Received August 8, 2020, accepted August 17, 2020, date of publication August 25, 2020, date of current version September 10, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3019278

Reversely Discovering and Modifying Properties Based on Active Deep Q-Learning

LEI YU, (Member, IEEE), AND ZHIFA HUO
Department of Computer Science, Inner Mongolia University, Hohhot 010021, China

Corresponding author: Lei Yu (yuleiimu@sohu.com)

This work was supported by the National Natural Science Foundation of China under Grant NSFC 61962040.

ABSTRACT Many researchers studied DQN (Deep Q-Networks) to train a game AI to beat human players, while we trained an improved AI to reversely modify properties of 3D video games. Our ultimate objective is to improve automatic debug for software and cloud services. However, the problem that reversely discovers properties in online 3D Video Games in an automatic way has not been studied yet. Therefore, related special difficulties are first discussed in the paper. RMDQN (a Reverse Method based on our active Deep Q-Networks) is proposed to deal with the problem, and an active DQN is invented to make the reverse procedure automatic and intelligent. The action engine of RMDQN is able to control any operational game object like a player is playing, which makes automatic debug possible. A video demonstration is provided to show the result of reversely modifying game properties by our method. It was proved that our method can improve debug technology in 3D video games, and it will be applied in cloud services with few modifications.

INDEX TERMS Online 3D video games, reverse engineering, deep Q-Networks.

I. INTRODUCTION

Nowadays, software engineers use semi-automatic tools for reverse engineering, including Ollydbg, Windbg, and Cheat Engine. The traditional and general reverse procedure in Windows operation system is shown as follows: engineers use a debugger to attach the debuggee process. According to the engineer experiences, recursively find suitable places to set hardware breakpoints or software breakpoints for monitoring computer Memory data access or for stopping code running. Before target values or target functions are found, engineers need to filter out many false addresses. Finally, engineers modify target values or target functions, and sometimes set variables and generate patches for repeating usage.

It still needs huge manual operations to manipulate debugging tools and needs large brainwork to analyze computer Memory data, register statues, stack traces, and assembly jumps, etc. It is such a double burden to analyze assembly code with human eyes and brains, especially for big software. This paper selects two typical games for demonstrations: a first-person shooting game (FPSG) and a real-time strategic game (RTSG).

Now video games are becoming larger, some require 30G-50G Hard disk space, and the code space is also extreme huge. For example, RTSG requires 35G Hard disk space. Properties in RTSG include: coordinates, munitions, fuels, human powers, hit points of solders and vehicles, sight ranges, damage ranges, and building time, etc. Properties in FPSG include: health, vectors of position, teams, bone matrix, glow, and view angles, etc. These properties can be reversely manipulated by locking Memory data or code injection. We used the two games for our experiments, because simulated environments have no match with real game environments in aspect of huge virtual addresses usages, real stack traces, and real function import tables, etc.

DeepMind proposed a method called Deep Q-Networks (DQN) to automatically play the Atari games. The key idea of basic DQN is to utilize deep neural networks to replace Q-network, and to train the Q-network to obtain Q values. Many researchers studied DQN (or Deep Q-Learning) to train a game AI to beat human players, while we trained an improved AI to reversely modify properties of 3D video games to "counter attack", in another word, to improve automatic debug, but there are several difficulties and challenges:

1. DQNs are often used in simulation environments or Atari games from the Arcade Learning Environment, so that an agent can easily interact with virtual ambient objects, and obtain reward immediately. However, DQNs need more supports in real software, because perceptions and actions

The associate editor coordinating the review of this manuscript and approving it for publication was Nilanjan Dey.

TABLE 1. Topics of reverse engineering.

Reverse Topics	Traditional reverse methods	New reverse methods
General theories	[1-2]	[15]
Specific applications	[3-5]	[16-18]
Codes to models	[6-7]	[19]
Data manipulation and detection	[8-10]	[20-22]
Multiplayer online game	[11]	
Logic analysis	[12-14]	

must be translated to specifications conforming to real software settings.

2. Generally, there are many same values in different virtual addresses of a video game in computer Memory, but only one virtual address stores the real property value, and it may be stored as a pointers link. Moreover, this virtual address may change after each starting of the video game.
3. Not only virtual addresses may change each time, but also the sequence of game functions may change, causing Memory stack traces change. In addition, Object-oriented programming introduces inherited classes and override functions. Their implementation in assembly codes (virtual table function) makes reverse even harder.

To tackle above difficulties and challenges, we proposed a reverse method for discovering and modifying properties in video games, called RMDQN (Reversely Discovering and modifying properties with active Deep Q-Learning). RMDQN includes two components: action engine and GDQN (Game DQN). Main contributions are shown as follows:

1 A new method (RMDQN) is proposed to reversely modify properties in 3D video games, and its reverse procedure is automatic and intelligent.

2 RMDQN action engine is created to deal with above challenge 1 in this paper. The action engine automatically uses a limited action space to control game objects and navigate in the game.

3 GDQN (A new attempt to combine active learning and improved DQN) is created to deal with challenge 2 and 3 in this paper. GDQN intelligently uses its Q-table and action engine to interact with game objects and observes corresponding changes. Finally, RMDQN filters out false addresses, and goes backwards to find out the base address of target value.

II. RELATED WORK

We categorized some typical related works about reverse engineering as below table shows. Unlike traditional reverse methods, most new reverse methods are based on deep learning.

Semi-automatic tools, including Ollydbg, Windbg, and Cheat Engine, are mature enough for reverse engineering. However, Ollydbg seems to be frozen because its main functions have not been updated since year 2014.

Academics continue updating their outcomes. Washizaki [1] confirmed that any reverse engineering activity can be clearly described as a pattern based on the framework from the viewpoint of program meta-models. Shu *et al.* [2] presented a control reconfiguration approach to improve the performance of two classes of dynamical systems by reverse-engineer. Poženel *et al.* [3] used clickstream data to enhance reverse engineering of Web applications. Miranda *et al.* [4] used reverse engineering techniques to infer a system use case model. It is possible to reversely generate a three-dimensional model from a set of 2D photographs [5].

Sabir *et al.* [6] implemented a novel framework named as "Source to Model Framework (Src2MoF)" is proposed to generate Unified Modeling Language structural (class) and behavioral (activity) diagrams from the Java source code. An open source transformation engine named "UML model generator" is implemented using Java, which takes these intermediate models as input, and produce high-level UML models of the subject legacy system. El Otmani *et al.* [7] proposed a process to migrate a Struts web applications into UML (Unified Modeling Language) model.

Basile *et al.* [8] presented a software protection meta-model that can be instantiated to construct a formal knowledge base to against reverse engineering. Applications of reverse engineering include data exploration, data security, relational classifier engineering, and the study of the expressiveness of query languages [9]. Martins [10] described strategies employed to leverage efficient example acquisition and query reverse engineering.

Tomičić *et al.* [11] presented a form of a network-level game cheating based on the MMORPG (Massively Multi-Player On-Line Role-Playing Games) client reverse engineering method within the game Mana World, an MMORPG with client/server architecture. They used the network protocol vulnerability of a non-encrypted data traffic in order to reverse-engineer the original data packets sent by the original game client, and replicated these packets in a AI-based game client which complete controls game character.

Botero *et al.* [12] surveyed challenges from two complementary perspectives: image processing and machine learning. These two fields of study form a firm basis for the enhancement of efficiency and accuracy of reverse engineering processes [13]. Fyrbiak *et al.* [14] presented a comprehensive reverse engineering and manipulation framework for gate-level netlists. It allows automating defensive design analysis (e.g., including arbitrary Trojan detection algorithms with minimal effort) as well as offensive reverse engineering and targeted logic insertion. Furthermore, they presented reverse engineering algorithms to disarm and trick cryptographic self-tests, and subtly leak cryptographic keys without any a priori knowledge of the design's internal workings.

Alam and Mukhopadhyay [15] provided an evaluation strategy for information leakages through DNN (Deep Neural Networks) by considering a case study on CNN classifier.

Qiu *et al.* [16] proposed a light-weight solution to automatically identify the Android malicious samples with high security and privacy impact. Nirumand *et al.* [17] presented a framework based on Model Driven Reverse Engineering.

In the proposed framework, some security-related information included in an Android app is automatically extracted and represented as a domain-specific model. Then, it is used for analyzing security configurations and identifying vulnerabilities in the corresponding application. The proposed framework automatically identifies the Intent Spoofing and Unauthorized Intent Receipt as two attacks related to the Android application communication model. Ray and Chowdhury [18] proposed the Reverse Engineering Technique (RET) to improve resource allocation accuracy. The paper used neural network based deep learning and Levenberg-Marquardt training algorithm for resource allocation prediction.

Gharibi *et al.* [19] automated the tasks of analyzing the Python source-code and extracting its structure; constructing static call graphs from the source code; and generating a similarity matrix of all possible execution paths in the system.

Miller *et al.* [20] proposed a purely unsupervised anomaly detector (AD), based on suitable (null hypothesis) density models for the different layers of a deep neural net and a novel decision statistic built upon the Kullback-Leibler divergence. Xiao *et al.* [21] proposed an approach of applying deep learning to solve the problem of data type identification in data segments. They defined 3 data types of data segment, then designed several data segment byte feature extraction methods to construct feature sequences, and finally presented a deep learning-based approach with feature sequences as input to recognize the data type byte by byte. Marchetti and Stabili [22] proposed a method for the automatic Reverse Engineering of Automotive Data frames. It has been designed to analyze traffic traces containing unknown CAN (controller area network) bus messages in order to automatically identify and label different types of data.

Above related works deal with parts of problem of reversely discovering and modifying properties (challenge 2, 3), but they cannot deal with our challenge 1. Some researchers [23]–[25] proposed scheduling and migration schemes in cloud using deep Q-learning. Based on deep Q-learning, researchers [26], [27] proposed algorithms to help the agent formulate a more useful strategy when playing video games. Chen *et al.* [28] proposed an algorithm that dynamically estimates the uncertainty of recent states, and utilizes the queried demonstration data by optimizing a supervised loss, in addition to the usual DQN loss [29]. Todd [30] presented an algorithm, Deep Q-learning from Demonstrations (DQfD), which leverages small sets of demonstration data to accelerate the learning process and is able to automatically assess the necessary ratio of demonstration data while learning thanks to a prioritized replay mechanism.

A core dilemma in learning [31] is the exploration-exploitation problem and long-term credit assignment problem: in video games, it's challenging to extensively explore to obtain good performance, and to match the consequences of an agents' actions to the rewards it receives.

Double DQN [32], dueling architecture [33] and prioritized experience replay are able to enhance learning efficiency and stability, which allow agents to effectively use their experiences. Next, distributed DQN [34] was introduced to be simultaneously run on many computers, which allow agents to learn from experiences more quickly. These agents interact with the same environment, updating data to a central memory. Another type of agents then samples the data from this central memory.

The role of memory is to aggregate information from previous observations (which is usually partial) and present observations (which can reveal more information). To take into account previous observations and present observations into decision making, agents use memory to improve the decision making. Recurrent neural networks such as Long-Short Term Memory (LSTM) are used as short term memories in deep reinforcement learning. Recurrent Replay Distributed DQN (R2D2) [35] combined off-policy learning with distributed training and neural network model of short-term memory. Never Give Up (NGU) [36] was designed to modify R2D2 with another form of memory: episodic memory, which enables agents explore newer parts of the game.

Relying on undirected random actions to discover unseen states, techniques of epsilon-greedy require large time to explore state-action spaces, thus they do not scale well to hard exploration problems. Many directed exploration strategies have been proposed to overcome this limitation. Intrinsic motivation rewards was developed for seeking novelty over long time scales (agents use an undirected exploration strategy when all states are familiar) and short time scales [37]. Recently, episodic memory and meta-controller (allows the agent to choose a trade-off between near or long term performance, as well as exploring or exploiting) [38] have been used to speed up learning. However, all related works cannot completely deal with all challenges in our environment.

III. FEATURES OF 3D VIDEO GAMES FROM REVERSE PERSPECTIVE

In this section, we take the video game RTSG as a scenario, and show how features of 3D video games and compilers affect reverse procedures.

A. SCENARIO FOR ACCESSING A GAME PROPERTY

If a coordinate (often refers to X, Y, Z, but assuming only X in this example) of a soldier is placed at Memory address 0×00000018 by game program, the relevant disassembly code is like this:

TABLE 2. Disassembly code.

```
mov eax, [0x00001234]
mov ebx, [eax + 4]
add [ebx + 8], 9
```

The game program first visits 0×00001234 to get a value, and takes this value +4 as an address. Next, it visits this address to get a value (0×00000010), and adds it to 8 to get the target address. Finally, the program adds 9 to the content of the target address, that is, coordinate increases 9.

TABLE 3. Part of import table.

DLLs	Functions
steam_api.dll	steam_api.SteamUtils
	steam_api.SteamRemoteStorage
KERNEL32.dll	kernel32.VirtualAlloc
	kernel32.LoadLibraryA
	kernel32.HeapFree
	ntdll.RtlDeleteCriticalSection
	ntdll.RtlInitializeCriticalSection
	ntdll.RtlLeaveCriticalSection
	ntdll.RtlEnterCriticalSection
	ntdll.RtlEncodePointer
	ntdll.RtlDecodePointer
	kernel32.IsDebuggerPresent
GDI32.dll	GDI32.CreateBitmap
	GDI32.CreateCompatibleDC
dxgi.dll	dxgi.CreateDXGIFactory1
d3d11.dll	d3d11.D3D11CreateDevice

The game data structure is like this: 0×00001234 is the address of a global pointer (address is fixed). This pointer points to the data structure of a game battle, meaning that the content of pointer 0×00001234 is the address of the game battle. The +4 pointer points to a player's data structure, which points to the data structure of a player. The soldier built by the player is stored in +8 places inside the data structure of the player. Therefore, the coordinate address of the soldier is $[[0 \times 00001234] +4] +8$.

Reverse engineering for above assembly code is the process of finding ebx, eax, and 0×00001234 in a reverse order of game programs. However, the reality is not neat like above scenario.

Not only the reality is cluttered, but also it is multiple. Besides many value caches of different types (arguments and stack residuals, etc.), engineers often find multiple same values of one coordinate, because 3D video games have rendering pipelines.

B. FEATURES OF 3D VIDEO GAMES AND COMPILERS

Ray tracing, Global illuminate, Shader, and Normal Mapping may use coordinates of an object, especially the newest DirectX 12 intensively uses real-time Ray tracing. The following table shows the parts of import table of RTSG.

Although sometimes rasterization in render pipeline can be computed by GPU and stored in graphics Memory, the fact is that many same coordinates exist in different addresses of non-graphics Memory, which makes reverse search of coordinates of game objects difficult. Moreover, computer threads will have performance problems of producer / consumer model, thus nowadays game studio prefer Coroutines to decrease resource usages.

If the game is developed with C# or Java, then the program will be interpreted and compiled by CLR (Common Language Runtime) or JVM (Java Virtual Machine), which means JIT (just in time) will be used. JIT will compile "hot spot code" into local machine-related code with escape analysis. The basic behavior of escape analysis is to analyze the dynamic scope of objects and optimize it with following plans.

1) ALLOCATION ON THE STACK

If a game object will not escape beyond the method, JIT will allocate the object's Memory on the stack, so that the Memory space occupied by the object can be destroyed as the stack frame pops out.

2) SYNCHRONOUS ELIMINATION

If the escape analysis finds that the game object will not escape beyond the thread, then the synchronization measures of the object can be eliminated.

3) SCALAR REPLACEMENT

If a game object will not be accessed externally, and the object can be disassembled, then JIT may not create the object when the program is executed, and instead allocate member variables of the object on the Memory stack.

The results of above plans is that more same cached values are generated, making reverse search difficult. Furthermore, today's most games use object-oriented programming which introduces inherited classes and override functions. The compilers implement override functions in assembly codes by virtual table function.

IV. RMDQN MECHANISM

In this section, RMDQN's main components are described, they are RMDQN action engine and GDQN. Before that, a manual reverse procedure and RMDQN reverse procedure are shown first.

A. MANUAL REVERSE PROCEDURE

The general manual reverse procedure in Windows operation system is shown as follows:

1 Engineers use a debugger to read the debuggee exe file and exports PE (Portable Executable) structure. Find out Entry Point, Image Base, Code Section, and Data Section of debuggee.

2 Engineers invoke operation system functions to attach the debuggee process. According to the engineer experiences, recursively find suitable places to set hardware breakpoints or software breakpoints, for monitoring Memory data access, or for stopping code running.

  2.1 Target values are found, but there are many same values in different virtual addresses, and they may be stored as a pointers link. Therefore, engineers need to filter out false addresses, and go backwards to find out the base address of target value.

  2.2 Target functions are found, but parameters (always hexadecimal characters) of functions need engineers to map to meaningful human-readable characters. Engineers sometimes monitor stack traces to analyze virtual table functions, ebp addressing and esp addressing.

3 Engineers modify target values or target functions, and sometimes engineers need set variables and generate patches for repeating usages.

B. RMDQN REVERSE PROCEDURE

To illustrate the convenience our method brought, RMDQN procedure is shown as follows:

1 RMDQN reads the debuggee exe file and exports PE structure. Find out Entry Point, Image Base, Code Section, and Data Section of debuggee.

2 RMDQN invokes operation system functions to attach the debuggee process. According to GDQN, it recursively finds suitable places to set software breakpoints if the debuggee is not protected by anti-debug function, or set hardware breakpoints if the debuggee is protected.

  2.1 Target values are found fast by reading CR3 register to obtain page directory table, and scanning the game module Memory. If many same values exist in different virtual addresses, or they may be stored as a pointers link, RMDQN will use its Q-table and action engine to interact with the game. Finally, RMDQN filters out false addresses, and goes backwards to find out the base address of target value.

  2.2 Target functions are found, RMDQN will use its Q-table and action engine to interact with the game. Hexadecimal parameters of functions is no need to be mapped to human-readable characters. RMDQN will monitor stack traces to analyze virtual table functions, ebp addressing and esp addressing.

3 RMDQN modifies target values or target functions, and sometimes RMDQN set variables and generate patches for repeating usages if needed.

C. RMDQN ACTION ENGINE

RMDQN action engine is developed to deal with above challenge 1 in this paper. The action engine automatically uses a limited action space to control game objects and navigate in the game.

State space of RMDQN can be divided to 2 types according to specific moments: the moment after conducting a debug related action, and the moment after conducting a game control related action. The Action list of RMDQN is shown below.

D. GAME DEEP Q-NETWORKS (GDQN)

Unlike other DQNs, our enhanced DQN can learn actively, which is also called as GDQN. GDQN is created to deal with above challenge 2 and 3 in this paper. GDQN intelligently uses its Q-table and action engine to interact with game objects and observe corresponding changes. This section introduces our enhancements and an active query algorithm.

Markov decision process (MDP) is a standard reinforcement learning method, which can be defined by a tuple $M = \{S, A, R, T, \gamma\}$, where S are the states, A are the actions, R is the reward function, T is the transition probability function, and $\gamma \ = \in [0, 1)$ is the discount factor.

TABLE 4. Action list.

Types	Actions
Debug related	Set hardware breakpoints.
	Set software breakpoints.
	Set data breakpoints.
	Search values (including 2 subtype: 4 Bytes and Float).
	Next values (including 5 subtype: bigger than, smaller than, increased, decreased and unchanged).
	Search what code access (including 2 subtype: read and write) a specific address.
	Read values of CPU registers or specific Memory addresses.
	Write values of a specific Memory address (including 2 subtype: once or repeatedly).
	Inject codes to a specific Memory address (codes are pre-generated by human according to required tasks).
Game control related	Select a game object.
	Move a game object to a specific location.
	Control a game object to do a specific action (attack, patrol, and retreat, etc.)
	Recognize a specific image.
	Wait a period of time.

Conforming to the Bellman Optimality Equation, DQNs also learned an approximation of the optimal value function by a neural network. Q(s, a) in MDP are changed to Q(s, a; θ) in DQNs, where the parameter are learned by minimizing the Temporal Difference (TD) loss function:

$$L(\theta) = \left(r + \gamma \times \max_{a' \in A} Q\left(s', a'; \bar{\theta}\right) - Q(s, a; \theta)\right)^2 \quad (1)$$

There are two deep networks in DQNs: a target deep network and an evaluated deep network, which have the same architecture, but the target deep network is kept frozen for a period of time. θ are the parameters of evaluated deep network. $\bar{\theta}$ are the parameters of target deep network, which are copied from θ regularly to stabilize the target Q-values. The evaluated deep network is trained in each step to minimize TD loss function.

To improved overall performance, we used a selection of existing modifications that each has addressed a limitation of classic DQNs.

1) DOUBLE Q-LEARNING

To reduce the overestimation of the target value, double Q-learning calculates the target value by replacing $\max_{a' \in A} Q\left(s', a'; \bar{\theta}\right)$ with $Q\left(s', \operatorname{argmax}_{a' \in A} Q\left(s', a'; \theta\right); \bar{\theta}\right)$. The TD loss is:

$$L(\theta) = \begin{pmatrix} r + \gamma \times Q\left(s', \operatorname{argmax}_{a' \in A} Q\left(s', a'; \theta\right); \bar{\theta}\right) \\ -Q(s, a; \theta) \end{pmatrix}^2 \quad (2)$$

2) PRIORITIZED EXPERIENCE REPLAY

Prioritized experience replay samples more frequently those transitions that needs much to learn. Prioritized experience replay uses weighted sampling of replay buffer to replace the uniform sampling of conventional DQN. The probability of weighted sampling each transition is relative to the last absolute TD error:

$$p_{\text{sampling}} \propto \left| \begin{array}{c} r + \gamma \times \max_{a' \in A} Q\left(s', a'; \bar{\theta}\right) \\ -Q\left(s, a; \theta\right) \end{array} \right|^{w} \tag{3}$$

3) DUELING NETWORKS

The dueling network uses the value and advantage streams. For a particular state, if the value of the action is greater than the average value, then the advantage function is positive and vice versa. V(s) concerns the value of a state, and A(s, a) concerns the relative importance of the action in this state:

$$Q(s, a) = V(s) + (A(s, a) - \frac{1}{|A|} \sum_{a}^{\prime} A\left(s, a'\right) \tag{4}$$

4) MULTI-STEP LEARNING

The learning speed of conventional DQN is relatively slow in the early stage of training. We can solve this problem through Multi-Step Learning, so that the target value can be estimated more accurately. The immediate reward can be accurately obtained through interaction with the environment, thus it speeds up the training speed. The truncated n-step reward from a given state St is defined as:

$$R_t^n = \sum_{k=0}^{n-1} \gamma_t^k R_{t+k+1} \tag{5}$$

The loss is thereby updated:

$$L(\theta) = \left(\begin{array}{c} \gamma_t^n \times Q\left(S_{t+n}, \operatorname{argmax}_{a' \in A} Q\left(S_{t+n}, a'; \theta\right); \bar{\theta}\right) \\ R_t^n - Q\left(S_t, a; \theta\right) \end{array} \right)^2 \tag{6}$$

5) DISTRIBUTION OF RETURNS

In DQN, the output of the network is the expected estimate of the state-action value Q. This expectation actually ignores a lot of information. For example, the expectation of the value of two actions in the same state is the same. However, the value of the first action is 10 in 90% of the cases, and 110 in 10% of the cases, the value of the second action is 15 in 50% of the cases and 25 in 50% of the cases. Although the expectation is the same, if we want to reduce the risk, we should choose the latter action. Therefore we used the cross entropy loss function (Kullbeck-Leibler divergence) to calculate the gap between the two distributions, where d_t^n is a target distribution after L2-projection, and d_t^1 is a 1-step distribution:

$$KL(d_t^n || d_t^1) \tag{7}$$

6) NOISY NETS

We added noise to replace epsilon-greedy policy to increase the exploratory ability of the model. The noisy parameters η, which are learnable parameters, are placed in the output layer of a network. The agent will obtain a sample of η and act according to the optimal policy.

Besides existing modifications mentioned above, we designed an active query algorithm to make RMDQN become an active learning method. Active learning requires an agent interacts with an expert by querying what action to take in a state. After querying, the expert will take over the decisions of actions for a few consecutive steps, then the agent takes back the decisions of actions. The challenge of this process is to decide when to query the expert so that efficiency and benefit are both kept. This challenge in our scenarios can be solved by the query algorithm below:

```
Query Algorithm
Static: Uque_Length, Limit_Count, Limit_Slope
Input: Unew, Uque, ref Count
Output: Trigger_Query

Add Unew to Uque
Trigger_Query = False
Count++
If Count == Limit_Count
Count = 0
Unorm = Norm (Uque)
Slope = OLS (Unorm)
If Slope > Limit_Slope
    Trigger_Query = True
Return Trigger_Query
```

Unew indicates uncertainty of a state, Uque is a queue that stores uncertainties of recent consecutive states, Uque_Length is the queue length. Count is a reference number which is initiated outside the algorithm and modified inside the algorithm. When Count reached Limit_Count, uncertainty values will be normalized and a linear regression method (Ordinary Least Square, OLS) will be used to compute a Slope. When Slope is larger than Limit_Slope, meaning recent uncertainties increase largely, the agent will query the expert. Unew can be defined by variance of a state:

$$\text{Unew} = \operatorname{Var}\left[Q\left(s, \operatorname{argmax}_{a \in A} Q(s, a; \theta)\right)\right] \tag{8}$$

V. EXPERIMENTS

This paper selects two typical games for experiments: FPSG and RTSG. We run the 3D video games in Windows 7 operation system, 8GB RAM, 1.6GHz Intel i7 CPU. RMDQN action engine is implemented with the help of a tool, AutoHotkey, which is a script programming software. AutoHotkey can simulate mouse move, click and keystrokes, etc. Therefore, RMDQN action engine is able to control any operational game object like a human player is playing.

We trained two tasks in each game (FPSG and RTSG) in the experiments to: discover coordinate values of a game object; discover and lock a health value of a game object. In each step of both tasks, the agent receives -1 reward. In the task of coordinate discover, the episode ends until 6000 steps or the goal is reached. The average reward of task is -527

TABLE 5. Methods comparisons.

Methods	Demonstration	Active query policy
DDQN	no	no
DQfD	yes	no
RMDQN-a	yes	actively query in all states, until reached Limit_Query
RMDQN-p	yes	actively query in part of states, with certain probability
RMDQN	yes	use active query algorithm in this paper

TABLE 6. Hyper-parameters.

Parameter	Value
Adam learning rate	0.00005
Adam ε	0.00013
Prioritization type	proportional
Prioritization exponent ω	0.5
Prioritization importance sampling β	0.3 → 1.0
Noisy Nets σ	0.6
Distributional atoms	49
Distributional min/max values	[−10, 10]
Multi-step n	4
Limit_Count	20
Limit_Slope	0.8

in 80 trials. In the task of health discover, the episode ends until 3000 steps or the goal is reached. The average reward of task is -273 in 80 trials.

For comparisons, a high quality method that mentioned in related works, DQfD, was modified and supported by RMDQN action engine to adapt to our environment. Thereafter, we compare RMDQN, DQfD and other variants in the experiments as the table below shows. DDQN is prioritized Double DQN.

Our query algorithm needs three parameters: Uque_Length, Limit_Count, and Limit_Slope. A normalization function proportionally scales each uncertainty in Uque and multiplies each normalized uncertainty with 100 to match Uque_Length. We use the Adam optimizer and tune hyper-parameters in this paper in a limited manual way. For each hyper-parameter in existing technologies, we started with the values set in the related papers, and tune the most sensitive hyper-parameters by manual coordinate descent.

A. ACTIVE QUERY POLICIES

When do agents query the expert? A simple way is to set a fixed threshold. Once the uncertainty of a state exceeds the fixed threshold, the agent asks the expert for demo. Otherwise the agent decides to take actions by itself. However, it is no necessary to query if the uncertainties of succeed states unintentionally decrease. In addition, it is difficult to find a proper threshold for different tasks. To prove that, we designed RMDQN-a and RMDQN-p. The number of Limit_Query is used to set max times of querying the expert in both RMDQN-a and RMDQN-p. The probability of querying the expert is no larger than 0.8 in RMDQN-p. The Limit_Query is 500 in the task of coordinate discover, and 200 in the task of health discover. The comparison results are shown in Fig. 1(a-d). The x-coordinate is the numbers of steps, and the y-coordinate is the scores in Fig. 1 and Fig. 2.

Fig. 1 (a) shows that besides RMDQN, RMDQN-a achieves better performance in the first 1000 steps. The reason is that RMDQN-a always queries the expert for a demo until running out of Limit_Query. However, thereafter its performance decreases and is surpassed by RMDQN-p, meaning the expert is a constantly good consultant. Fig. 1 (b) shows a different picture, the performance of RMDQN-a and RMDQN-p do not bias too much. The reason is that initial queries about coordinates are more important in RTSG. Once obtaining correct direction of searching, following steps would be easier.

RTSG often has more game objects than FPSG, and the construction order of these game objects is not fixed. Likewise, the destruction order of these game objects is also not fixed. These uncertainties make the performance of RMDQN-a decreases as Fig. 1 (d) shows, which are common phenomenon.

In general, the code space and data space of Memory of RTSG is larger than FPSG, therefore the complexity of searching is higher and more steps will be taken in RTSG as Fig. 1 (a-d) presented. Among the compared methods, RMDQN is the most competitive one.

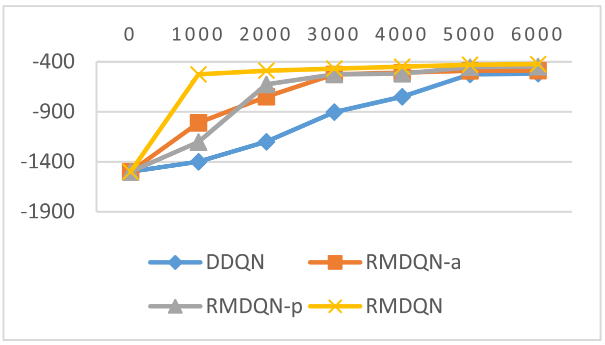
B. AI EXPERT ABILITIES

Why use AI experts instead of human experts? We need automatic debug in the last. We set three levels of ability (low, middle, and high) of AI experts according to performances of solving tasks. To simulate human experts as far as possible: Low ability AI expert cannot perfectly solves tasks; Middle ability AI expert solves tasks with low performance; High ability AI expert solves tasks before the termination condition is reached. These AI experts are used to collect demonstration data in DQfD and take over decisions of actions in RMDQN.

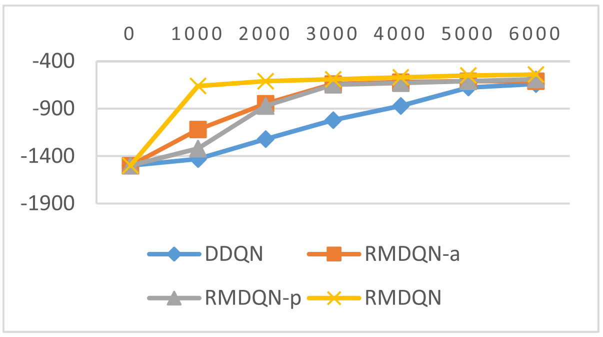
C. AI EXPERTS IN RMDQN AND DQFD

How do three kinds of AI experts affect the performance of RMDQN and DQfD, especially with different Limit_Query? The high ability AI expert can make optimal choices and solve tasks efficiently, while weaker experts (including middle ability AI experts and low ability AI experts) may fail to solve the task. Weaker experts are obtained by random actions with a probability rather than following the perfect policy. However, the random actions sometimes successfully reach the end of episode.

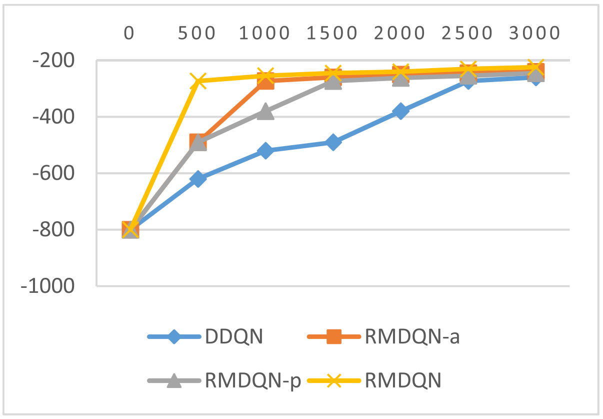
In previous experiments, RMDQN is used with high ability AI expert, and the previous notation RMDQN equals to RMDQN-h, likewise DQfD-h. In following experiments, RMDQN-h and DQfD-h means that high ability AI experts are used, RMDQN-w and DQfD-w means that weaker experts are used. Fig. 2(a-d) shows the effect of expert's ability on RMDQN and DQfD in two kinds of tasks.



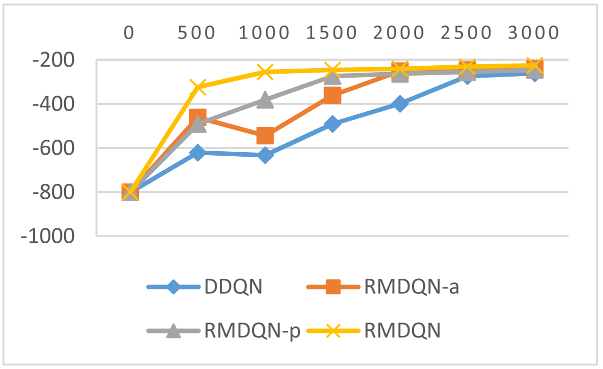
(a) Coordinate discover in FPSG



(b) Coordinate discover in RTSG



(c) Health discover in FPSG



(d) Health discover in RTSG

FIGURE 1. **(a) Coordinate discover in FPSG. (b) Coordinate discover in RTSG. (c) Health discover in FPSG. (d) Health discover in RTSG.**

The Fig.2 demonstrates both RMDQN and DQfD can solve the tasks in few steps and learn efficiently with the high ability AI expert. In contrast, they learn slower and are misled by weaker AI experts more or less. Nevertheless, DQfD converges at a worse score with the weaker AI experts, and RMDQN converges at higher scores. Therefore, RMDQN can take full advantage of weaker AI experts, which are similar to ordinary human experts, meanwhile RMDQN can make better choices by itself.

TABLE 7. **Statistics of different ai experts.**

Tasks		Statistics	Low ability AI	Middle ability AI	High ability AI
Coordinate discover	FPSG	Mean score	-589	-543	-527
		Avg. steps	2987	2210	1022
	RTSG	Mean score	-690	-644	-613
		Avg. steps	3102	2341	1256
Health discover	FPSG	Mean score	-288	-256	-245
		Avg. steps	1782	989	552
	RTSG	Mean score	-297	-269	-240
		Avg. steps	2248	1177	670

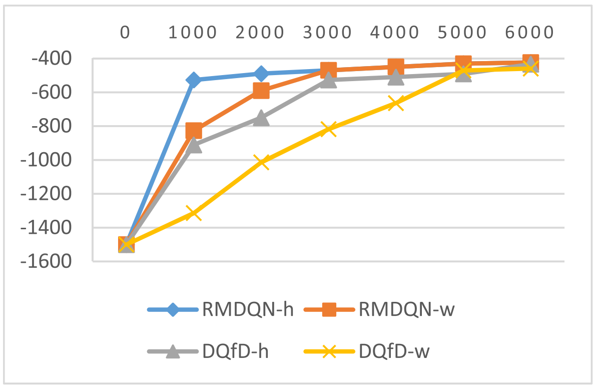
In general, DQfD is passive and RMDQN is active. With bigger Limit_Query, DQfD begins with more demonstration data, which leads to better initial performance, but its performance suffers by the imperfect demonstration data. RMDQN is more robust to learn from weaker AI experts and surpasses DQfD in most of cases, which proved advantages of active learning.

D. UNCERTAINTY FLUCTUATIONS

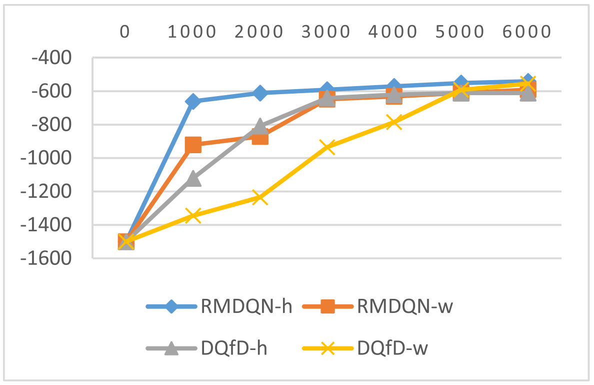
How do the uncertainties of states change during the learning processes of agents? Uncertainties of a state can be calculated by variances of the state. Overall uncertainties in critical steps are obtained during the learning processes. The settings of this experiment are according to previous settings. Fig. 3(a-d) shows the uncertainties fluctuation in two kinds of tasks.

The x-coordinate is the numbers of steps, and the y-coordinate is the degrees of uncertainty in Fig. 3. The degrees of uncertainty depend on the complexity of problem. There are periods that the uncertainty is non-increasing in the two kinds of tasks. In the beginning, the number of exploration of the agent is larger than the number of exploitation. Therefore, the uncertainties in the beginning increase in both RMDQN and DQfD. It is observed that the uncertainties are not decreasing all the time. During the learning processes, the general trend of uncertainties decrease. In the end of learning processes, the uncertainties gradually converge.

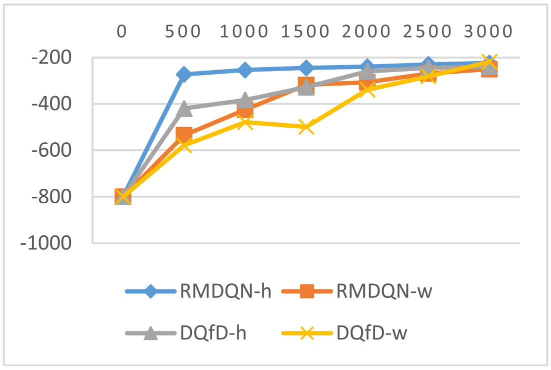
Generally speaking, Fig. 3(a-d) showed that the uncertainties are churned to gradually converge. It is precisely because of helps of AI experts, whether in a passive way or active way. In addition, RMDQN demonstrates its adaption again, shrinking the uncertainties fast.



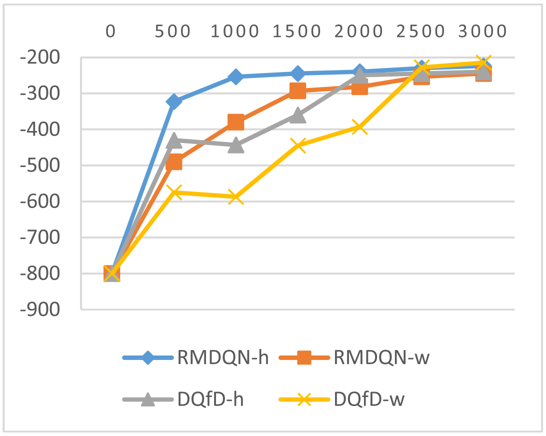
(a) Coordinate discover in FPSG



(b) Coordinate discover in RTSG

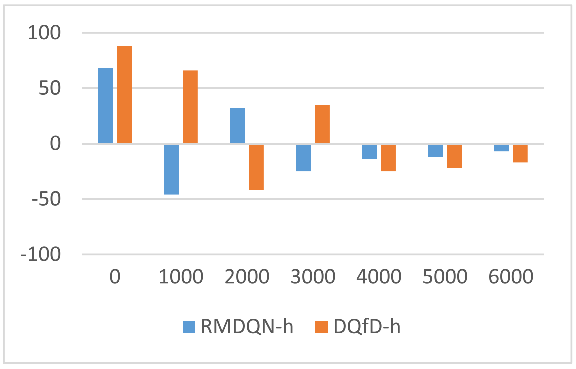


(c) Health discover in FPSG

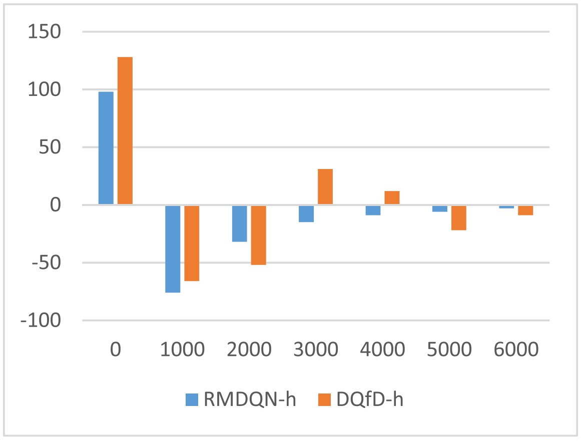


(d) Health discover in RTSG

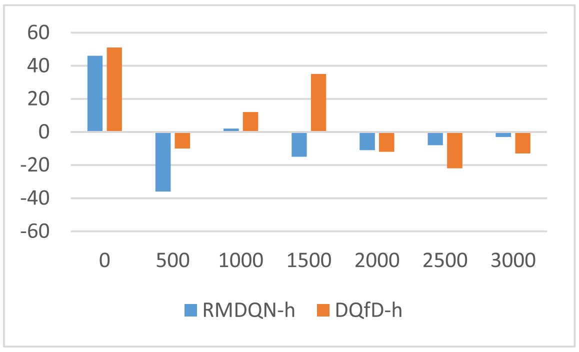
FIGURE 2. (a) Coordinate discover in FPSG. (b) Coordinate discover in RTSG. (c) Health discover in FPSG. (d) Health discover in RTSG.



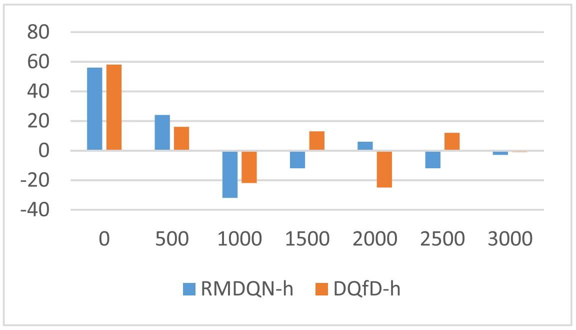
(a) Coordinate discover in FPSG



(b) Coordinate discover in RTSG



(c) Health discover in FPSG



(d) Health discover in RTSG

FIGURE 3. (a) Coordinate discover in FPSG. (b) Coordinate discover in RTSG. (c) Health discover in FPSG. (d) Health discover in RTSG.

E. GAME AI ANALYSIS

Why are game AIs not suitable for Memory search? We concisely analyze AlphaGo, which used modified CNN and MCTS (Monte Carlo Tree Search) to find a proper policy in a short time. This idea may be used in reverse engineering. However, we discovered that ''temporal difference'' is better than MCTS in our scenario after experiments, because rollout policy may not reach a meaningful Memory address in the end, which is not suitable for our scenario.

F. VIDEO DEMONSTRATIONS AND IMPROVEMENTS

A video demonstration of RMDQN is provided [39], which shows the result of reversely modifying properties in two games. Three typical modifications are shown in the video: a light car is modified to be invincible; the war fog is removed so that enemy unites in anywhere can be seen (which is called map hack); enemies glow behind walls.

Some games use VAC (Valve Anti Cheat) system to detect and ban cheating players, but it does not detect our program, thus we successfully changed Memory values in steam online games while contesting with other human players, as the video demo shows.

In addition, we propose some anti-reverse engineering means. Knowing how to debug the video games is the premise of knowing how to anti-debug. Therefore I suggest that:

1. Using CRC (Cyclic Redundancy Check) or other integrity check methods to protect target code integrity.
2. Forbidding system call OpenProcess, GetProAddress, CreateRemoteThread, WriteProcessMemory and VirtualAllocEx, etc. to access target process.
3. Regularly searching whether there are handles of other processes in the system that contain handles of target processes. Eprocess is the process structure of Ring 0, and PEB (Process Environment Block) is the structure of Ring 3.

VI. RESEARCH MOTIVATION

Software engineers use semi-automatic tools for debugging software. Semi-automatic tools, including Ollydbg, Windbg, and Cheat Engine, are also mature enough for reverse engineering. However, it still needs huge manual operations to manipulate the tools and needs large brainwork to analyze computer Memory data, register statues, stack traces, and assembly jumps, etc. It is such a double burden to analyze assembly code with human eyes and brains, especially for big software. Furthermore, Ollydbg seems to be frozen because its main functions have not been updated since year 2014.

Therefore, we proposed a new automatic method for discovering and modifying software properties to improve debugging technology. For evaluations of our method, we choose a specific software domain, 3D video games. According to the features of video games, this paper selected two typical games for demonstrations and evaluations of our method. Our method has following advantages:

1. Because of RMDQN action engine, RMDQN does not need simulation environments, i.e., it works in real software environments. Traditional DQNs for games are often used in simulation environments or Atari games from the Arcade Learning Environment, so that an agent can easily interact with virtual ambient objects, and obtain reward immediately. However, traditional DQNs need more supports in real software, because perceptions and actions must be translated to specifications conforming to real software settings.
2. Because of GDQN, RMDQN learns to find target memory addresses intelligently and automatically. Generally, there are many same values in different virtual addresses of a video game in computer Memory, but only one virtual address stores the real property value, and it may be stored as a pointers link. Moreover, this virtual address may change after each starting of the video game. Not only virtual addresses may change each time, but also the sequence of game functions may change, causing Memory stack traces change. In addition, Object-oriented programming introduces inherited classes and override functions. RMDQN can handle above problems.

Our method has following disadvantages. The fundamental technology of our method is reinforcement learning. Any reinforcement learning method needs to tackle two important problems: exploration-exploitation problem and long-term credit assignment problem. However, different reinforcement learning tasks require different exploration policy and assignment policy depends on types of tasks. Therefore, our method may encounter performance problem in general tasks.

RMDQN tackled the two problems above. Furthermore, if integrating with debugging software, it is possible to interact with human experts to switch debugging control seamlessly, and allow human experts to inspect recent states of Memory, registers and stack traces. It will be proved that our method of active and deep learning can improve debugging technology in wider scenarios, or general software. For example, cloud services are connected by network protocols. We can hook Send() function of DLL in the operation system to do the reverse jobs.

VII. CONCLUSION

This paper did not use artificial intelligent to train game NPCs (Non-Player Character) or like AlphaStar in game StarCraft 2, but trained an improved AI to modify properties of video games. The purpose of this paper is to boost automatic debugging in video games. In this paper, we proposed RMDQN to reversely and automatically discover and modify properties in video games. A video demonstration is provided to show the effect of reversely modifying game properties by out method. If integrating with debug software, it is possible to interact with human experts to switch debug control seamlessly, and allow human experts to inspect recent states of Memory, registers and stack traces. It was proved that our method can improve debugging technology in 3D video games, and it will be applied in cloud services with few modifications.

REFERENCES

[1] H. Washizaki, Y.-G. Guéhéneuc, and F. Khomh, "ProMeTA: A taxonomy for program metamodels in program reverse engineering," *Empirical Softw. Eng.*, vol. 23, no. 4, pp. 2323–2358, Aug. 2018.

[2] H. Shu *et al.*, "Control reconfiguration of dynamical systems for improved performance via reverse-engineering and forward-engineering," 2020, *arXiv:2003.09279*. [Online]. Available: https://arxiv.org/abs/2003.09279

[3] M. Poženel and B. Slivnik, "Using clickstream data to enhance reverse engineering of Web applications," *Adv. Comput.*, vol. 116, no. 1, pp. 305–349, 2020.

[4] E. A. Miranda, M. Berón, G. Montejano, and D. Riesco, "Using reverse engineering techniques to infer a system use case model," *J. Softw., Evol. Process*, vol. 31, no. 2, Feb. 2019, Art. no. e2121.

[5] D. Parras, "Reconstruction by low cost software based on photogrammetry as a reverse engineering process," in *Proc. HCI*, vol. 9, 2018, pp. 145–154.

[6] U. Sabir, F. Azam, S. U. Haq, M. W. Anwar, W. H. Butt, and A. Amjad, "A model driven reverse engineering framework for generating high level UML models from java source code," *IEEE Access*, vol. 7, pp. 158931–158950, 2019.

[7] F. ELotmani, R. Esbai, and M. Atounti, "The reverse engineering of a Web application struts based in the ADM approach," *Int. J. Online Biomed. Eng. (iJOE)*, vol. 16, no. 2, p. 112, Feb. 2020.

[8] C. Basile, D. Canavese, L. Regano, P. Falcarin, and B. D. Sutter, "A meta-model for software protections and reverse engineering attacks," *J. Syst. Softw.*, vol. 150, pp. 3–21, Apr. 2019.

[9] P. Barceló, "A theoretical view on reverse engineering problems for database query languages," *Description Logics*, 2019.

[10] D. M. L. Martins, "Reverse engineering database queries from examples: State-of-the-art, challenges, and research opportunities," *Inf. Syst.*, vol. 83, pp. 89–100, Jul. 2019.

[11] I. Tomicic, P. Grd, and M. Schatten, "Reverse engineering of the MMORPG client protocol," in *Proc. 42nd Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2019, pp. 1099–1104.

[12] U. J. Botero *et al.*, "Hardware trust and assurance through reverse engineering: A survey and outlook from image analysis and machine learning perspectives," 2020, *arXiv:2002.04210*. [Online]. Available: https://arxiv.org/abs/2002.04210

[13] M. Fyrbiak *et al.*, "Hardware reverse engineering: Overview and open challenges," in *Proc. IEEE 2nd Int. Verification Secur. Workshop (IVSW)*, 2017, pp. 88–94.

[14] M. Fyrbiak, S. Wallat, P. Swierczynski, M. Hoffmann, S. Hoppach, M. Wilhelm, T. Weidlich, R. Tessier, and C. Paar, "HAL—The missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 3, pp. 498–510, May 2019.

[15] M. Alam and D. Mukhopadhyay, "How secure are deep learning algorithms from side-channel based reverse engineering?" in *Proc. 56th Annu. Design Autom. Conf.*, Jun. 2019, pp. 1–2.

[16] J. Qiu, W. Luo, L. Pan, Y. Tai, J. Zhang, and Y. Xiang, "Predicting the impact of Android malicious samples via machine learning," *IEEE Access*, vol. 7, pp. 66304–66316, 2019.

[17] A. Nirumand, B. Zamani, and B. Tork Ladani, "VAnDroid: A framework for vulnerability analysis of Android applications using a model-driven reverse engineering technique," *Softw., Pract. Exper.*, vol. 49, no. 1, pp. 70–99, Jan. 2019.

[18] B. R. Ray and S. Chowdhury, "Reverse engineering technique (RET) to predict resource allocation in a Google cloud system," in *Proc. 8th Int. Conf. Cloud Comput., Data Sci. Eng. (Confluence)*, Jan. 2018, pp. 688–693.

[19] G. Gharibi, R. Tripathi, and Y. Lee, "Code2graph: Automatic generation of static call graphs for Python source code," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng. - ASE*, Sep. 2018, pp. 880–883.

[20] D. J. Miller, Y. Wang, and G. Kesidis, "Anomaly detection of attacks (ADA) on dnn classifiers at test time," in *Proc. IEEE 28th Int. Workshop Mach. Learn. Signal Process. (MLSP)*, Sep. 2018, pp. 1–6.

[21] R. Xiao, Y. Zhu, B. Lu, X. Zhu, and S. Liu, "Recognizing the data type of firmware data segments with deep learning," *IEEE Access*, vol. 8, pp. 69167–69185, 2020.

[22] M. Marchetti and D. Stabili, "READ: Reverse engineering of automotive data frames," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 4, pp. 1083–1097, Apr. 2019.

[23] Z. Tong, H. Chen, X. Deng, K. Li, and K. Li, "A scheduling scheme in the cloud computing environment using deep Q-learning," *Inf. Sci.*, vol. 512, pp. 1170–1191, Feb. 2020.

[24] A. Kaur, P. Singh, R. Singh Batth, and C. Peng Lim, "Deep-Q learning-based heterogeneous earliest finish time scheduling algorithm for scientific workflows in cloud," *Softw., Pract. Exper.*, Feb. 2020.

[25] D. Zhang, F. R. Yu, and R. Yang, "Blockchain-based distributed software-defined vehicular networks: A dueling deep Q -learning approach," *IEEE Trans. Cognit. Commun. Netw.*, vol. 5, no. 4, pp. 1086–1100, Dec. 2019.

[26] C. J. Lin, J. Y. Jhang, H. Y. Lin, C. L. Lee, and K. Y. Young, "Using a reinforcement Q-learning-based deep neural network for playing video games," *Electronics*, vol. 8, no. 10, p. 1128, Oct. 2019.

[27] Z. Ji and W. Xiao, "Improving decision-making efficiency of image game based on deep Q-learning," *Soft Comput.*, vol. 24, no. 11, pp. 8313–8322, Jun. 2020.

[28] S.-A. Chen *et al.*, "Active deep Q-learning with demonstration," *Mach. Learn.*, pp. 1–27, 2019. [Online]. Available: https://link.springer.com/article/10.1007%2Fs10994-019-05849-4

[29] G. Jeong and H. Y. Kim, "Improving financial trading decisions using deep Q-learning: Predicting the number of shares, action strategies, and transfer learning," *Expert Syst. Appl.*, vol. 117, pp. 125–138, Mar. 2019.

[30] Hester, T., Vecerik, M., Pietquin, and O., Lanctot, "Deep Q-learning from demonstrations," in *Proc. 32nd AAAI Conf. Artif. Intell., AAAI*, vol. 6, Apr. 2018, pp. 1–12.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[32] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, vol. 1509, 2016, Art. no. 06461.

[33] Z. Wang, T. Schaul, M. Hessel, H. V. Hasselt, M. Lanctot, and N. D. Freitas, "Dueling network architectures for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1995–2003.

[34] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. van Hasselt, and D. Silver, "Distributed prioritized experience replay," 2018, *arXiv:1803.00933*. [Online]. Available: http://arxiv.org/abs/1803.00933

[35] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, "Recurrent experience replay in distributed reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–19.

[36] A. P. Badia, P. Sprechmann, A. Vitvitskyi, D. Guo, B. Piot, S. Kapturowski, O. Tieleman, M. Arjovsky, A. Pritzel, A. Bolt, and C. Blundell, "Never give up: Learning directed exploration strategies," 2020, *arXiv:2002.06038*. [Online]. Available: https://arxiv.org/abs/2002.06038

[37] J. Fu, J. Co-Reyes, and S. Levine, "EX2: Exploration with exemplar models for deep reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 2577–2587.

[38] Z. Xu, H. V. Hasselt, and D. Silver, "Meta-gradient reinforcement learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 2396–2407.

[39] *Video Demo*. Accessed: Apr. 21, 2020. [Online]. Available: https://www.bilibili.com/video/BV1Yp4y1X7fB/

LEI YU (Member, IEEE) graduated from the Beijing University of Posts and Telecommunications. He is currently a Lecturer and a M.S. Supervisor working with the Inner Mongolia Engineering Laboratory of Cloud Computing and Service Software, Inner Mongolia University, China. He published more than 20 SCI and EI articles as first author. His research interests include data mining, artificial intelligence, and Web service composition. He received grants including: Program of Higher-level talents of Inner Mongolia University, and Natural Science Foundation of China.

ZHIFA HUO is currently pursuing the M.S. degree with the College of Computer Science, Inner Mongolia University, China. His research interests include software engineering and cloud computing: service discovery, selection, and verification.