2018, Shimla, India, July 14–15, 2018, Revised Selected Papers, Part I 2, pp. 160–170. Springer (2019)
16. Gibbons, J.D.: Nonparametric measures of association. 91. Sage (1993)
17. Google: How to use google sheets. https://support.google.com/docs/answer/6000292?hl=en&co=GENIE.Platform%3DDesktop (2020). Accessed: 2023-02-01
18. Hamidi, A., Antoniol, G., Khomh, F., Di Penta, M., Hamidi, M.: Towards understanding developers' machine-learning challenges: A multi-language study on stack overflow. In: 2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM), pp. 58–69. IEEE (2021)
19. Han, J., Deng, S., Lo, D., Zhi, C., Yin, J., Xia, X.: An empirical study of the dependency networks of deep learning libraries. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 868–878. IEEE (2020)
20. Haque, M.U., Iwaya, L.H., Babar, M.A.: Challenges in docker development: A large-scale study using stack overflow. In: Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–11 (2020)
21. He, J., Xin, Z., Xu, B., Zhang, T., Kim, K., Yang, Z., Thung, F., Irsan, I., Lo, D.: Representation learning for stack overflow posts: How far are we? arXiv preprint arXiv:2303.06853 (2023)
22. Heo, K., Lee, W., Pashakhanloo, P., Naik, M.: Effective program debloating via reinforcement learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, pp. 380–394 (2018)
23. Huang, K., Chen, B., Wu, S., Cao, J., Ma, L., Peng, X.: Demystifying dependency bugs in deep learning stack. arXiv preprint arXiv:2207.10347 (2022)
24. Humbatova, N., Jahangirova, G., Bavota, G., Riccio, V., Stocco, A., Tonella, P.: Taxonomy of real faults in deep learning systems. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 1110–1121 (2020)
25. Islam, M.J., Nguyen, G., Pan, R., Rajan, H.: A comprehensive study on deep learning bug characteristics. In: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 510–520 (2019)
26. Islam, M.J., Pan, R., Nguyen, G., Rajan, H.: Repairing deep neural networks: Fix patterns and challenges. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, ICSE '20, p. 1135–1146. Association for Computing Machinery, New York, NY, USA (2020). DOI 10.1145/3377811.3380378. URL https://doi.org/10.1145/3377811.3380378

27. Jelodar, H., Wang, Y., Yuan, C., Feng, X., Jiang, X., Li, Y., Zhao, L.: Latent dirichlet allocation (lda) and topic modeling: models, applications, a survey. Multimedia Tools and Applications **78**, 15169–15211 (2019)
28. Kuhnle, A., Schaarschmidt, M., Fricke, K.: Tensorforce: a tensorflow library for applied reinforcement learning. Web page (2017). URL `https://github.com/tensorforce/tensorforce`
29. Li, Y.: Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274 (2017)
30. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., Stoica, I.: Rllib: Abstractions for distributed reinforcement learning. In: International Conference on Machine Learning, pp. 3053–3062. PMLR (2018)
31. Lorenz, U.: Reinforcement Learning From Scratch: Understanding Current Approaches - with Examples in Java and Greenfoot. Springer International Publishing (2022). URL `https://books.google.ca/books?id=bV2YEAAAQBAJ`
32. Lune, H., Berg, B.L.: Qualitative research methods for the social sciences. Pearson (2017)
33. Lwakatare, L.E., Raj, A., Bosch, J., Olsson, H.H., Crnkovic, I.: A taxonomy of software engineering challenges for machine learning systems: An empirical investigation. In: Agile Processes in Software Engineering and Extreme Programming: 20th International Conference, XP 2019, Montréal, QC, Canada, May 21–25, 2019, Proceedings 20, pp. 227–243. Springer International Publishing (2019)
34. McDonald, N., Schoenebeck, S., Forte, A.: Reliability and inter-rater reliability in qualitative research: Norms and guidelines for cscw and hci practice. Proceedings of the ACM on human-computer interaction **3**(CSCW), 1–23 (2019)
35. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. nature **518**(7540), 529–533 (2015)
36. Mojica-Hanke, A., Bayona, A., Linares-Vásquez, M., Herbold, S., González, F.A.: What are the machine learning best practices reported by practitioners on stack exchange? arXiv preprint arXiv:2301.10516 (2023)
37. Morovati, M.M., Nikanjam, A., Khomh, F., Jiang, Z.M.: Bugs in machine learning-based systems: a faultload benchmark. Empirical Software Engineering **28**(3), 62 (2023)
38. Morovati, M.M., Tambon, F., Taraghi, M., Nikanjam, A., Khomh, F.: Paper replication package. `https://github.com/drlchallenges/drl_challenges` (2023). Accessed: 2023-02-01
39. Nguyen, H.A., Nguyen, T.T., Wilson Jr, G., Nguyen, A.T., Kim, M., Nguyen, T.N.: A graph-based approach to api usage adaptation. ACM Sigplan Notices **45**(10), 302–321 (2010)
40. Nikanjam, A., Braiek, H.B., Morovati, M.M., Khomh, F.: Automatic fault detection for deep learning programs using graph transformations. ACM

Transactions on Software Engineering and Methodology (TOSEM) **31**(1), 1–27 (2021)
41. Nikanjam, A., Morovati, M.M., Khomh, F., Ben Braiek, H.: Faults in deep reinforcement learning programs: a taxonomy and a detection approach. Automated Software Engineering **29**(1), 1–32 (2022)
42. Openja, M., Majidi, F., Khomh, F., Chembakottu, B., Li, H.: Studying the practices of deploying machine learning projects on docker. In: Proceedings of the 26th International Conference on Evaluation and Assessment in Software Engineering, pp. 190–200 (2022)
43. Peruma, A., Simmons, S., AlOmar, E.A., Newman, C.D., Mkaouer, M.W., Ouni, A.: How do i refactor this? an empirical study on refactoring trends and topics in stack overflow. Empirical Software Engineering **27**(1), 11 (2022)
44. Plappert, M.: keras-rl. `https://github.com/keras-rl/keras-rl` (2016)
45. Pumperla, M., Oakes, E., Liaw, R.: Learning Ray: Flexible Distributed Python for Machine Learning. O'Reilly Media (2023). URL `https://books.google.ca/books?id=vKj0zgEACAAJ`
46. Qualtrics: Qualtrics official website. `https://www.qualtrics.com/` (2023). Accessed: 2023-04-01
47. Rao, Q., Frtunikj, J.: Deep learning for self-driving cars: Chances and challenges. In: Proceedings of the 1st international workshop on software engineering for AI in autonomous systems, pp. 35–38 (2018)
48. Reda, D., Tao, T., van de Panne, M.: Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In: Proceedings of the 13th ACM SIGGRAPH Conference on Motion, Interaction and Games, pp. 1–10 (2020)
49. Schaarschmidt, M., Kuhnle, A., Ellis, B., Fricke, K., Gessert, F., Yoneki, E.: LIFT: Reinforcement learning in computer systems by learning from demonstrations. CoRR **abs/1808.07903** (2018). URL `http://arxiv.org/abs/1808.07903`
50. Schoop, E., Huang, F., Hartmann, B.: Umlaut: Debugging deep learning programs using program structure and model behavior. In: Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems, pp. 1–16 (2021)
51. de Souza Nascimento, E., Ahmed, I., Oliveira, E., Palheta, M.P., Steinmacher, I., Conte, T.: Understanding development process of machine learning systems: Challenges and solutions. In: 2019 acm/ieee international symposium on empirical software engineering and measurement (esem), pp. 1–6. IEEE (2019)
52. StackExchange: Stack exchange data dump. `https://archive.org/details/stackexchange` (2022). Accessed: 2023-02-01
53. Tambon, F., Majdinasab, V., Nikanjam, A., Khomh, F., Antonio, G.: Mutation testing of deep reinforcement learning based on real faults. arXiv preprint arXiv:2301.05651 (2023)
54. Vijayaraghavan, G., Kaner, C.: Bug taxonomies: Use them to generate better tests. Star East **2003**, 1–40 (2003)

55. Wan, Y., Zhao, Z., Yang, M., Xu, G., Ying, H., Wu, J., Yu, P.S.: Improving automatic source code summarization via deep reinforcement learning. In: Proceedings of the 33rd ACM/IEEE international conference on automated software engineering, pp. 397–407 (2018)
56. Xiao, Y., Liu, J., Wu, J., Ansari, N.: Leveraging deep reinforcement learning for traffic engineering: A survey. IEEE Communications Surveys & Tutorials **23**(4), 2064–2097 (2021)
57. Xie, D., Li, Y., Kim, M., Pham, H.V., Tan, L., Zhang, X., Godfrey, M.W.: Docter: documentation-guided fuzzing for testing deep learning api functions. In: Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 176–188 (2022)
58. Yahmed, A.H., Abbassi, A.A., Nikanjam, A., Li, H., Khomh, F.: Deploying deep reinforcement learning systems: A taxonomy of challenges. arXiv preprint arXiv:2308.12438 (2023)
59. Yu, C., Liu, J., Nemati, S., Yin, G.: Reinforcement learning in healthcare: A survey. ACM Computing Surveys (CSUR) **55**(1), 1–36 (2021)
60. Zahedi, M., Rajapakse, R.N., Babar, M.A.: Mining questions asked about continuous software engineering: A case study of stack overflow. In: Proceedings of the evaluation and assessment in software engineering, pp. 41–50 (2020)
61. Zhang, T., Gao, C., Ma, L., Lyu, M., Kim, M.: An empirical study of common challenges in developing deep learning applications. In: 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), pp. 104–115. IEEE (2019)
62. Zhang, X., Zhai, J., Ma, S., Shen, C.: Autotrainer: An automatic dnn training problem detection and repair system. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 359–371. IEEE (2021)
63. Zhu, W., Zhang, H., Hassan, A.E., Godfrey, M.W.: An empirical study of question discussions on stack overflow. Empirical Software Engineering **27**(6), 1–25 (2022)
64. Zolfagharian, A., Abdellatif, M., Briand, L., Bagherzadeh, M., et al.: Search-based testing approach for deep reinforcement learning agents. arXiv preprint arXiv:2206.07813 (2022)