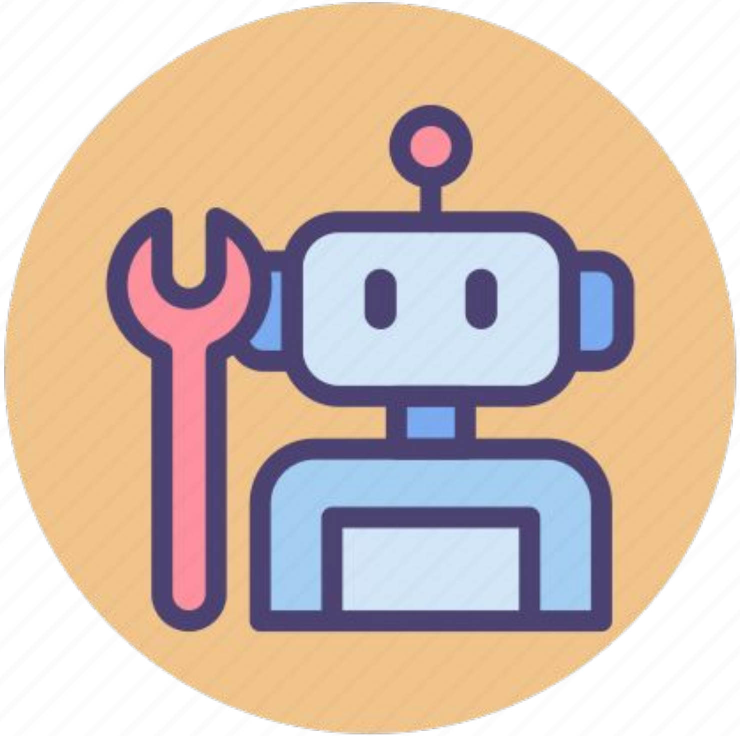
# Faults in Deep Reinforcement Learning Programs A Taxonomy and A Detection Approach 2021

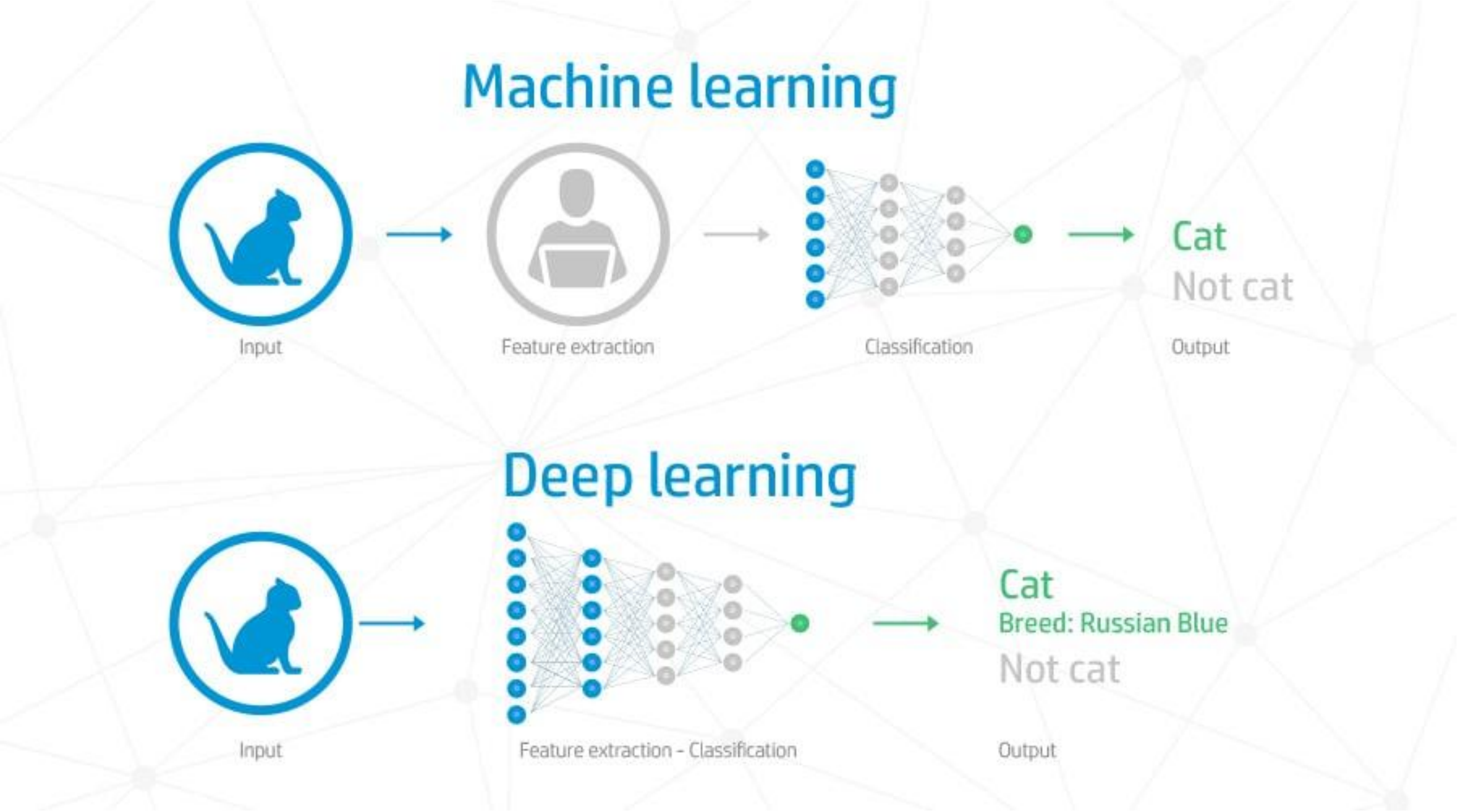
Presented by : Mayuresh Nene, Prasad Chavan, Ryan Chui

# Agenda

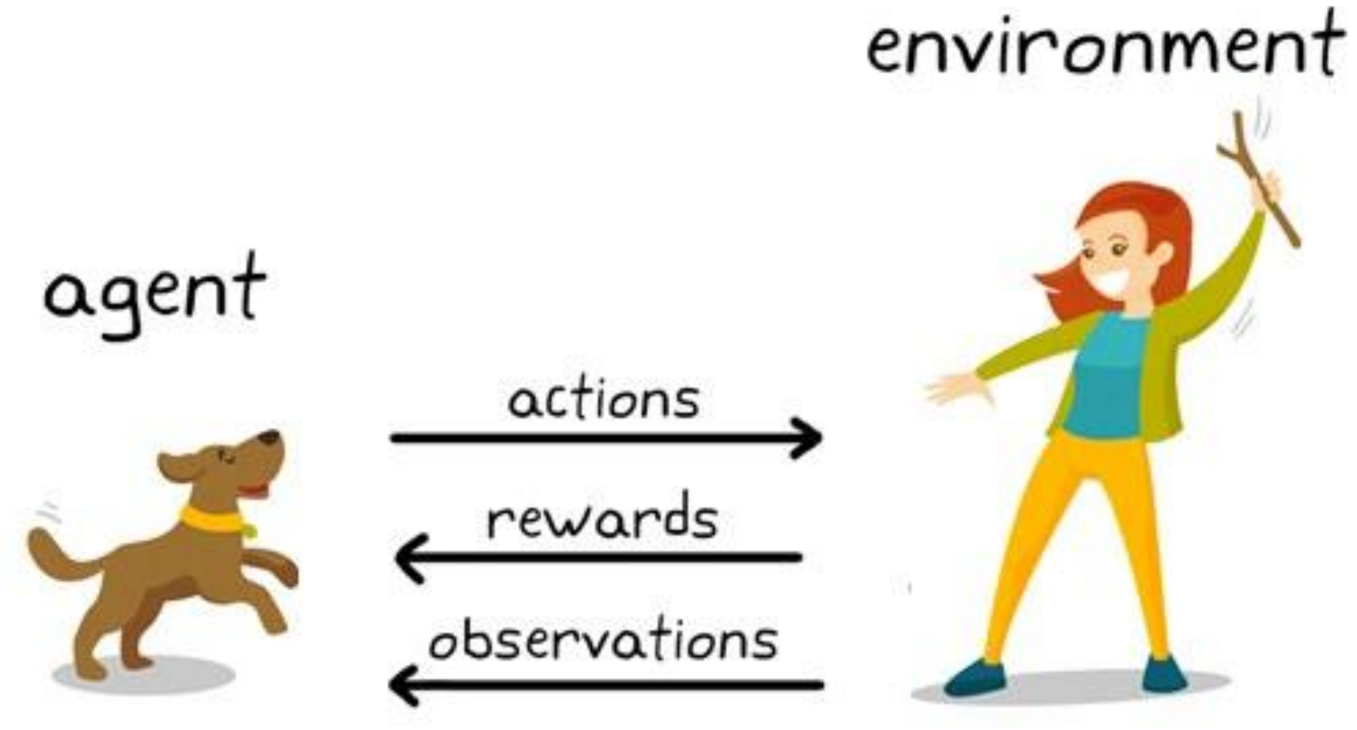
- Introduction
- Key definitions
- Problem
- Solution
- Meta Model for DQN
- Experimental Design and Result
- Research Validation



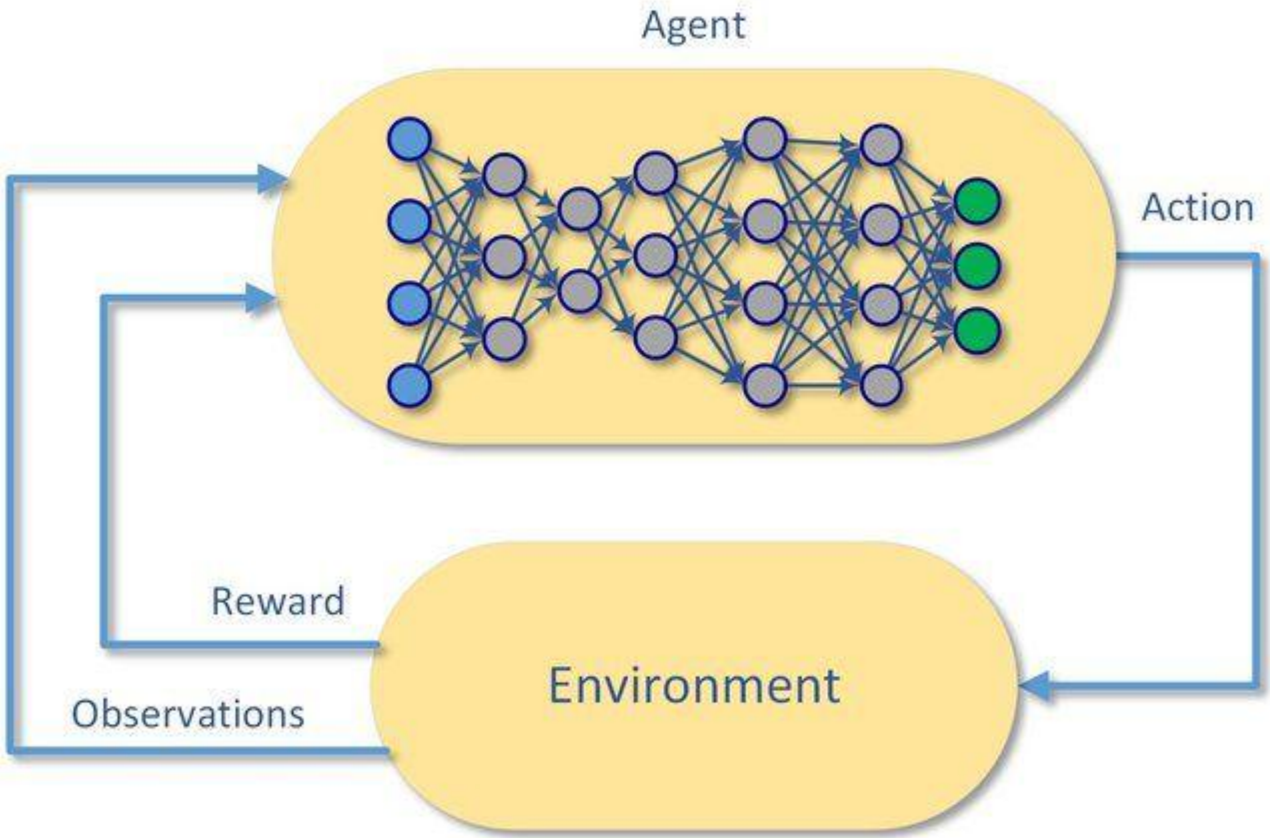
# What is Deep Learning?



## What about Reinforcement Learning?



Deep Reinforcement Learning (DRL) Systems



# Deep Reinforcement Learning (DRL) Systems

- Deep Reinforcement Learning tries to solve problems that require dynamic sequential decision making.

- Agent Exploration/Exploitation tradeoff balancing.

  - Decide whether to go for decision with known high yield or to explore a new decision which may or may not have a higher yield.

  - They usually collect data with a stochastic policy.

- Idea to promote exploration is giving the agent a motive to explore unknown outcomes.

  - Generally done by incentivising exploration by modifying the loss function.

# Applications of Deep Reinforcement Learning

- Automobile industry :
  - Autonomous Cars
  - Intelligent Braking Systems
- Healthcare :
  - Automated Diagnosis
  - Chronic disease treatments
- Robotics :
  - Manufacturing (Assembly lines)
  - Combat Training

# Why go for DRL and not RL Systems?

- Example of a video game :

    - A reinforcement learning model can keep track of all the (state, action) pairs.
    - Maintaining all these pairs is possible in case of a 2D game such as Pacman.
    - In case of bigger games, even a slightly changed state is still a distinct state. It becomes infeasible for an RL to keep track of all (state, action) pairs.
    - You could use something that can generalize the knowledge instead of *storing* and *looking up* every little distinct state.
    - This is where a DL neural network comes into the picture which can predict the reward for an input (state, action) pair or or pick the best action given a state, however you like to look at it.

# Faults in DRL Systems

- Cartpole was stuck at a suboptimal reward level without further improvements.

- Missing random actions implementation.

- Agent fails to perform random actions to gather information from the environment.

```
class DQNAgent:
    def __init__(self, state_size, action_size):
        #initialization
    def _build_model(self):
        #define DL model
    def remember(self, state, action, reward, next_state, done):
        #define replay buffer
    def act(self, state, sess):
        act_values = sess.run(self.model[3], feed_dict = { self.model[1]: state})
        return np.argmax(act_values[0])
    def replay(self, batch_size, sess):
        #replaying samples from buffer
if __name__ == "__main__":
    # setting up the environment
    agent = DQNAgent(env.observation_space.shape[0], env.action_space.n)
    for e in range(episodes):
        for time_t in range(500):
            #interacting with the environment
            action = agent.act(state, sess)
            next_state, reward, done, _ = env.step(action)
```

1

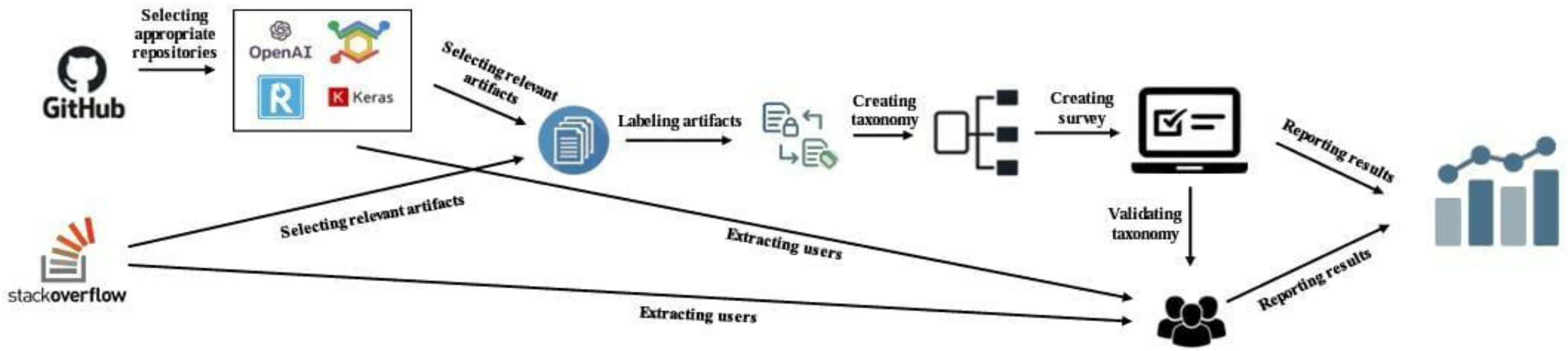
```
def act(self, state, sess, episode):
    if random.random() < math.pow(2, -episode / 30):
        return env.action_space.sample()
    act_values = sess.run(self.model[3], feed_dict = { self.model[1]: state})
    return np.argmax(act_values[0])
```

2

# Mining the required data

| *Project Name* | *stars* | *commits* | *issues* | *contributors* |
|---|---|---|---|---|
| OpenAI Gym | $22k$ | 1,217 | 1,179 | 248 |
| Dopamine | $9.1k$ | 197 | 118 | 8 |
| keras-rl | $4.8k$ | 308 | 214 | 39 |
| Tensorforce | $2.7k$ | 1,979 | 512 | 60 |

# Methodology of Obtaining the Taxonomy




Steps to build the taxonomy

- Manual Analysis of the DRL Programs
- Building and Validating the taxonomy

# Manual Analysis

Data was mined from Github and Stack Overflow posts

Stack Overflow:

Yielded 2072 posts

After filtration: 329 posts

Github:

Extracted all issues from the 4 libraries

Filtered by label as ‘closed’

| OpenAI Gym | Keras-RL | TensorForce | Dopamine |
|---|---|---|---|
| 151 | 200 | 300 | 110 |

# Manual Analysis

Manual labeling was performed

Criteria to reject a artifact from analysis:

- Not related to the bug fixing activity
- Related to an issue with the framework itself
- Common errors
- Root cause wasn’t clear for the authors

# Building Taxonomy

Bottom up approach used:

Labels → Categories

Double check each category

Subcategories → Parent Categories

Explore all categories, subcategories and leaf nodes

Finalize the taxonomy

# Validating Taxonomy

Survey involving DRL practitioners was used to validate
The practitioners were selected from Github and Stack Overflow

A total of 210 practitioners were selected

- 140 from Github
- 40 from Stack Overflow

19 practitioners responded to the survey

- 8 researchers
- 11 developers

Experience of the practitioners (in years):

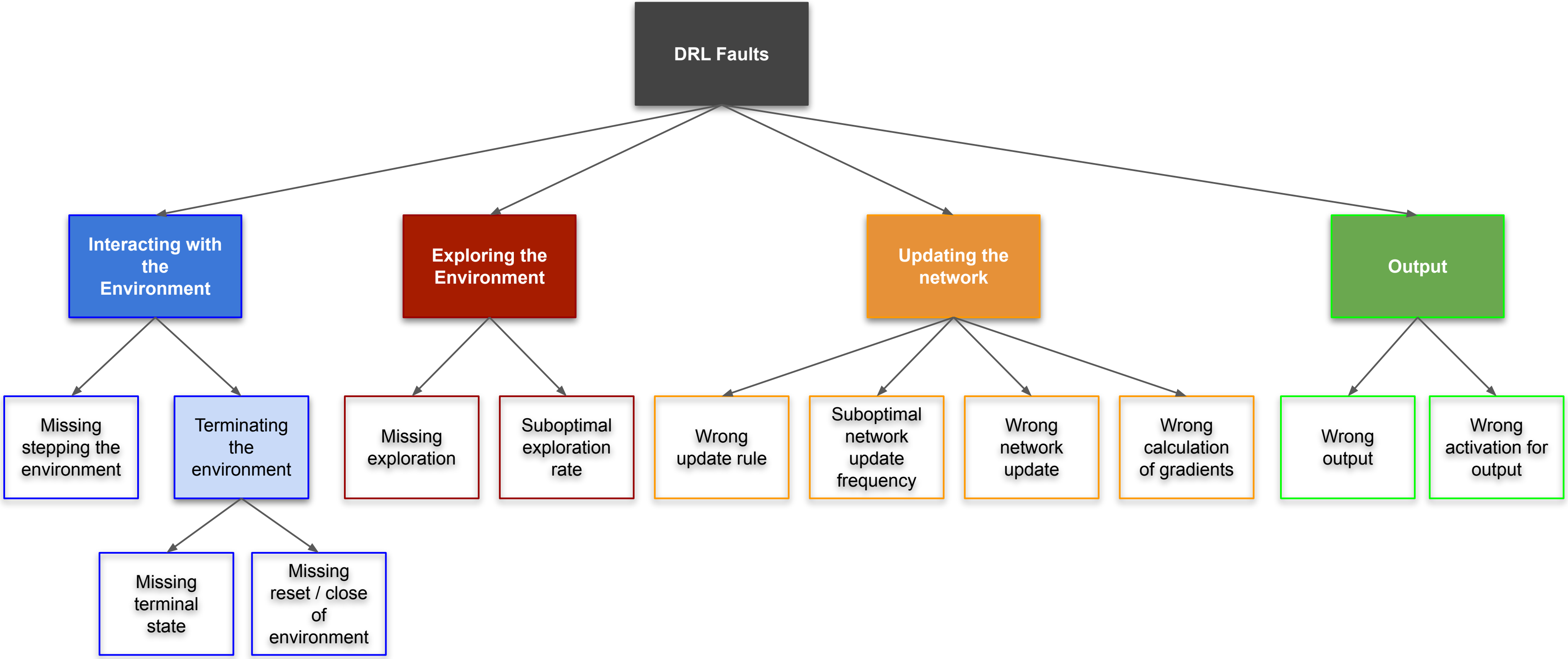
| | ML & DL | DRL |
|---|---|---|
| Least | 1 to 3 | Less than 1 |
| Median | 3 to 5 | 1 to 3 |
| Most | 5 + | 5+ |

# DRL Faults

5 Main Categories

- Model
    - Model type and properties
    - Layers
- Tensors and Inputs
    - Wrong tensor shape
    - Wrong input
- GPU Usage
- Training
    - Hyperparameter selection fault
    - Loss function and Optimizer faults
- Application Programming Interface

# Taxonomy Obtained



# Interacting with the Environment

Type 1: Missing stepping the environment:

- Failure to move the environment to a new state

Type 2: Missing terminal state:

- Wrong detection of the terminal state
- Completely missing the terminal state

Type 3: Missing reset / close environment:

- Bad termination problems

Faults while terminating the environment

# Exploring the Environment

Type 4: Missing exploration:
- Failure to explore the environment while in a new state

Type 5: Suboptimal exploration rate:
- Problems related to exploration parameters
- For example the Epsilon in Epsilon greedy method

# Updating Network

Type 6: Wrong update rule:

- Incorrect update rule for a value or policy function

Type 7: Suboptimal network update frequency:

- Network frequency update parameters cause issues if not properly calibrated

Type 8: Wrong network update:

- Wrong update of the parameters of the network
- Wrong update of the network itself

Type 9: Wrong calculation of gradients:

- Gradients of learning

# Output

Type 10: Wrong output:
- Failure to define a correct output layer

Type 11: Wrong activation:
- Failure to define a correct activation function for output

# Validating Results

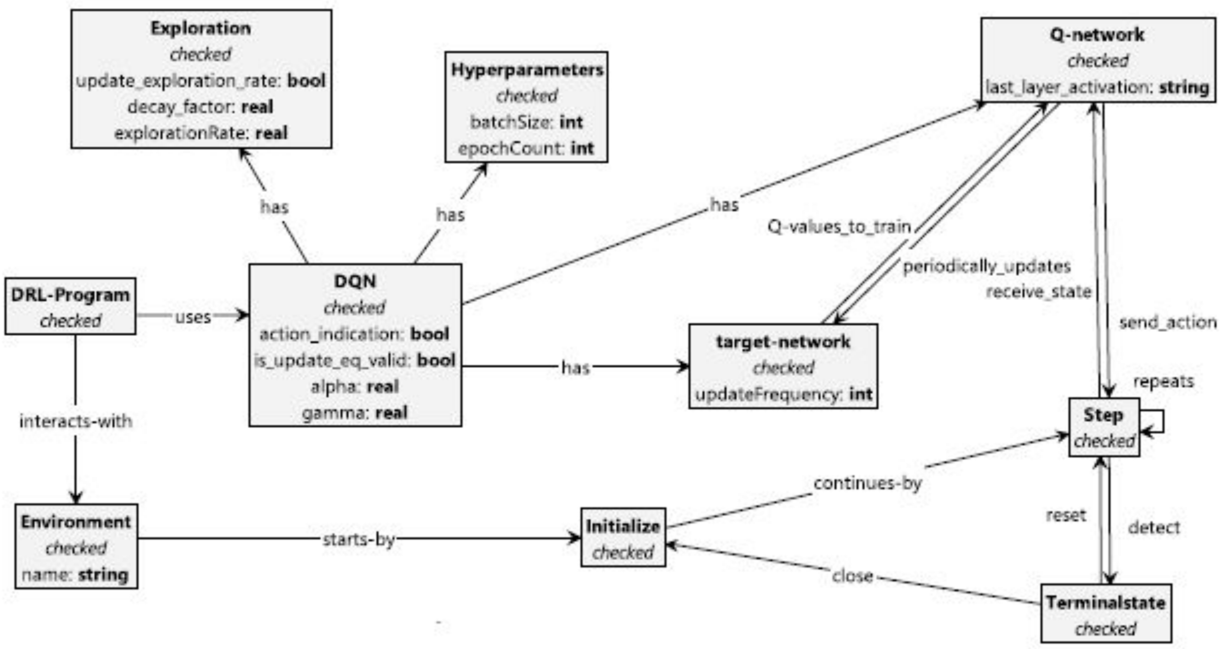
| Faults Type | Responses | | | | |
|---|---|---|---|---|---|
| | No | Yes, minor and easy | Yes, minor but hard | Yes, major but easy | Yes, major and hard |
| Type 1 | 63% | 16% | 5% | 11% | 5% |
| Type 2 | 42% | 16% | 16% | 21% | 5% |
| Type 3 | 37% | 53% | 0% | 11% | 0% |
| Type 4 | 11% | 5% | 5% | 16% | 63% |
| Type 5 | 11% | 16% | 16% | 16% | 42% |
| Type 6 | 11% | 21% | 11% | 26% | 32% |
| Type 7 | 16% | 0% | 26% | 21% | 37% |
| Type 8 | 16% | 0% | 26% | 21% | 37% |
| Type 9 | 32% | 32% | 5% | 16% | 16% |
| Type 10 | 53% | 32% | 0% | 11% | 5% |
| Type 11 | 42% | 32% | 5% | 21% | 0% |

# Meta Model

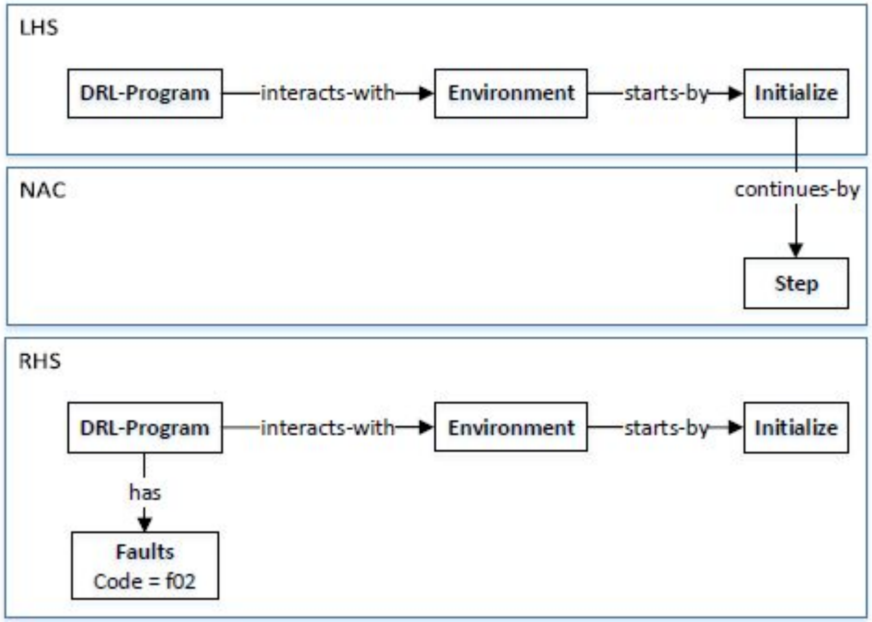
Let's consider the meta model for a DQN

- Environment: Variables for number of actions and number of states

- Deep Q-Network: The decision making component



# Detect Faults by Graph Transformation




Two ways to build a DRL Program

1) Configure an arbitrary model directly.
2) Transform a DRL program to a model.

- LHS shows DRL-program with its initialized Environment.
- The fault is detected if there is not a Step node just after Initialize.
- NAC forbids the existence of Step right after Initialize. Thus, if the fault is detected, RHS adds a Faults node with relevant fault code to the DRL-program.

# Implementation

**Algorithm 1:** *DRLinter*: Model-based Fault Detection in DRL Programs by Graph Transformations

**Input:** A DRL program, *program*, and *rules* as graph transformations
**Output:** List of detected bugs of the program
$graph \leftarrow$ convertDRLProgram($program$)
$graph \leftarrow$ graphChecker($graph$, $rules$) :
 1. starting by $graph$, apply enables rules.
 2. terminate when there is no applicable rule.
 3. **return** $graph$.
$report \leftarrow$ extractReport($graph$)
**return** $report$

- In ConvertDRLProgram step, the source code is parsed to extract relevant information in order to build the model.
- Once the DL source code is modeled as a graph, by calling graphChecker, the detection rules can be used to execute the sequence of graph transformations on the model.
- Current version of DRLinter are developed on OpenAI Gym and TensorFlow libraries in order for synthetic DRL programs to work.
- By calling extractReport, a report will be extracted from the output of graphChecker.

# Experimental Design

1) Need some buggy DRL codes that contain the types of faults covered in DRLinter.
2) Evaluate DRLinter using some synthetic faulty DRL programs that are created by reproducing real DRL faults.
3) Use StackOverflow and Github as a platform for DRL faulty program to construct taxonomy to synthesize buggy examples.
   a) Step 1: Run a DRL programing using OpenAI Gym or Tensorflow.
   b) Step 2: Injected the fault type to the code.
   c) Step 3: If observed pattern of faults is found, they will be used to reproduce synthetic faulty samples for future use. Note that at least one faulty example will be executed at least once during detection rule process.

# Experimental Sample Result

| No. | SO#(link) | Symptom | Recommended Fix | Fault Type |
|---|---|---|---|---|
| 1 | 57106676 | Unstable learning, increasing loss | Increase the update frequency of the target network | Type 7 |
| 2 | 56964657 | Unstable learning, increasing loss | Increase the update frequency of the target network | Type 7 |
| 3 | 47750291 | Bad performance | Use an exploration mechanism | Type 4 |
| 4 | 54385568 | Bad performance | Decrease the exploration rate | Type 5 |
| 5 | 51425688 | Bad performance | Decrease the exploration rate | Type 5 |
| 6 | 49035549 | Bad performance | Decrease the exploration rate, Improve DNN design | Type 5 |
| 7 | 50308750 | Compile-time error | Add proper API to close environment | Type 3 |
| 8 | 47643678 | Bad performance | Detect the terminal state properly | Type 2 |
| 9 | 40896951 | Bad performance | Change the activation of the last layer | Type 11 |
| 10 | 37524472 | Bad performance | Change Q-learning update equation | Type 6 |
| 11 | link[1] | Compile-time error | Detect state and action correctly | Type 8 |

[1]https://github.com/tensorforce/tensorforce/issues/697.

- DL interface can detect the bugs in all 15 synthetic examples, but failed to detect all existing faults in the programs.

# Research Validation

- Two ways to validate taxonomy of real faults.

  a) Manual Analysis of Github artifacts and StackOverflow posts.

  b) Conducted survey with developers/ ML researchers to verify completeness and usefulness of identified faulty type categories.

- Pros and Cons discussion.

# ANY QUESTIONS ??