Adaptive Web Server Optimisation: A FuzzyART-Based Classifier for Dynamic Request Patterns

Lee Xing Le
B.Sc. (Hons) Computer Science
School of Computing and Communications Lancaster University
21 March 2025

Statement of Originality

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted work, I consent to this being stored electronically and copied for assessment purposes, including the School's use of plagiarism detection systems in order to check the integrity of assessed work.

I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Name: Lee Xing Le
Date: 21 March 2025

Abstract

This research focuses on developing an automated classifier model to dynamically categorise web request patterns, addressing the challenge of web server systems that struggle to adapt to unpredictable web request environments. By integrating a FuzzyART, this study aims to improve the learning efficiency of reinforcement learning (RL)-enabled software systems, allowing them to adapt to changing web request patterns more effectively. The proposed model classifies distinct web request patterns into different groups, enabling a multi-instance learning agent to determine the optimal web server composition for each pattern. Experimental results demonstrate that the classifier performs well, accurately categorising web requests and significantly improving the adaptability of the RL system. As a result, the learning agent can efficiently identify the best server configurations, leading to better response times and overall web server performance. These findings highlight the importance of the automated classifier in optimising RL-based web server adaptation, ultimately improving system efficiency in dynamic environments.

Contents

1 Introduction

Modern software systems are becoming increasingly complex and are deployed in dynamic and unpredictable environments as discussed in [1]. This complexity is widely recognised as unsustainable for humans to manage effectively, prompting significant research on autonomic, self-adaptive, and self-organising software systems [2]. These systems aim to shift critical management responsibilities from humans to software, enabling them to autonomously configure and adapt by assembling themselves from available building blocks without human intervention. They are designed to modify their behaviour during runtime while ensuring that system requirements are met to a satisfactory level. These systems achieve this by gathering new knowledge at runtime to address uncertainties, reasoning about their state, context, and goals, and adapting their behaviour to fulfil their objectives, or gracefully degrading performance, when necessary. Such capabilities are critical for maintaining operational efficiency in dynamic settings where workloads and requirements can change unexpectedly.

Dynamic environments are becoming more prevalent due to the continuous increase in Internet usage and the demand for high-performance software systems. As shown in the Gmail production data, both the load and the nature of the load change continuously [3]. Events such as sudden spikes in traffic or resource-intensive processes can quickly lead to server overload if the system does not adapt promptly. This can severely impact the performance of a website, resulting in user dissatisfaction, loss of revenue, and reputation damage for businesses. Conversely, during periods of low traffic or under utilisation, servers must downgrade resource usage to prevent wastage, reduce operational costs, and promote energy efficiency. This aligns with the growing emphasis on sustainable technology, a key trend shaping the future of the software industry.

One significant challenge in these systems lies in their ability to adapt to changing web request patterns while maintaining optimal performance. Reinforcement learning (RL) has been employed to optimise server configurations based on observed patterns [4]. However, existing RL systems often struggle with high-convergence times when facing complex patterns, such as fluctuating traffic, varying entropy, or diverse resource demand requests. Most RL approaches are pre-trained on predicted likely environments, making them less effective when encountering unforeseen or highly dynamic scenarios that deviate from initial assumptions. Since web server request patterns change dynamically in real-world settings, pre-trained models often struggle to adapt efficiently, leading to suboptimal performance and resource inefficiencies. This challenge is particularly critical in scenarios requiring rapid adaptation. Addressing this limitation is essential to ensure that software systems remain robust and efficient across a wide range of web request environments.

The motivation for this research arises from the need for more adaptive and efficient software systems capable of handling dynamic workloads. Current RL approaches face limitations that hinder their practical application in real-world, high-variability scenarios. The proposed classifier aims to bridge the gap between static RL approaches and the demands of dynamic environments by equipping RL systems with the ability to adapt more quickly and efficiently.

This research addresses these challenges by developing an automated classifier model that categorises web request patterns dynamically to enhance the performance of RL-

enabled software systems. Unlike traditional approaches, the proposed model can adapt automatically to changes in the request environment without human intervention, ensuring optimal system performance even in dynamic conditions.

Specifically, the research will focus on:

1. Analysing features that are significant for categorising web request patterns.
2. Developing an automated classifier model to classify web request patterns.
3. Evaluating the impact of the classifier on the performance of RL systems.

By achieving these objectives, this research seeks to contribute to the development of more responsive, sustainable, and efficient software systems that meet the needs of today's dynamic technological landscape.

The paper is divided into the following sections: Section 2 reviews related works. Section 3 discusses the methodology, providing a detailed explanation of the approach. In Section 4, the implementation of the system is explained, including descriptions of the Python code used. Section 5 evaluates the performance of the classifier and its significance. Finally, Section 6 concludes the paper and outlines potential directions for future work.

2 Related Works

In this section, we review existing research on self-adaptive systems, reinforcement learning, and automated classifier methods.

2.1 Self-adaptive Systems

Self-adaptive systems are systems that can manage themselves with minimal human intervention, they actively modifying their behaviour during runtime to meet predefined objectives, such as maintaining performance or ensuring resource efficiency.

One study by [5] focuses on improving the response speed and accuracy of Transient Stability Assessment (TSA) in power systems. The Intelligent System (IS) proposed in the paper utilises a series of Extreme Learning Machine (ELM)-based ensemble classifiers, each configured to response in different response times. The system adapts by selecting the classifier with the quickest response time that maintains accuracy, thereby improving reliability and performance. This adaptive approach ensures faster decision-making, leaving more time for emergency controls, with an overall classification accuracy exceeding 99.4%. While the domain of this research differs from web request patterns, it provides valuable insights into balancing speed and accuracy in dynamic environments. Similar to this study's goal of improving TSA response times, my research aims to enhance the response time of web servers under dynamic workloads. However, my work focuses on classifying web request patterns in real time, even in previously unseen environments. Unlike the predefined categories used in the TSA system, my approach relies on dynamic classifying web request patterns, ensuring adaptability without prior training. By applying these principles to web servers, my research addresses a gap in improving response times in dynamic web environments.

In the field of self-adaptive systems, the paper on the Self-Adaptive Blockchain-Based Intrusion Detection System (SAB-IDSI) addresses the challenge of improving the accuracy of intrusion detection [6]. SAB-IDSI employs a reinforcement learning framework to enhance its detection capabilities over time by dynamically adjusting detection thresholds, response tactics, and anomaly profiles based on observed network behaviour. This system focus on improving accuracy and overall performance in detecting genuine intrusions while minimising false alarms. Its decentralised design also reduces the risk of single points of failure, a critical concern in centralised systems. The paper demonstrates measurable improvements, with SAB-IDSI achieving a detection accuracy of 96.2%, a recall rate of 94.8%, and a balanced F1-Score of 95.5%. The insights from SAB-IDSI are relevant to my research, as both projects share a focus on self-adaptation to improve system performance and utilise reinforcement learning for decision-making. However, there are key differences in the domains and objectives. While SAB-IDSI is designed for intrusion detection with adaptation based on training, my project focus on dynamic classifying of web request patterns to optimise response times for web server components. With requiring the classifier to handle new patterns and dynamically adjust to changing web traffic conditions. SAB-IDSI's success in adapting to evolving network behaviours provides a foundation and inspiration for my work, However, my project innovates by emphasising dynamic classifying to enhance web server performance, ensuring that the system can respond effectively to unknown or highly variable web request patterns.

Another paper [4] addressed the problem of developing a self-adaptive runtime emer-

gent software system, in which decisions about the assembly and adaptation of software were machine-derived without relying on human experts to specify models, policies, or adaptation processes. It focused on performance improvements, with the system demonstrating a highly effective ability to rapidly discover optimal compositions of behaviour in a web server environment. The system effectively balanced exploration and exploitation and was highly responsive to changes in deployment conditions over time, such as varying input patterns and system loads. The paper proposed several methods to achieve this. It introduced Dana, a programming language designed for creating small, modular software components that could be assembled into emergent systems with near-zero-cost runtime adaptation. It also featured a perception, assembly, and learning (PAL) framework that dynamically discovered, assembled, and perceived the effectiveness of software configurations. The perception data was fed into an online learning module, which applied statistical linear bandits with Thompson sampling to efficiently search through the vast space of possible configurations. The system monitored metrics like response time to client requests and the size and type of requested resources, using this information to determine when to adapt. Events and metrics were reported to a Recorder interface, enabling continuous monitoring and adaptation based on real-time environmental conditions. The key findings highlighted that the system could identify different optimal configurations for various environments using reinforcement learning (RL). However, it faced significant limitations in dynamic environments where web request patterns frequently changed. The RL system struggled with high convergence times due to its reliance on past experiences in a stable environment. When the environment shifted, the system had to relearn optimal configurations, slowing the overall adaptation process. This limitation indicated the need for more efficient handling of dynamic traffic patterns. My research builds on the original paper by introducing a classifying step before applying RL to web request patterns. This dynamically categorises any request pattern, even in unforeseen environments, without relying on predefined scenarios. By classifying request patterns before RL, my system can handle unpredictable web traffic more efficiently and reduce convergence times by tailoring RL instances to specific, identified categories. This advancement provides a more robust and adaptive approach to improving response times and optimising web server configurations in dynamic and evolving environments.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a type of machine learning technique that focuses on how an agent makes decisions in a dynamic environment to achieve the best possible reward. In RL, the agent learns through trial and error by interacting with its environment, selecting actions, and receiving feedback in the form of rewards or penalties. This iterative process allows the agent to optimise its decision-making strategy over time, aiming to maximise cumulative rewards.

In a recent study [7], Deep Reinforcement Learning (DRL) was employed to enhance the speed and efficiency of virtual IP watermark detection while minimising resource overhead. The DRL model, specifically Deep Q-Networks (DQN), adaptively generated watermark positions based on the structure of the IP circuit, optimising security, circuit integrity, embedding cost, and embedding density. The DQN was trained on the characteristics of watermark positions, aiming to balance watermark security with proximity to the original design, resulting in a high aggregation degree and low detection overhead. This approach utilises reinforcement learning to select the best action, similar to how my project leverages RL to dynamically choose the most effective web server compon-

ents based on real-time data. However, while both methods focus on adaptive decision-making, there are key differences in the environments. The DRL model operates in a static environment, focused on the fixed structure of IP circuits in IoT, while my research works with dynamic, real-time web request patterns that must be classified without predefined categories. Additionally, the DRL agent is trained offline, while my approach emphasises online learning and adaptation. This paper provides valuable insights into reinforcement learning for adaptive decision-making and highlights challenges relevant to my work, particularly the need for dynamic responses in environments where potential states are not known in advance. The concept of adaptive decision-making in response to evolving conditions offers inspiration for addressing the dynamic nature of web request patterns in my project.

In the research on multi-zone ventilation control [8], the authors employ an RL approach with pretraining to enhance efficiency, achieving accurate ventilation control within 7.4% error while improving time efficiency by over 98%. The proposed dynamic-target RL (dtRL) algorithm enables the RL model to continue learning during execution, adapting to changing flow targets. By pretraining in a virtual environment, the model optimally initialises neural network parameters, reducing the computational load during real-world deployment. The reward function incorporates penalties for damper motion and airflow violations while rewarding flow target achievement. This approach shares similarities with my research, as both employ RL for adaptive decision-making in dynamic environments. However, a key difference is that dtRL incorporates state information, whereas my research involves indirectly improving a stateless RL model. Additionally, while dtRL benefits from offline pretraining for optimal parameter initialisation, my approach relies on fully online adaptation. This paper reinforces the importance of RL in adaptive decision-making and highlights a key limitation: RL models trained in a controlled environment may not generalise optimally to unseen conditions. Understanding this limitation helps refine my approach by ensuring the classifier remains effective when encountering new web request patterns.

Another paper explores the use of online reinforcement learning (RL) to optimise job dispatching in edge computing systems, aiming to minimise the total response time of computation-intensive tasks submitted by mobile devices [9]. It employs the UCB1 algorithm, a bandit-based approach, to continuously update dispatching policies in response to dynamic conditions. The reward function is defined based on the average response time of jobs sent to servers. Notably, the paper pre-classifies jobs based on job size, with each class has its own decision-making process to learn optimally for different servers. While my research also employs RL and focuses on adaptive decision-making in dynamic environments, it currently does not include predefined categories for web request patterns. However, insights from this paper suggest that incorporating a classifying step before the reinforcement learning could enhance learning efficiency by allowing the model to handle distinct request patterns more effectively. This work reinforces the necessity of designing an automated classifying mechanism to categorise web request patterns in RL-based decision-making systems, enabling more efficient learning and adaptation.

2.3 Automated Classifier

A classifier is a model that assigns input data to groups based on their features, ensuring that each group shares similar characteristics. An automated classifier enhances this process by incorporating some mechanisms that enable the system to operate independ-

ently without human intervention. This automation can involve selecting optimal parameters, determining category boundaries, or dynamically adjusting the number of groups to improve classifying accuracy and efficiency. Generally, classifiers fall into two categories: classification and clustering. Classification involves training a model on pre-labelled data, allowing it to recognise patterns and assign new inputs to predefined categories. In contrast, clustering works with unlabelled data, grouping similar data points without predefined labels.

In related work on automated classification systems, the study by [10] focuses on classifying the degree of liver fibrosis based on the METAVIR score using B-mode ultrasonography images. The authors developed a deep convolutional neural network (DCNN) to automate the classification process. Their model achieved an accuracy of 83.5% on the internal test set and 76.4% on the external test set, demonstrating its effectiveness in distinguishing between four levels of liver fibrosis. This research aligns with our study in adopting an automated approach that minimises human intervention for categorising data into groups. However while the study utilises images as input data, our research processes numeric or categorical data to represent web request environments. Additionally, the predefined categories of liver fibrosis in the study contrast with our approach, which dynamically identifies groups without prior category definitions. Moreover, the DCNN in their work relies on pretrained models, whereas our method employs online dynamic learning, allowing it to classify unseen web request patterns in real time.

Another study [11] addresses the problem of providing reliable and reproducible heart/mediastinum (H/M) ratio cut-off values for diagnosing parkinsonian disorders. The authors employed two machine learning techniques, Support Vector Machines (SVM) and Random Forest (RF) classifiers, to classify data into distinct groups effectively. Their findings demonstrated that the H/M cut-off value of 1.55 achieved a classification accuracy of 100% for SVM and 98.5% for RF, successfully distinguishing between Parkinson's disease and other related conditions such as parkinsonism and essential tremor (ET). This work shares a conceptual similarity with our research in its goal of classifying data into groups, as the authors focus on disease diagnosis while we aim to classify web request environments. However, there are significant differences in the scope and methodology. In contrast to the predefined classes of Parkinson's disease used in their study, our automated classifying framework does not rely on predefined categories. Instead, it dynamically groups web request patterns without labelling. Moreover, the authors' approach is trained on a static dataset and does not account for unseen categories, whereas our research tackles the challenge of real-time classifying dynamic data without prior training, enabling adaptability to evolving patterns.

In related work on automated clustering systems, one study introduced an innovative automated clustering approach to analyse unlabelled biomolecular structural ensembles, aiming to extract meaningful conformations linked to protein function and mechanism [12]. Specifically, CLoNe computes local densities using nearest neighbours with a Gaussian kernel and identifies cluster centres from local density maxima, then merges clusters based on the Bhattacharyya coefficient to accommodate non-spherical shapes, and removes noise-induced outliers through a Bayesian classifier. This automated determination of cluster centres and dynamic grouping effectively mitigates common issues in clustering, and the underlying principles are directly relevant to our work on automated classification of dynamic web request patterns. However, while CLoNe focuses on automatically forming groups based on density, our work extends this concept by emphasising

pattern recognition within web requests.

Another study introduced an automated clustering approach to enhance the classification of Early-Onset Scoliosis (EOS) patients into clinically relevant subgroups, addressing the limitations of the existing C-EOS system [13]. The study applied Fuzzy C-means clustering, which assigns membership values to each data point, enabling flexible group assignments and outlier detection. The optimal number of clusters was determined based on the highest average membership values, ensuring the formation of statistically distinct subgroups. Unlike expert-driven methods, this data-driven approach objectively categorises patients, enhancing the understanding of disease progression and treatment. The principle of clustering unlabelled data aligns with our work on classifying dynamic web request patterns. While the study predefined etiological categories to ensure clinical relevance, our research automatically determines the number of groups based on emerging web patterns. This prevents the reinforcement learning agent from being confused by shifts in web request patterns, ensuring optimal web server composition. Unlike the EOS study, where the cluster count was manually selected, our approach automates the classifier, dynamically creating new groups as distinct request patterns emerge.

Since our application lacks predefined categories and labels, a clustering approach is more suitable for this project. It allows the model to identify patterns and form groups dynamically based on similarities in web request patterns. This ensures adaptability in recognising emerging request patterns without relying on manual labelling.

In reviewing existing research on self-adaptive systems, reinforcement learning, and automated classifier, most prior works have focused on static or predefined environments. However, there is a significant research gap in the domain of automatically classifying dynamic web request patterns. Addressing this gap, our research aims to improve the performance of web servers in self-adaptive systems by integrating dynamic, real-time classifier with reinforcement learning.

3 Methodology

3.1 Background on Adaptive Web Servers

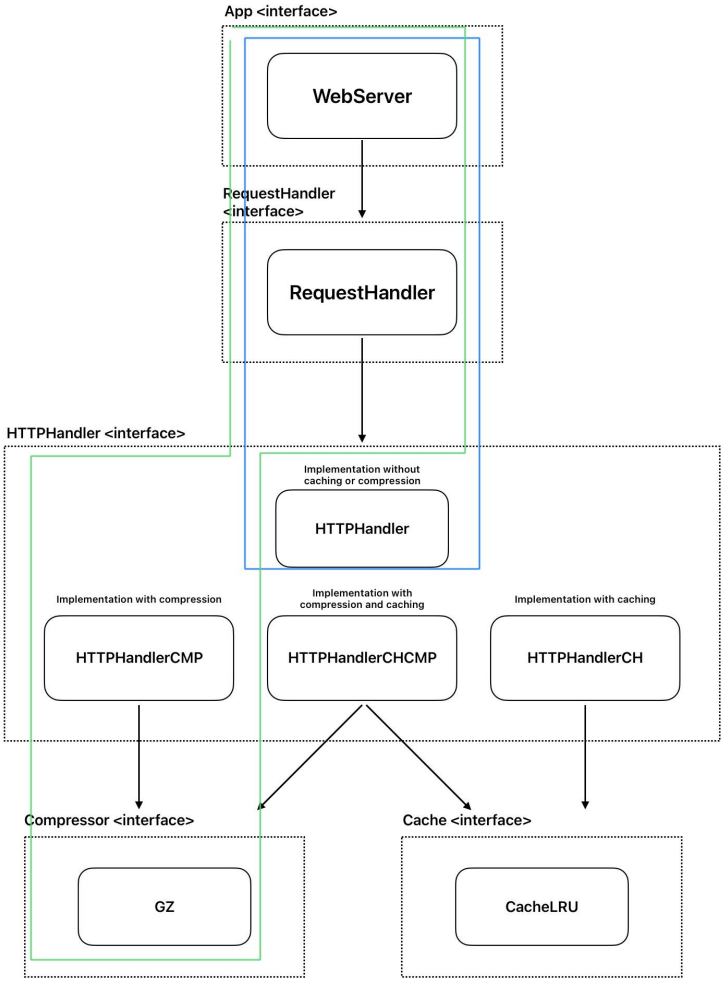

Figure 1: The set of web server composition.

A web server is responsible for handling web requests from clients. When a client requests a file, the web server processes the request and responds by sending the requested file back to the client. Adaptive web servers can be built using different components, which determine how requests are processed and managed.

Figure 1 illustrates the structure of the web server used in this system, showing various components that make up a web server. The blue and green boxes highlight examples of possible web server compositions that can be formed.

- **Dotted-line boxes** represent interfaces.
- **Solid-line boxes** indicate components that implement these interfaces.
- **Arrows** denote the required interfaces for each component.

The key interfaces in the web server are as follows:

- **App Interface**: Opens a server socket, accepts client connections, and passes each connection to a request handler.
- **RequestHandler Interface**: Takes a client socket, applies a concurrency approach, and forwards the socket to the HTTP handler.
- **HTTP Interface**: Processes the client socket, parses HTTP request headers, and generates an appropriate response.

Type	Web Server
0	Default configuration
1	Caching enabled
2	Compression enabled
3	Both caching and compression enabled

Table 1: Web server compositions

Table 1 presents all the possible web server compositions that can be configured. To manage different configurations, the system provides an RESTAPI with the following functions:

- `getAllConfigs()`: Returns a list of available web server configuration strings, where each configuration represents a specific composition of components.
- `setConfig(configString)`: Takes a configuration string representing a web server composition and assembles its components to handle incoming requests.
- `getPerception()`: Retrieves all events and metrics collected by the web server since the last time this function was called.

Additionally, a client program is used to generate web requests following different dynamic request patterns. A *request pattern* refers to the sequence and frequency of incoming requests, and different patterns exhibit different properties. Each web server composition responds differently to various request patterns, some compositions may struggle to handle certain patterns effectively, while others perform better depending on the request characteristics.

To address this, the objective of this project is to classify request patterns automatically. By doing so, the reinforcement learning agent can identify the optimal web server composition for each detected request pattern, improving overall system performance.

To evaluate which web server composition performs best, response time is used as the cost metric, where lower values indicate better performance. However, since reinforcement learning agents typically maximise rewards, the cost metric must be transformed into a reward where higher values are preferable. This is achieved by first normalising the response time:

$$normalised\_rp = \frac{response\_time}{normalisation\_range}$$

where the normalisation range (e.g., 100ms) ensures that the response time remains within a manageable scale. The final reward is then computed as:

$$R = 1 - normalised\_rp$$

For example, if a web server composition results in a response time of **50ms** and the normalisation range is **100ms**, the normalised response time is:

$$normalised\_rp = \frac{50}{100} = 0.5$$

Thus, the reward is:

$$R = 1 - 0.5 = 0.5$$

Similarly, if the response time is **20ms**, the reward is:

$$R = 1 - \frac{20}{100} = 0.8$$

This transformation ensures that lower response times produce higher rewards, allowing the agent to effectively learn which web server composition performs best for each request pattern.

3.2 System Architecture Overview

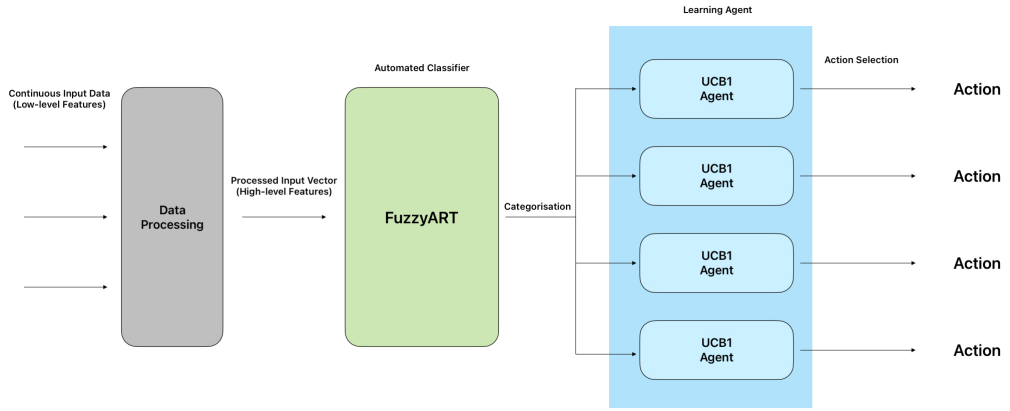

Figure 2: System Architecture

This research develops an automated classifier system to categorise dynamic web request patterns using Fuzzy Adaptive Resonance Theory (FuzzyART). Each identified patterns are then used by a distinct reinforcement learning model based on the Upper Confidence Bound 1 (UCB1) algorithm, a stateless reinforcement learning approach, to determine the optimal web server composition for each request category.

The introduction of the classifying step to form a multi-instance UCB1 system is crucial in addressing learning inconsistencies. Without the classifier, a single reinforcement learning agent would encounter highly variable rewards when exposed to different web re-

quest patterns. This variability arises because different request patterns affect web server performance differently, leading to inconsistent action-reward mappings.

For example, Table 2 represents a single-agent system learning optimal actions based on a specific web request pattern. If the request pattern changes, the reward associated with an action (e.g., Action 0) may vary significantly (e.g., dropping to 50), confusing the agent and leading to suboptimal learning. Since different web server compositions respond uniquely to various request patterns, separating request patterns into distinct categories ensures that each UCB1 instance learns in a stable and consistent environment.

Action	Reward
0	90
1	50
2	60
3	80

Table 2: Example of a single learning agent's knowledge based on a specific web request pattern. Actions represent different web server compositions, while rewards indicate the performance of each composition in handling the request pattern.

The methodology follows a three-stage design, as illustrated in Figure 2:

1. **Data Processing**: Continuous web requests are processed to extract key features that characterise different request patterns. Low-level features are transformed into high-level representations, which are then encapsulated into a single feature vector prepared as input for the classifier.
2. **Automated Classifier**: The FuzzyART model categorises dynamic web request patterns into distinct groups based on their characteristics.
3. **Learning Agent Integration**: A UCB1 agent is assigned to each detected category to determine the optimal action (web server composition) for each request pattern.

3.3 Data Processing

The information on file requests is retrieved from the REST API using `getPerception()` at regular time intervals. From this data, several low-level features are extracted:

- Total text size
- Total image size
- Type of requested files
- Total response time

To make these features more informative, the sizes are averaged over the time interval, providing the average request file size for each period.

Additionally, this system introduces entropy as a high-level feature to classify web request patterns. Entropy measures the randomness of file requests within a given period, providing insight into whether requests follow a predictable or diverse pattern.

- Higher entropy → More diverse file requests
- Lower entropy → More predictable, repetitive file requests

While adding entropy enhances the classifier's ability to distinguish request patterns, it introduces additional computational overhead. Entropy calculation requires processing the distribution of requested files within each interval, which adds a layer of complexity. However, this trade-off is considered worthwhile for improving pattern recognition.

The transformed high-level features including averaged text size, image size, and entropy are then combined into a one-dimensional feature vector, which serves as the classifier's input. The reward for the UCB1 learning agent is prepared by normalising the average response time for each period as mentioned in Section 3.1.

Hypothesis and Justification for Entropy Selection
Entropy is selected as a key feature based on the hypothesis that low-entropy request patterns may perform better in cache-enabled web compositions. The underlying assumption is that web servers with caching mechanisms are more efficient when handling repetitive file requests (low entropy) since frequently requested files are stored and served quickly.

3.4 FuzzyART

Adaptive Resonance Theory (ART) is an artificial neural network model designed for pattern recognition and prediction. It is an unsupervised machine learning technique often considered a clustering method. ART models categorise input data into groups based on similarity. One of the key strengths of ART is its ability to automatically create new categories when encountering novel input patterns. It is designed to be open to learning new information while retaining previously learned data.

Variants of ART
ART has several variants, each extending the basic architecture for different use cases:

- ART1: The simplest form of ART, which handles binary input (0 or 1).
- ART2: An extension of ART1 that can process continuous input data.
- FuzzyART: A combination of fuzzy logic and ART, enabling the model to handle noisy and ambiguous continuous inputs.
- ARTMAP: A supervised learning model where one ART model learns based on a previous ART model to perform classification.

Other variants include Fuzzy ARTMAP and Gaussian ART, which build on the basic ART architecture to address specific problems in pattern recognition.

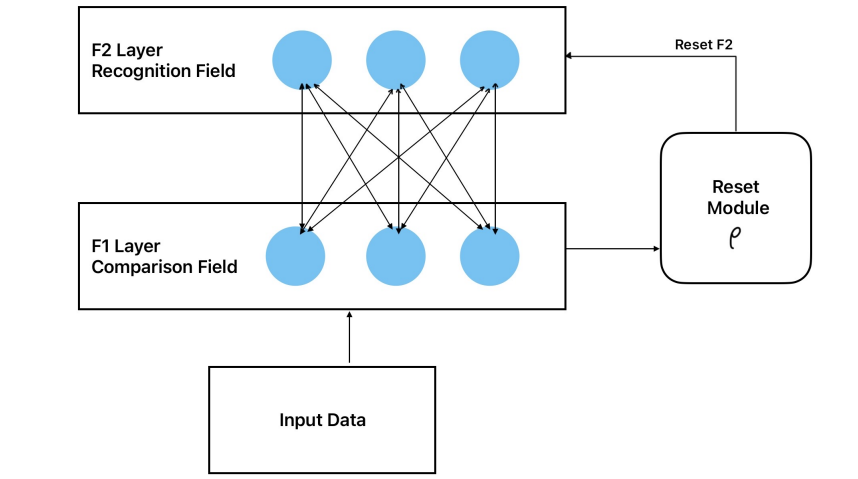

Figure 3: ART Architecture

ART Architecture
The basic ART architecture consists of three key layers, as illustrated in Figure 3:

- F1 Layer (Comparison Field): The F1 layer processes the input as a set of values arranged in a one-dimensional array. It then transfers this data to the F2 layer for further analysis.
- F2 Layer (Recognition Field): This layer contains a group of neurons, each representing a category of input patterns. The "winning" neuron is selected based on the closest match of its weights to the input vector.
- Reset Module: This module compares the dissimilarity between the recognised neuron and the input vector. If the input is within the specified vigilance parameter (ρ), the weights of the winning neuron are adjusted to align more closely with the input vector according to the learning rate (β). If the dissimilarity exceeds the threshold, the neuron is inhibited, and the system continues to search for a matching neuron. If no neuron matches the input, a new neuron is created to represent a new category, and its weights are adjusted according to the input.

Vigilance Parameter (ρ)
The vigilance parameter (ρ) plays a critical role in controlling the strictness of categorisation in the ART model. It defines a threshold that determines whether an input should be grouped with the winning cluster or lead to the creation of a new cluster.

- If the dissimilarity between the winning neuron and the input vector is less than ρ: The input is assigned to the winning cluster, and the network reaches "resonance", meaning it recognises the input as belonging to this category:

$$D \leq \rho$$

- If the dissimilarity is greater than ρ: The input does not match any existing cluster, and a new cluster is created:

$$D > \rho$$

The impact of ρ is as follows:

- A low ρ results in fewer, larger clusters since the network is less strict and groups more inputs together.
- A high ρ leads to many small clusters as the network is more selective and only groups very similar inputs together.

Learning Parameter (β)
Once the vigilance test is passed, the weights of the winning neuron are updated according to the learning rate (β). The learning rate controls the extent to which the weights are adjusted toward the new input. A value close to 0 results in minimal updates, while a value close to 1 leads to significant changes in the weights.

The weight update is performed using the component-wise minimum between the input and the weight vector. This ensures that only the shared features are reinforced. The formula for the weight update is:

$$w_{new} = w_{old} \wedge x$$

where:

- w_{old} is the weight vector of the winning neuron.
- x is the input vector.
- $\wedge$ represents the component-wise minimum between the input vector and the weight vector.

Cluster Creation
If the vigilance test fails, no weight update occurs. Instead, a new cluster is created, with the current input as the centre of this new category, and its weights are initialised accordingly.

After the inputs inserted to the FuzzyART, the final category assigned for the input is returned.

Alternative Approaches Considered
To determine the best approach for this project, Principal Component Analysis (PCA) and Density-Based Spatial Clustering of Applications with Noise (DBSCAN) were considered. PCA is a dimensionality reduction technique that preserves significant data patterns while reducing computational complexity and eliminating redundant features. However, since this experiment only involves three features, PCA provides minimal benefits and is better suited for feature reduction rather than pattern recognition. While it was not implemented, PCA may be useful in future research if additional features are introduced.

DBSCAN is a density-based clustering algorithm that groups data points based on spatial density. It does not require a predefined number of clusters, effectively identifies clusters of arbitrary shapes, and detects outliers. However, it relies heavily on data density, which can lead to misclassification if patterns are widely dispersed, and it lacks the ability to continuously learn new patterns.

Justification for Choosing FuzzyART
ART is chosen for this approach because of its ability to classify data based on similarity while dynamically creating new categories for unseen patterns, while exhibiting online learning. It can adapt to dynamic web request patterns without forgetting previously learned categories. The FuzzyART variant is selected because web requests often exhibit gradual variations and uncertainties. Unlike strict classifying methods, FuzzyART allows web requests to belong partially to multiple categories, which better reflects real-world scenarios where request patterns are not always distinct.

3.5 UCB1 Integration

UCB1 is a reinforcement learning algorithm designed to solve the Multi-Armed Bandit (MAB) problem. The objective of the MAB problem is to maximise the total reward by selecting the optimal action from a set of k possible actions. Each chosen action yields a reward, and the challenge lies in balancing **exploration** and **exploitation** to make optimal decisions.

Exploitation: The agent selects the action that currently has the highest estimated reward, aiming for short-term gains. By consistently choosing the action with the largest estimated value, the agent maximises immediate rewards.

Exploration: The agent occasionally selects less-explored actions to gather more information about their potential rewards. It does not choose the largest estimated value and may involve in sacrificing short-term gains in the hope of discovering an action that yields higher long-term rewards.

To balance these competing objectives, UCB1 assigns a confidence interval to each action and selects the action with the highest upper bound. If an action is uncertain, it has a wider confidence interval, resulting in a higher upper bound, making it more likely to be explored.

UCB1 Formula
At each time step t, the agent selects the action A that maximises the Upper Confidence Bound (UCB) score, given by:

$$UCB1(A) = Q(A) + \sqrt{\frac{2 \ln t}{N(A)}}$$

where:

- $Q(A)$ is the current estimated action-value (mean reward for action A).
- t is the total number of times any action has been selected.
- $N(A)$ is the number of times action A has been chosen.
- $\sqrt{\frac{2 \ln t}{N(A)}}$ is the confidence term, which encourages exploration of less frequently chosen actions.

After selecting an action, the algorithm updates the estimated reward and count for that action, dynamically balancing exploration and exploitation over time.

The Multi-Armed Bandit problem is a widely studied topic in reinforcement learning, with several approaches available. Another well-known method is Thompson Sampling, which utilises Bayesian inference to select actions based on probability distributions. An example use case can be found in [14].

In this work, UCB1 is utilised to determine the optimal web server composition for dynamically changing web request patterns. A separate UCB1 agent is assigned to each detected category of web request patterns. When a new category is identified, a new UCB1 agent is instantiated to learn the optimal composition for that pattern.

Once the classifier assigns an incoming web request pattern to a category, the corresponding UCB1 agent is activated. This agent continuously updates its knowledge based on the last action (web server composition) used to handle the web request and its corresponding reward. The reward is derived from the normalised response time of handling the request. Over time, it learns to predict the best action for handling future web requests within that category.

4 Implementation

This section presents Python code snippets illustrating how the system operates. The main method, as described in Algorithm 1, initialises and runs the system.

Algorithm 1 Main Method

```
1: // Initialise the system and execute
2: system = System()
3: system.run()
```

Algorithm 2 corresponds to the `run()` method of the System. The key steps in its execution are as follows:

- **Lines 3–5:** The system initialises the first learning agent for training. The status of all learning agents is maintained in the dictionary `rl_instances`.
- **Line 12:** The action selected by the learning agent is sent to the REST API, which configures the web composition to handle the file request.
- **Line 19:** The system processes perception data received from the REST API by extracting low-level features and transforming them into high-level features to be classified as described in Algorithm 3:
  - The function `extract_event_data()` extracts and averages the image and text size.
  - The entropy of resource requests is computed using the formula in Algorithm 4.
  - The extracted image size, text size, and entropy are combined into a single feature vector.
- **Line 22:** The high-level feature vector is fed into the FuzzyART model, which determines the request pattern category, as detailed in Algorithm 5.
- **Line 25:** If a new request pattern category is detected, a new UCB1 agent is initialised and assigned to the corresponding category.
- **Line 31:** The reward for the current action is calculated using Algorithm 6, which normalises the response time and transforms it into a reward.
- **Line 34:** The learning agent's status, including the selected action, reward, and response time, is updated in the `rl_instances` dictionary. Additionally, the normalisation range of rewards for recent response is adjusted based on Algorithm 7.
  - By default, the normalisation range is set between 0-100ms.
  - If the bottom 25% of the last 20 response times exceeds 50ms, it suggests that response times for this request pattern are consistently high, regardless of the action taken.
  - In such cases, the normalisation range is adjusted to 0-1500ms.

  This adjustment prevents most reward values from being reduced to 0 after normalisation when response times are consistently high. The updated normalisation range ensures that reward calculations remain meaningful and appropriately scaled for learning.

- **Line 37:** After updating the action and reward, the current learning agent selects the new optimal action for handling subsequent web requests.

Algorithm 2 Run() in System

```
// initialise first learning agent
iteration = 0
self.rl_agent.initialise_category_rl(0)
current_rl, current_actions, current_rewards, response_times, min_norm, max_norm, features = self.rl_agent.rl_instances[0]
current_config_index = current_rl.predict()

// Start classifying and learning
while iteration < MAX_ITERATIONS do

    // Set selected action (web composition) to RESTAPI to handle web request
    current_config = self.configs[current_config_index]
    api_client.set_config(current_config)

    // 2 second time interval to receive web request
    time.sleep(2)

    // Process perception data from RESTAPI
    perception_data = self.api_client.fetch_perception_data()
    features = self.data_processor.process_perception_data(perception_data)

    // Classify web request into categories
    category = self.train_and_categorise(features)

    // Initialise new learning agent for new category detected
    if category ∉ rl_agent.rl_instances then
        rl_agent.initialise_category_rl(category)
    end if

    // Calculate reward for RL agent
    min_norm, max_norm = self.rl_agent.rl_instances[category][4][5]
    reward = self.reward_calculator.calculate_reward(perception_data. get('metrics', []), min_norm, max_norm)

    // Update status of RL Agent
    current_rl = rl_agent.update_rl_instance(category, current_config_index, reward, response_time, features)

    // Rl agent select optimal action
    current_config_index = current_rl.predict()

    iteration++

end while
```

Algorithm 3 Process Perception Data

```
image, content_length_image, text, content_length_text = self._extract_event_data(perception_data)
resources = self._extract_resources(perception_data)
entropy = self._calculate_entropy(resources)
return np.column_stack(( content_length_image, content_length_text, abs(entropy) ))
```

Algorithm 4 Calculate Entropy

```
counts = Counter(resources)
total = len(resources)
probabilities = [count / total for count in counts.values()]
return -sum(p * log2(p) for p in probabilities if p > 0)
```

Algorithm 5 FuzzyART

```
// Iterates over all existing learned categories
for each category c_i in categories do
    // Computes match value between input_vec and current category
    match = self._compute_match(input_vec, category)

    // If it exceeds the vigilance threshold, the input is similar enough to the category
    if match >= vigilance then

        // Updates the category representation using weighted learning
        self.categories[i] = ( self.learning_rate * self._fuzzy_and(input_vec, category) + (1 - self.learning_rate) * category)
        return i                                   ▷ Return index of matched category
    end if
end for

// If no category matched, creates a new category
self.categories.append(input_vec.copy())
return len(self.categories) - 1                    ▷ Return index of new category
```

Algorithm 6 Calculate Reward

```
// Set normalisation range
self.low = min_norm
self.high = max_norm

// Calculate average response time of web request
first_metric = metrics[0]
value = first_metric.get('value', 0)
count = first_metric.get('count', 0)
response_time = value / count

// Normalise response time and transform to reward
normalised = response_time / self.high
reward = 1.0 - cost
return reward
```

Algorithm 7 Update Normalisation Range

```
if len(response_times) ≥ 20 and len(response_times) mod20 = 0 then
    recent_times = response_times[-20:]
    if np.percentile(recent_times, 25) > 50 then
        min_norm = 0
        max_norm = 1500
    else
        min_norm = 0                                   ▷ Default value
        max_norm = 100                                 ▷ Default value
    end if
end if
```

5 Results

This section presents the findings of the experiment, beginning with the experiment setup, which describes the request patterns, the experimental procedure, and the parameters used for both FuzzyART and UCB1 algorithms. Following this, the ground truth is examined to assess how different web server configurations perform under each request type based on response times. Next, the performance of a single UCB1 agent is evaluated, highlighting its difficulty in adapting to changing patterns and the need for classifier. The next section explores the effectiveness of the FuzzyART classifier to group request patterns into distinct categories. Finally, the benefits of multiple-instance UCB1 agents are analysed, demonstrating improved learning efficiency and optimised web server response times.

5.1 Experiment Setup

Five different request patterns were prepared:

1. **var_big_img**: A collection of different large-sized images.
2. **var_med_img**: A collection of different medium-sized images.
3. **big_img**: A large-sized image file.
4. **med_img**: A medium-sized image.
5. **small_txt**: A small-sized text file.

For each request pattern, the contents were repeatedly requested to the RESTAPI for **20 seconds**. The sequence of request patterns was repeated in a loop **15 times**, following the order:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \ldots \text{ (repeated 15 times)}$$

When the system starts, it calls `get_perception()` from the REST API at a time interval of **2 seconds**. Since each request pattern lasts **20 seconds**, information is retrieved approximately **10 times per request period**.

The parameters for **FuzzyART** were configured to best classify the set of web request patterns, ensuring neither over-classify nor under-classify, as follows:

FuzzyART Parameter	Value
Vigilance	0.5
Learning rate	0.5

Table 3: FuzzyART parameters

The **UCB1** algorithm was initialised with four actions representing different web server compositions: Action 0 corresponds to the default configuration; Action 1 enables caching; Action 2 enables compression; and Action 3 applies both caching and compression, all with an initial reward of 0.5.

UCB1 Parameter	Value
Actions	[0, 1, 2, 3]
Initial rewards	[0.5, 0.5, 0.5, 0.5]

Table 4: UCB1 initialisation values

5.2 Groundtruth

To evaluate whether the learning agent is learning for each request pattern, the groundtruth of the web server is examined. The groundtruth demonstrates how the web server responds to each request pattern. Table 5 presents the response times (in milliseconds) for different web servers actions corresponding to each request pattern. The results indicate that for the **big_img**, **med_img**, and **small_txt** request patterns, Action 1 yields the lowest response time. This finding supports the hypothesis that low-entropy request patterns perform well in cache-enabled web server compositions as the same file is repeatedly requested during the request period. Notably, for the groundtruth of the **small_txt** pattern, the response times across all actions are striking similar, with a range of only 1.13 ms. This suggests that, for this pattern, the performance of any web server composition is nearly distinguishable, and thus, any server's response time is less sensitive to variations in this particular request pattern.

Action\Request Pattern	var_big_img	var_med_img	big_img	med_img	small_txt
0	68.31	10.17	32.62	19.78	0.37
1	364.66	22.54	7.68	4.04	0.06
2	1217.00	52.56	31.05	18.00	0.07
3	1067.00	48.89	487.00	287.00	1.19

*Cells highlighted in gray indicate the best action for each request pattern.

Table 5: Groundtruth for each request pattern

5.3 Single UCB1 Agent Performance

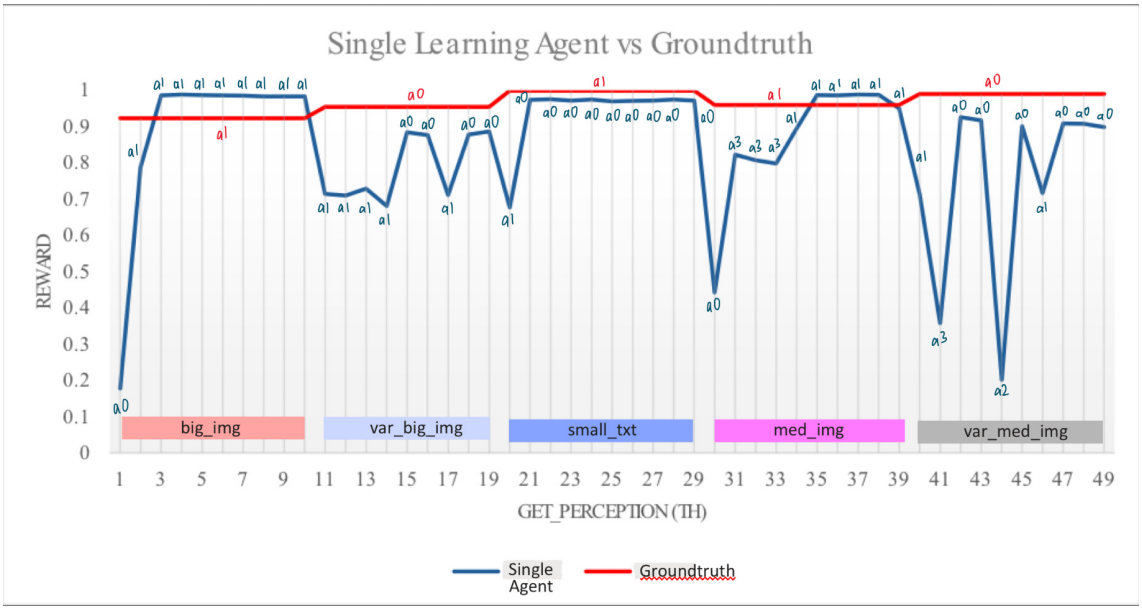

Figure 4: Action Selection of Single Agent vs Grountruth

In Figure 4, the graph illustrates the action selection and respective rewards for both the single agent and the groundtruth in response to a dynamic request pattern over time. The red line represents the groundtruth, showing the expected action and reward for the request pattern, while the blue line represents the performance of the single UCB1 agent, including its selected actions and corresponding rewards. The labels on the blue line indicate the action chosen by the agent at each time point. The default normalisation range is set to 0–100. However, if the response time exceeds 100, the normalisation range is adjusted to 0–1500. This ensures that rewards remain meaningful when a request pattern naturally results in higher response times, regardless of the chosen action, making it the most effective approach for this scenario.

A notable observation is that in the first request pattern, **big_img**, both the ground truth and the single learning agent continuously select Action 1 (a1). However, the single learning agent shows a higher reward compared to the ground truth. This could be due to measurement variability, such as noise or statistical bias. These natural fluctuations occur in real-time performance measurements, even under the same conditions.

The graph demonstrates that when the request pattern switches, the learning agent struggles to identify the optimal action for the new pattern. For instance, when transitioning from **big_img** to **var_big_img**, the agent initially has a high Upper Confidence Bound (UCB1) value to action 1 due to the consistently high rewards it received for selecting action 1 in the **big_img** pattern. As a result, the agent continues to choose action 1, even when it is no longer optimal. Over time, it gradually unlearns this choice and adapts to the optimal action but it requires some time to identify the correct action. This highlights a key limitation: without a classifier to assign different learning agents to distinct request categories, a single agent takes longer time to adjust when the request pattern changes. To address this, FuzzyART is employed to classify web requests based on high-level input vectors, grouping similar request patterns into distinct categories.

5.4 FuzzyART Result

By the end of the experiment, the FuzzyART model successfully identified five categories, each corresponding to a distinct request pattern. This helps separate request patterns, ensuring that each category can be handled by a dedicated learning agent rather than relying on a single agent to adapt across all patterns. The classifying results for the last loop of requests are shown in Figure 5.

The x-axis represents the time at which `get_perception` was called, approximately starting from the beginning of each request pattern, forming a time series. The y-axis represents the category labels assigned to each request pattern.

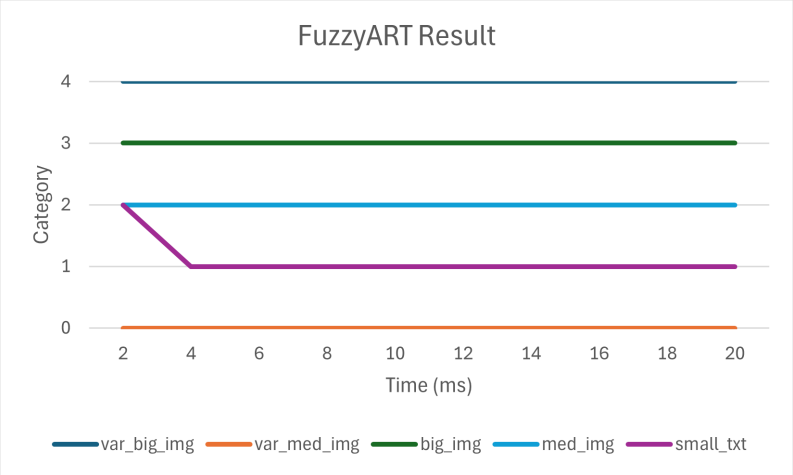

Figure 5: FuzzyART results for the last loop of requests.

The results show that the FuzzyART classifier achieved an high accuracy of 98%. Each of the data points from **var_big_img**, **var_med_img**, **big_img** and **med_img** were correctly assigned to their respective categories. However, for **small_txt**, 9 out of 10 requests were correctly assigned to its category, while the first request was mistakenly assigned to the **med_img** category. This could be due to the 2-second time interval between requests, which likely included the remaining request from **med_img**. As a result, the feature bias might have been influenced by the characteristics of **med_img**, causing the classifier to assign it to the wrong category.

5.5 Performance Evaluation of Multiple-instance UCB1 Agent

In this section, we evaluate the performance of the UCB1 agent for each web request pattern by comparing its selected actions with the ground truth. The goal is to determine whether the classifier helps the learning agent identify the optimal web server composition. We analyse the results from the final loop of web requests.

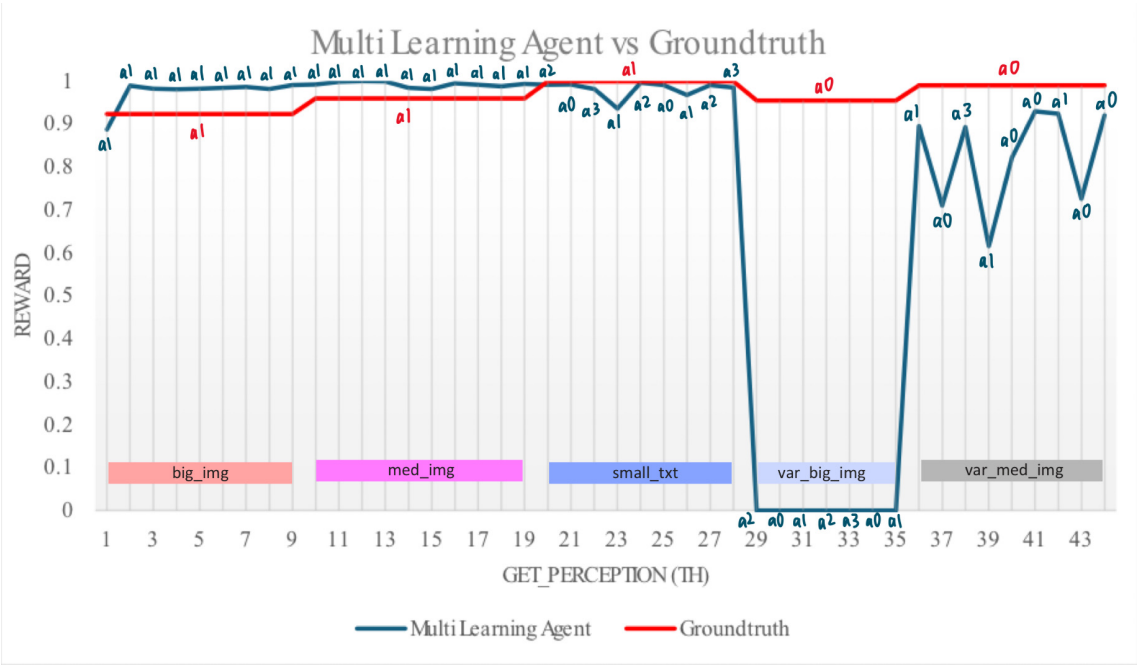

Figure 6: Multiple-instance UCB1 agent without Dynamic Normalisation vs Groundtruth

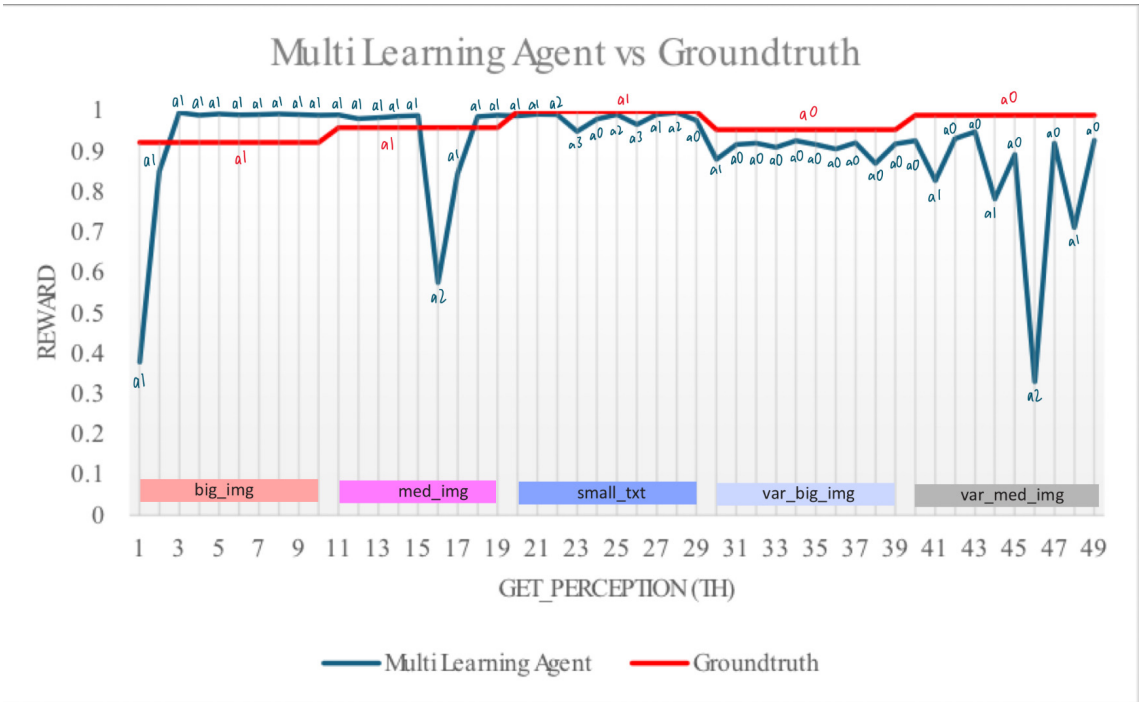

Figure 7: Multiple-instance UCB1 agent with Dynamic Normalisation vs Groundtruth

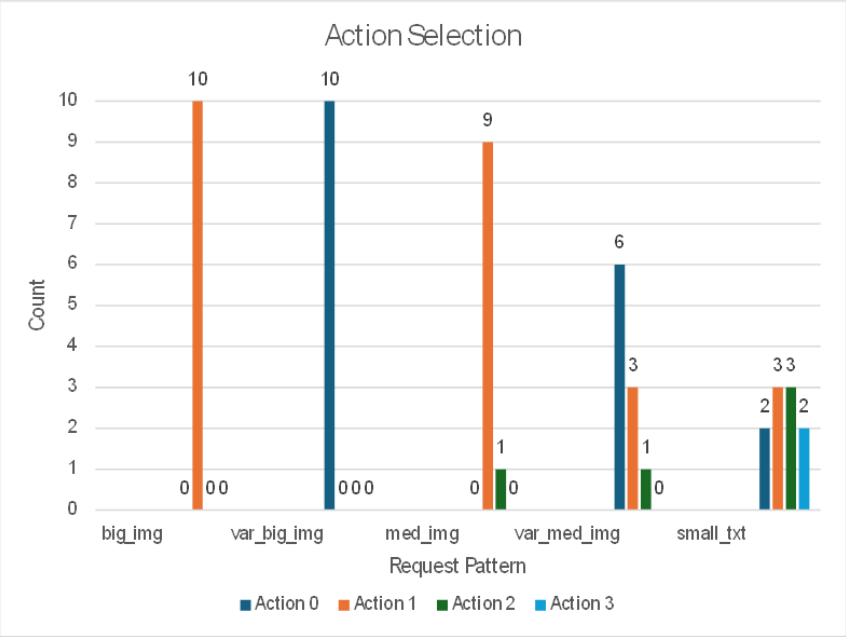

Figure 8: Action selection distribution for all request pattern with dynamic normalisation multi-instance learning agent.

Impact of Dynamic normalisation on Learning Performance
Figures 6 and 7 illustrate the action selection and corresponding rewards for both the multiple-instance UCB1 agent and the groundtruth in response to a dynamic request pattern over time.

- Figure 6 represents the performance of agent without dynamic normalisation.
- Figure 7 represents the performance of agent with dynamic normalisation.

In both figures, the red line represents the groundtruth, showing the expected action and reward for each request pattern. The blue line represents the multiple-instance UCB1 agent, where each request pattern has its own dedicated agent. The labels on the blue line indicate the actions chosen by the agents at each time period, along with their corresponding rewards.

A key observation is that the multiple-instance UCB1 agents in Figures 6 and 7 demonstrate more consistent action selection and rewards compared to the single-agent case in Figure 4. Since each request pattern has its own dedicated agent, there is no need for relearning when the request pattern changes. This allows the multiple-instance UCB1 agents to maintain stable performance and adapt quickly to different request patterns, avoiding the delays seen in the single-agent approach.

Comparing the two figures:

- In Figure 6, for **var_big_img** request pattern, all of the rewards are 0 because the response time exceeds 100. This prevents the learning agent from effectively learning the optimal action, as no meaningful feedback is given. The default normalisation range is 0–100, so when the response time exceeds 100, the reward becomes 0.
- In Figure 7, the learning agent effectively learns the best action for **var_big_img** because the normalisation range is adjusted to higher values, making the rewards more distinguishable.

This highlights the importance of customised normalisation ranges for different request patterns. A properly adjusted normalisation range ensures that rewards are meaningful, allowing the learning agent to learn more effectively and make better decisions.

Action Selection Across Request Patterns
Figure 8 illustrates the distribution of actions chosen by the UCB1 agents for different request patterns in the dynamically normalised scenario (Figure 7).

For the **big_img** and **var_big_img** request patterns, the agent consistently selected action 1 and action 0, respectively, in all 10 trials, this aligns perfectly with the ground truth. With 100% accuracy, the agent demonstrates effective learning in these scenario.

For the **med_img** request pattern, the agent selected action 1 in 90% of cases and action 2 in 10% of cases. Since the ground truth identifies action 1 as optimal, the agent achieved a 90% accuracy rate, indicating strong learning performance. The occasional selection of action 2 is likely due to UCB1's exploration strategy, which seeks to verify whether an alternative action might yield higher rewards.

For the **small_txt** request pattern, the learning agent's action selection is more varied. Having 20% for action 0, 30% for action 1, 30% for action 2, and 20% for action 3. This is consistent with the ground truth, as the response times for different actions are similar. Although no single action dominates, the agent assigns higher probabilities to actions 1 and 2, which have the lowest response times, demonstrating that it is making reasonable choices based on available data.

For the **var_med_img** request pattern, the rewards exhibit high variability, causing the learning agent to fluctuate between selecting action 0 and action 1. Given that action 0 is the optimal choice, the agent achieved 60% accuracy. While this is lower than in other cases, its preference for action 0 suggests it is gradually converging toward the correct decision. Additionally, since action 1 has the second-best response time, the agent's selection pattern indicates ongoing exploration, likely due to the minor reward difference between actions 0 and 1.

To further examine this, an extended experiment was conducted exclusively on **var_med_img** requests. By the 27th loop of the experiment, the agent selected action 0 in 80% of cases and action 1 in 20%, improving its accuracy to 80%. This confirms that the agent requires a longer training period to consistently identify the optimal action.

Overall Performance and Conclusion
The evaluation confirms that each learning agent behaves expectedly for all request patterns. The classifier improves the agent's ability to learn the optimal web server composition for high performance. However, some rewards appear to outperform the groundtruth. Similar to the single-agent case, slight variations in rewards may result from real-time measurement inconsistencies.

Additionally, the total response time for the entire experiment was averaged 130.77% higher without implementing the classifier. This further supports the idea that the classifier enables the system to handle requests more efficiently by ensuring that each request category is processed by a specialised learning agent, reducing the time needed for adaptation.

In conclusion, the results demonstrate that the classifier significantly enhances learning consistency. With a classifier, each category maintains its own dedicated learning agent, ensuring that request patterns are handled optimally regardless of their sequence. This proves the importance of the classifier in achieving stable and reliable decision-making in dynamic web request environments.

6 Conclusion

6.1 Review of Aims

In this project, we designed a classifier model to classify dynamic web request patterns. The aim of this project was to develop a automated classifier model to improve the performance of RL systems. The results of the classifier demonstrate that the features selected, such as text size, image size, and entropy, are significant for categorising web request patterns. Each learning agent was able to effectively learn the optimal web server composition based on these features. In this section, we demonstrated that we achieved the objective.

6.2 Future Work

The study presented in this report identifies text size, image size, and entropy as high-level features for classifying web requests. However, these features alone may not be sufficient for handling more complex and dynamic web requests beyond those considered in this system. Future research could explore additional features to enhance the system's ability to classify diverse request patterns more effectively.

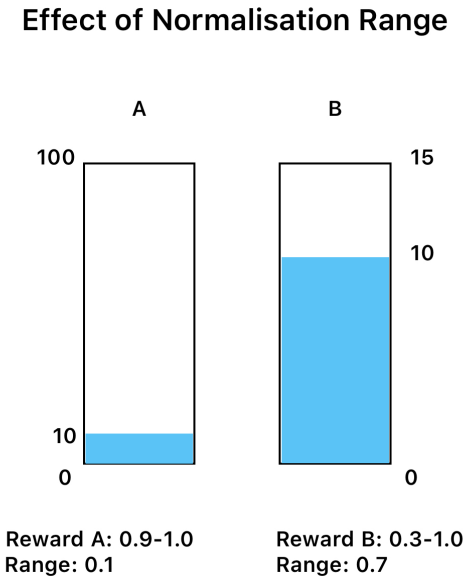

Figure 9: Comparison of different normalisation range

Additionally, the normalisation range of rewards in this system is configured to either 0–100 (default) or 0–1500 (for higher response times). A more dynamic normalisation approach could be investigated to make the learning agent's rewards more meaningful and distinct.

For example, in the case of **small_txt** and **var_med_img**, the difference in rewards between actions is minimal, making it difficult for the learning agent to differentiate between optimal and suboptimal actions. As a result, learning takes longer. The effect of a customised normalisation range is illustrated in Figure 9.

For a request pattern with response times ranging from 0–10 ms, using a 0–100 nor-

malisation range results in rewards between 0.9 and 1.0, with a narrow range of 0.1. This makes it difficult for the learning agent to determine which action is more optimal. However, if the normalisation range is adjusted to 0–15, the rewards span 0.3 to 1.0, with a broader range of 0.7. This larger reward range allows the learning agent to differentiate between actions more effectively, leading to faster and more efficient learning.

6.3 Changes from the Proposal

The study was initially planned to use the ART2 version. However, we discovered that integrating fuzzy logic would be more suitable, as web requests often exhibit gradual variations and uncertainties. As a result, we decided to use FuzzyART instead.

6.4 Learning Outcomes

This project provided deep insights into how web servers handle web requests and how different web server compositions affect the response time. I also gained an understanding of reinforcement learning behaviour in dynamic environments. Throughout the project, I learned how to conduct a research on a topic, review related works, and write a research paper to present project findings. Additionally, I gained experience in showing and analysing results, as well as using LaTeX for report writing. Conducting my third-year project presented various challenges, as it was my first time conducting research and learning new skills. Successfully completing this project has equipped me with the confidence to conduct future research.

6.5 Overall Conclusion

This study implemented a FuzzyART model to classify dynamic web request patterns, enabling the learning agent to better identify the optimal web server composition for handling specific request patterns. This approach contributes to the development of more adaptive and efficient software systems capable of managing dynamic workloads, improving web server performance.

References

[1] M. Chow, D. Meisner, J. Flinn, D. Peek, and T. F. Wenisch, "The mystery machine: End-to-end performance analysis of large-scale internet services," in *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, 2014, pp. 217–231.

[2] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, no. 2, May 2009. [Online]. Available: https://doi.org/10.1145/1516533.1516538

[3] D. Ardelean, A. Diwan, and C. Erdman, "Performance analysis of cloud applications," in *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018, pp. 405–417.

[4] B. Porter, M. Grieves, R. R. Filho, and D. Leslie, "Rex: a development platform and online learning approach for runtime emergent software systems," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'16. USA: USENIX Association, 2016, p. 333–348.

[5] R. Zhang, Y. Xu, Z. Y. Dong, and K. P. Wong, "Post-disturbance transient stability assessment of power systems by a self-adaptive intelligent system," *IET Generation, Transmission & Distribution*, vol. 9, no. 3, pp. 296–305, 2015. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-gtd.2014.0264

[6] A. Rullo, D. Midi, A. Mudjerikar, and E. Bertino, "Kalis2.0 -a secaas-based context-aware self-adaptive intrusion detection system for the iot," *IEEE Internet of Things Journal*, vol. 11, no. 7, pp. 1–1, 2024.

[7] W. Liang, W. Huang, J. Long, K. Zhang, K.-C. Li, and D. Zhang, "Deep reinforcement learning for resource protection and real-time detection in iot environment," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6392–6401, 2020.

[8] C. Cui, C. Li, and M. Li, "An online reinforcement learning method for multi-zone ventilation control with pre-training," *IEEE Transactions on Industrial Electronics*, vol. 70, no. 7, pp. 7163–7172, 2023.

[9] X. Wang, J. Tang, M. Yu, G. Yin, and J. Li, "A ucb1-based online job dispatcher for heterogeneous mobile edge computing system," in *2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications (SSIC)*, 2018, pp. 1–6.

[10] J. Lee, I. Joo, T. Kang, and et al., "Deep learning with ultrasonography: automated classification of liver fibrosis using a deep convolutional neural network," *Eur Radiol*, vol. 30, no. 3, pp. 1264–1273, 2020. [Online]. Available: https://doi-org.ezproxy.lancs.ac.uk/10.1007/s00330-019-06407-1

[11] S. Nuvoli, A. Spanu, M. Fravolini, and et al., "[123i]metaiodobenzylguanidine (mibg) cardiac scintigraphy and automated classification techniques in parkinsonian disorders," *Mol Imaging Biol*, vol. 22, no. 5, pp. 703–710, 2020. [Online]. Available: https://doi-org.ezproxy.lancs.ac.uk/10.1007/s11307-019-01406-6

[12] S. Träger, G. Tamò, D. Aydin, G. Fonti, M. Audagnotto, and M. Dal Peraro, "Clone: automated clustering based on local density neighborhoods for application to biomolecular structural ensembles," *Bioinformatics*, vol. 37, no. 7, pp. 921–928, may 2021.

[13] G. Viraraghavan, P. J. Cahill, M. G. Vitale *et al.*, "Automated clustering technique (act) for early onset scoliosis: A preliminary report," *Spine Deformity*, vol. 11, pp. 723–731, 2023. [Online]. Available: https://doi-org.ezproxy.lancs.ac.uk/10.1007/s43390-022-00634-1

[14] K. Klarich, B. Goldman, T. Kramer, P. Riley, and W. P. Walters, "Thompson sampling—an efficient method for searching ultralarge synthesis on demand databases," *Journal of Chemical Information and Modeling*, vol. 64, no. 4, pp. 1158–1171, 2024.

A Project Proposal

Optimizing Web Server Performance by Identifying Web Request Features and Patterns with ART-2 and Reinforcement Learning

1. Abstract

This project aims to develop a system that classifies web request patterns to enhance web server performance in dynamic environments. This is important in contributing to the development of more efficient and adaptive web server systems. By leveraging an Adaptive Resonance Theory (ART-2) model integrated with a reinforcement learning (RL) algorithm, the system will dynamically select optimal server configurations based on real-time request patterns. The research will focus on identifying the web request features most relevant to server performance and evaluating whether the classifier improves server response times. The expected outcomes include insights into feature relationships and enhanced efficiency in adapting server configurations to dynamic traffic patterns.

2. Introduction

The usage of web services has become increasingly widespread, with demand continuing to rise as more applications rely on web servers to handle user requests by exchanging data with connected devices. Responsiveness is critical in determining user satisfaction, as slow response times directly affect engagement. [1] shows that a mere 1-second delay in webpage loading can lead to a 11% decrease in views, emphasizing the importance of system performance. As the request environment constantly changes, web servers must dynamically adapt to maintain optimal efficiency. However, while reinforcement learning (RL) systems have been employed to optimize server configurations based on request patterns, they often face challenges in quickly adapting when these patterns shift. One of the main challenges in applying RL to web server management is the high convergence time when the environment changes, due to the need to relearn optimal configurations for new request patterns as mentioned in [2]. This project aims to improve server operations by enhancing the performance of RL systems by introducing a classification step that pre-emptively categorizes web requests using Adaptive Resonance Theory (ART). By doing so, RL algorithms can select the most appropriate server configuration more efficiently, avoiding the need for constant relearning.

This project proposal will be split into the following sections: background, the proposed project; the programme of work; the resources required; and references. The background will contain a description about the related research. The proposed project will contain the project's overall aim and its main objectives, along with the methodology approach taken for data retrieval and evaluation. The programme of work will describe the project plan, breaking down the schedule of tasks for the year with the use of a Gantt chart. The resources required section will detail the resources needed within the project and the references section will contain references to any resources and papers used within this proposal.

3. Background

According to research [2], web server performance can vary greatly depending on how the server is configured and the nature of the traffic it handles. Common server configurations include default setups, caching, compression, or a combination of both caching and compression. Each composition tends to perform better under specific traffic patterns. For example, in low-entropy request patterns, configurations that include caching may outperform others. Conversely, in high-entropy environments, some compositions perform better without caching and compression, while others benefit from both features. Research has explored the use of reinforcement learning (RL) to dynamically select the most efficient server composition based on current traffic patterns. However, the RL system in this study faced significant delays due to high convergence times when the web environment changed. This is because RL systems learn based on past experiences in a stable environment, and when the web request pattern changes, the environment effectively shifts, requiring the RL system to relearn optimal configurations for the new conditions, which slows convergence. This indicates the need for a more efficient way to handle dynamic traffic patterns.

To address this challenge, this research draws on Adaptive Resonance Theory (ART), a neural network model known for its pattern recognition capabilities. In [3], ART was applied to chemical process monitoring and demonstrated higher performance compared to traditional function fitting methods. Additionally, ART-2 was used in time-series prediction for electricity pricing in Russia's "day-ahead market" (DAM) in [4], successfully identifying complex price patterns. Given ART-2's success in recognizing patterns in time series data, it is intriguing to investigate whether it could also be applied to classify web request patterns. By integrating ART-2 into the RL framework, the aim is to reduce the convergence time when request patterns change, improving the overall efficiency of server management in dynamic environments.

This project will focus on optimizing the classification of web requests to enhance the adaptability of RL systems in selecting server configurations. This research aims to contribute to the development of more efficient and adaptive web server systems, benefiting industries that rely on scalable web services.

4. Aim & Objectives

This project aims to develop a system that classifies web requests to enhance online reinforcement learning in dynamic server environments using Adaptive Resonance Theory (ART). The following objectives will be involved:

- To determine the features that will affect the performance of web server
- To design and implement a classifier that categorizes web requests based on their characteristics.
- To evaluate the classifier’s impact on reinforcement learning performance across different server compositions.

5. Methodology

5.1 Framework Overview

The proposed methodology aims to classify web requests based on features that allow effective classification using the ART-2 model. The classified web requests are integrated into an existing reinforcement learning (RL) algorithm, which selects the optimal server configuration for given patterns. The study involves training the ART-2 model on different sets of web requests, extracting key features, and analysing the impact of these features on system performance.

5.2 Dataset

A pre-existing dataset of web requests has been provided. This dataset includes combinations of text and image-based requests, simulating typical traffic patterns seen in web environments. A set of potential features from the web requests (e.g., request size, type, frequency, and resource demand) will be identified. These features will be used as input to the classifier to determine how well they can distinguish different web request patterns. For the experiment, we assume that web request patterns are uniform within given time intervals, meaning requests exhibit consistent behaviour over short periods.

5.3 Clustering Using ART-2

ART-2 is selected for its ability to classify patterns and retain learned patterns for future use. It is known for its capability to recognize recurring patterns and adapt to new ones. The features extracted from web requests will be input into the ART-2 model to classify request patterns. The model will be trained in an unsupervised manner, allowing it to group similar patterns without pre-labelled data. The output of the ART-2 model will be analysed to determine whether the identified classes represent meaningful distinctions between web request patterns.

5.4 Integration with Reinforcement Learning (RL)

An existing RL algorithm will be integrated with ART-2 to optimize server configurations based on the classified patterns. The RL system will learn which server compositions perform best for each identified pattern. Each classified web request pattern will have its own RL agent assigned. The RL agent will learn and store the best-performing server configuration for that class. When a new request pattern is classified by ART-2, a new RL agent is created. If a similar pattern reappears, the RL system retrieves the learned state from the previous agent to avoid relearning, improving response time efficiency.

5.5 Performance Analysis

After training the ART-2 model and integrating it with the RL system, we will analyse the performance of the overall system by measuring the server's response time across different web request patterns. The server's response time will be compared against a baseline or ground truth, where the optimal server configuration for each web request pattern is known in advance. This will help determine how well the RL system, assisted by ART-2, is performing. Analysis will be conducted to examine which features contribute most to the classifier's success in grouping web requests. This can provide insight into what characteristics of web requests are most important in driving the RL system’s decision-making.

5.6 Potential Limitations

One potential limitation is the volatility of real-time web request patterns, which may be harder to predict and classify than simulated patterns. The assumption that web request patterns are uniform within short periods may not always hold true, leading to potential classification errors if patterns change unexpectedly.

5.7 Conclusion

This experimental methodology will provide insights into the features that best classify web request patterns using the ART-2 model. It will also examine how integrating this classification into a reinforcement learning system can improve web server performance. By reducing the need for RL systems to relearn for recurring patterns, the study aims to enhance server response times and overall efficiency in dynamic environments.

6. Programme of Work

The project will begin 4th week of Term 1, end of October 2024, running until last week of Term 2 end of March 2025, and it will be broken up into the following stages:

Week 1-2: Problem Definition and Literature Review

- Deep dive into recent papers on Adaptive Resonance Theory-2 (ART-2), reinforcement learning (RL), and web request patterns.

Week 3-4: Feature Selection and Dataset Preprocessing

- Identify relevant features for classifying web requests from the dataset
- Preprocess the data for ART-2.

Week 5-7: ART-2 Integration for Online Clustering

- Implement ART-2 for dynamic classification of new patterns.
- Perform initial runs of the ART-2 model to test its classification of web request patterns.

Week 8-9: Reinforcement Learning (RL) Integration

- Integrate the classified outputs from ART-2 with the existing RL algorithm.
- Assign learning agents for each classified pattern.

Week 10-12: Model Evaluation and Feature Tuning

- Evaluate the performance of ART-2 by analysing the classes it generates.
- Fine-tune the feature set, removing or adding features that improve classification accuracy and relevance.

Week 13-14: Testing and Validation

- Test the integrated model using different web request patterns and monitor server response times.

Week 15: Performance Evaluation

- Analyse the overall response time of the web server configurations and compare with the ground truth.
- Evaluate the impact of the classification on RL.
- Document key findings, challenges, and insights gained from the experiments.

Week 16-17: Final Analysis and Report Writing

- Compile the results and insights from the experiments into the final research paper.
- Include potential improvements and future work suggestions.

The overall schedule for the project in is displayed by a Gantt chart in Figure 1.

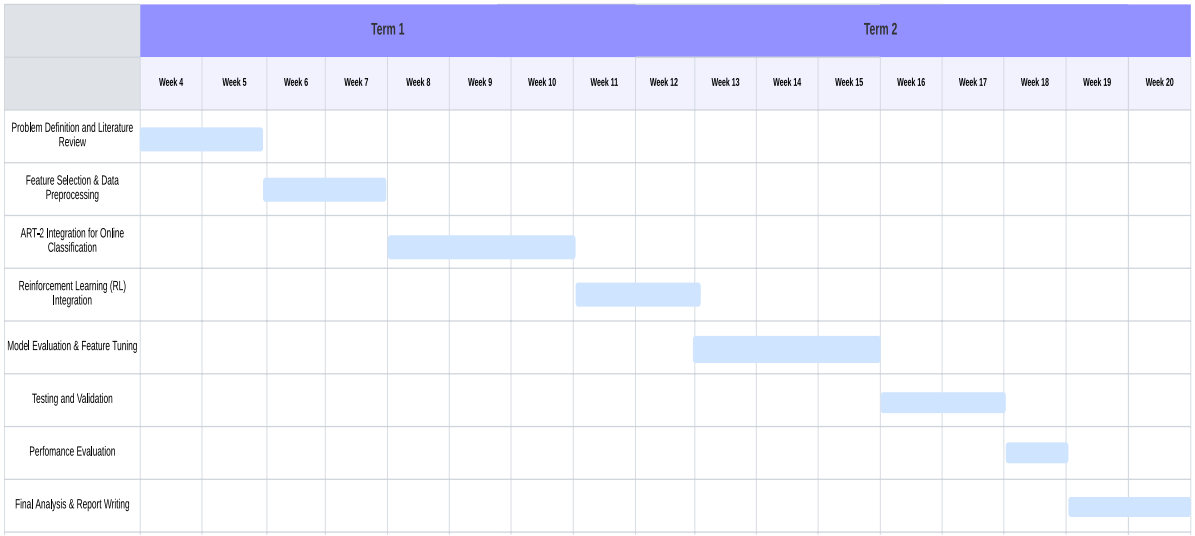

Figure 1: Project Schedule

7. Resources Required

High-performance computing resources are needed to train and test the ART-2 algorithm.

8. References

[1] Envisage Digital, "Website load time statistics: Why speed matters." Envisage Digital. [Online]. Available: https://www.envisagedigital.co.uk/website-load-time-statistics/. [Accessed: Oct. 23, 2024].

[2] B. Porter, M. Grieves, R. Rodrigues Filho, and D. Leslie, "REX: A development platform and online learning approach for runtime emergent software systems," School of Computing and Communications & Department of Mathematics and Statistics, Lancaster University, UK.

[3] D. Wienke and L. Buydens, "Adaptive resonance theory based neural networks — the ‘ART’ of real-time pattern recognition in chemical process monitoring?," TrAC, Trends in Analytical Chemistry, vol. 14, no. 8, pp. 398–406, 1995. doi: 10.1016/0165-9936(95)90918-D.

[4] A. V. Gavrilov and O. K. Alsova, "Time series prediction using the adaptive resonance theory algorithm ART-2," J. Phys. Conf. Ser., vol. 1333, no. 3, p. 032004, 2019. doi: 10.1088/1742-6596/1333/3/032004.