TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Oskar Pihlak 211913IAPM

COOPERATIVE REAL-TIME REINFORCEMENT LEARNING IN A LIMITED DATA ENVIRONMENT

Master's Thesis

Supervisor: Gert Kanter
PhD

Co-supervisors: Riivo Kikas
PhD
Ilja Samoilov
MSc

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Oskar Pihlak 211913IAPM

KOOSTÖÖL PÕHINEV REAALAJALINE STIIMULÕPE PIIRATUD ANDMEKESKKONNAS

Magistritöö

Juhendajad: Gert Kanter
PhD

Kaasjuhendajad: Riivo Kikas
PhD
Ilja Samoilov
MSc

Tallinn 2024

Author's Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Oskar Pihlak

23.05.2024

Abstract

This thesis investigates methods for optimizing message delivery success rates in telecommunication networks and brings examples from short message and multi-media message routing. It uses reinforcement learning, specifically Multi-Armed Bandit algorithms, to solve real-time decision-making problems under uncertainty.

The explorable environment introduces challenges such as limited data and non-stationarity in success rates, as the effectiveness of different message delivery paths can change over time. The integration of federated learning principles is investigated to enable collaborative learning among multiple agents.

Various algorithms are designed, implemented, simulated, and novel Multi-Armed Bandit variants are proposed, termed "Collaborative Memory Sharing UCB" and "UCB Monitored." These variants enhance the adaptability and performance under the described requirement space.

By constructing a simulation framework and running subsequent experiments using synthetic datasets, the work demonstrates the effectiveness of cooperative strategies in improving message delivery success rates compared to isolated agent approaches. The findings have practical implications for optimizing message routing in real-world telecommunication networks.

The thesis is written in English and is 59 pages long, including 9 chapters, 22 figures, and 16 tables.

Annotatsioon

Koostööl põhinev reaalajaline stiimulõpe piiratud andmekeskkonnas

Magistritöö uurib meetodeid sõnumite edastamise edukuse määra optimeerimiseks telekommunikatsioonivõrkudes ning toob näiteid lühisõnumite ja multimeediasõnumite suunamisest. Töö kasutab stiimulõppe alla kuuluvaid *Multi-Armed Bandit* algoritme, et lahendada reaalajas otsuste tegemise probleeme.

Uurimiseskkond toob esile väljakutsed nagu piiratud andmed ja edukuse määrade mittestatsionaarsus, kuna erinevate sõnumite edastamise tõhusus võib aja jooksul muutuda. Uuritakse föderaalse õppe põhimõtete integreerimist, et võimaldada agentide vahelisel koostööl põhinevat õpet.

Erinevaid algoritme disainitakse, implementeeritakse ja simuleeritakse. Pakutakse välja uusi *Multi-Armed Bandit* variante, mis on nimedega *"Collaborative Memory Sharing UCB"* ja *"UCB Monitored"*. Antud variandid suurendavad kohanemisvõimet ja jõudlust kirjeldatud nõuetes.

Simulatsiooniraamistiku loomise ja sellele järgnevate eksperimentide läbiviimisega sünteetiliste andmekogumite abil demonstreeritakse koostööstrateegiate tõhusust sõnumite edastamise edukuse määrade parandamisel võrreldes isoleeritud agentide lähenemisega. Tulemused omavad praktilisi rakendusi sõnumite suunamise optimeerimisel reaalsetes telekommunikatsioonivõrkudes.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 59 leheküljel, 9 peatükki, 22 joonist ja 16 tabelit.

List of Abbreviations and Terms

AWS	Amazon Web Services
CMAB-DO	Contextual Multi-Armed Bandit with a Dominant Objective
CT	Control Theory
CxMAB	Contextual Multi-Armed bandit
EC2	Elasic Compute Cloud
ETC	Explore-then-Commit
FCB	Federated Contextual Bandits
FedIGW	Federated Inverse Gap Weighing
Fed-PE	Federated Phase Elimination
FL	Federated Learning
FMAB	Federated Multi-Armed Bandit
FS	Feedback Supplier
IGW	Inverse Gap Weighing
LLM	Large Language Model
MAB	Multi-Armed Bandit
MDP	Markov Decision Process
ML	Machine Learning
MMS	Multimedia Messaging Service
RL	Reinforcement Learning
SL	Supervised Learning
SMS	Short Message Service
SMPC	Secure Multi-Party Computation
TS	Thompson Sampling
UCB	Upper Confidence Bound
UI	User Interface
VFL	Vertical Federated Learning

Table of Contents

List of Figures

List of Tables

1 Introduction

Reinforcement learning (RL) is a machine learning (ML) paradigm in which agents learn and adapt based on their environment's feedback. The agent is an entity that makes decisions about what actions to take and is considered real-time if it can immediately react to any feedback upon its arrival. This thesis focuses on an RL sub-domain where the agents are primarily algorithmic. Algorithms are instruction sets with predefined steps for solving a problem.

To make different agent comparisons more accurate, they are run through simulations for the same period and on the same test cases. The agent aims to make the best possible decisions and receive the best results. As the same decisions yield varying results over time in this work, it must continuously update its knowledge of the success rates for all options to maintain the likelihood of choosing the best option. Achieving the most optimal result requires balancing exploiting the most effective option to optimize outcomes and exploring alternative options to ensure the optimal choice is being made.

This thesis investigates optimal decision-making in the telecommunications field. This can be exemplified by sending a short message (SMS) or multimedia message (MMS), as seen in Figure 1. The sender sends an SMS to a recipient through a platform like Plivo, Twilio, Infobip or Sinch. The platform attempts to choose the most optimal path for sending the message to the destination with the highest likelihood of delivery.

A messaging platform is a service that enables the sending, receiving, and management of text, voice, and multimedia messages between users or systems across various communication channels. Messaging aggregators are organizations that use technology to distribute messages to vendors. It's possible to send the SMS through different vendors regardless of the vendor used by the sender.

The goal of the messaging platform is to choose the best vendor at the given moment to maximize the likelihood of successful message deliveries. In this setting, the receiver is almost always an individual, while the sender could be either an individual or a business entity. After the SMS is sent to a vendor, the messaging platform that sent the message can receive feedback on whether the message was successfully delivered, depending on the delivery status trustworthiness level, which isn't discussed in the thesis.

Such a platform approach can also be applied to email delivery. The email can be sent to the recipient via different vendors like Brevo, Hubspot, Mailchimp and Woodpecker. In this case, the successful feedback signal for a sent-out email would be if the email reached the inbox in a timely manner and the end-user opened the message.

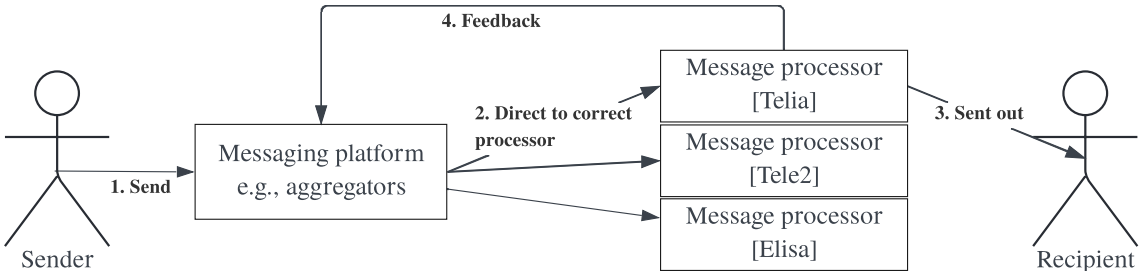

Figure 1. Message sending between entities.

This thesis will provide an overview of related work and look at possible approaches, disregarding suboptimal variants. Then, construct a system to compare viable methods and develop novel variants that better fit the given environment setting if necessary.

The primary challenges encountered include maximizing optimal decision-making over time, gathering knowledge for options with limited information, handling sudden shifts in option results throughout time, and sharing correlating information between agents. These issues highlight the need to investigate approaches that align with the evolving environment.

Algorithm analysis provides an overview of possible solutions. The proposed algorithms are more optimal for environments with data characteristics similar to those described in the work.

The author of this thesis identified and investigated existing solutions and developed a simulation system to validate the approaches empirically. Although this does not guarantee optimal performance on an absolute and infinite scale, the empirical results give sufficient confidence to test these solutions with real traffic.

1.1 Problem

The goal of this thesis is to investigate methods for sending messages to end users via paths to maximize successful delivery rates. Various options are available for doing this. Feedback is received on the message delivery results from specific providers known as feedback suppliers (FS). Continuously adjusting the decisions based on this input ensures

a higher quality of message deliveries. The objective is to select the path that provides the best possible quality at that point in time. However, messages must periodically be sent through each available path to make the best decision. Certain environmental constraints include some feedback suppliers not providing data for all paths or not allowing traffic termination for specific routes.

The trained and evaluated models will be used to improve message routing. The telecommunication context was chosen because it poses a peculiar data environment where message delivery success rates vary in time. The success rates also depend on the destination country, network, and telecommunications operator chosen for message delivery. The problem can be formulated as finding the best single-step path to reach a desired goal.

Table 1 shows the connection between FS requests and the mean success rate for providers P handling the requests. An FS is a client with requests whose termination statuses can be accurately determined. For the context of the thesis, the terms FS and client will be used interchangeably, but in reality, not all clients are FS. Clients whose request statuses cannot be determined will not be discussed in the thesis; such clients are handled differently from what the thesis discusses. A provider is an entity that has connections to multiple Mobile Network Operators (MNO); a provider itself can also be an MNO.

The bottom row of Table 1 shows how many messages clients have sent within an arbitrary time frame. The table has a cell marked with an "?" indicating that FS3 requests have not been sent through provider P2 in the arbitrary time frame or too few requests have been sent to make value predictions. There could be $0 \leq N \leq P \times FS$ connections marked with "?". When a new agent version is deployed, its state starts with all cells $P \times FS$ marked as "?". Most of them get replaced relatively quickly with the numerical mean success rate of the provider as time goes on and possible paths are explored.

Table 1. Success rate connection between providers and clients.

	FS1	FS2	FS3	FS4
P1	0.9	0.6	0.7	0.8
P2	0.8	0.57	?	0.6
P3	0.88	0.55	0.8	0.66
Volume	10k	1k	100	10

In a setting where multiple RL agents are learning from proximally similar datasets and potentially with differing goals, sharing experiences between the agents should be possible. This would help solve the data-deprived option marked with "?". The table contents can

be simplified and represented as the Figure 2 graph.

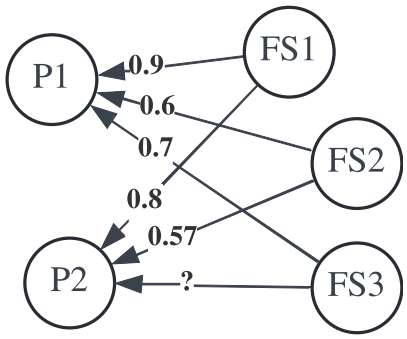

Figure 2. Success rate connection between providers and clients as a graph.

Sharing experiences could lead agents to share conflicting data. As seen in row P3, the mean success rate differs widely per client. This poses the need to check for value correlation between agents before sharing data and ignore agents whose data dramatically differs. Despite this, the message request volumes must be considered before disregarding an agent's success rates per provider.

Conflicting data can exist for a multitude of reasons. Clients send SMS and other message requests to the platform with varying levels of quality and content. Covering some examples, they send requests to numbers without specifying the country prefix (+372 in Estonia) or send message requests with content that gets blocked down the line by a provider. Contact numbers can go out of use, and as a result, the mobile network where the message is routed fails to deliver the request. A mobile phone is turned off for a prolonged period and can't receive the message. In addition, the providers have differing capabilities in handling the requests.

In more complex cases, it can be taken into account that a considerable number of providers operate in a so-called grey area, sometimes engaging in practices such as SIM box routing or traffic throttling, which can artificially inflate or deflate delivery rates and lead to significant variance across clients. Some systematically drop messages, creating a situation where there is a considerable variance of successful deliveries for a provider across clients; the reasons and further background on this aren't discussed.

A question might arise about whether it is worth solving an unknown connection "?" between some P and FS that don't get enough traffic. Information about all connections must be available to make an optimal decision. In the event of data drift, knowing the "?" can be beneficial for maintaining a higher success rate.

Training agents to meet specific objectives is important, as different clients have differing priorities. For example, those sending out high-traffic volumes often prioritize the cost per message. Meanwhile, the speed of message delivery is necessary for those handling login and authentication requests. Reliability in message delivery becomes important when sending order confirmations.

RL systems, such as recommenders [1] can be implemented using Multi-Armed Bandits (MAB) [2], which have various use cases in Google [3], Meta [4], Microsoft [5] and Netflix [6], who have also published papers on the topics. Analogous systems apply to multiple fields.

- In marketing, ad placement optimization involves continuously testing different ad creatives to identify and serve effective ads to various segments [7];
- In e-commerce, product display optimization focuses on testing different product layouts or features on the website to maximize user engagement and sales [8];
- For content streaming, content placement optimization entails testing different content or thumbnails to increase user engagement and viewing times [9];
- In the gaming industry, feature testing involves testing different game features or mechanics to determine which ones enhance player engagement and retention [10];
- Social networks continuously test the placement and type of ads shown in user feeds to improve ad performance [11];
- Experimenting with various teaching methods or engagement strategies aims to enhance student participation and retention [12].

The focus is on optimizing message delivery success rates, considering immediate performance and factors such as the potential for changes in provider reliability over time (data drift).

1.2 Thesis Objectives

The goal of this work is to research and develop algorithms for maximizing successful message delivery rates in telecommunication networks. The thesis will create a simulator to explore and implement numerous Multi-Armed Bandit and baseline models described in the literature. It will then compare the results and conclude.

The research questions that will be tackled in this thesis are the following:

1. Considering the baseline of existing approaches and model designs in the given data setting, can a novel variant be created that is on par with or exceeds the performance

of existing solutions?
   (a) How can real-time models improve performance by learning from adjacent models set in the same data environment but receiving different data chunks?
2. What methods are optimal in the given environment?
   (a) What approaches can improve model change detection in negative and positive cases?

1.3 Design Goal and Requirements

This thesis aims to construct methods that meet the following statements.

1. Convergence in the thesis means reaching the optimal message delivery success rate value. The goal is to achieve quick convergence to optimal actions within at most 50 sampling rounds or 10.000 samplings;
2. Maintain an overview of the relevant options in a short period, having an overview of at least the best two arms within a time frame of 5000 samplings and an overview of all arms within 40.000 samplings;
3. Stable performance on stationary periods, with 6% of the decisions deviating from the optimal decision over the indefinite timespan. This value is derived from the quality threshold of 94%, which is based on analysing real data and considering the expected variance. This value is explicitly stated to give a rationale for exploration even when the period is more stationary.
4. Ability to make real-time decisions;
   (a) Computationally resource-efficient enough to be runnable on a laptop with some clients with their options and runnable on the cloud with a single machine that handles the entire client space with option calculations in under 5 minutes;
   (b) Quick adaptations to reward variability;
5. Robust enough to handle noise and outliers in the reward signals, ensuring that the decision-making process is not unduly affected by anomalous data;
6. Configurable and customizable via hyperparameter tuning;
7. Adaptability for different objectives and use cases. For example, adjusting the algorithm to only look at the reward or also looking and trying to minimize costs;
8. Reward targets;
   (a) Success rate maximization;

1.4 Novelty

To the author's best knowledge, no comprehensive approach or solution sufficiently addresses the problem. This highlights the need for further research and innovation in the field to address the challenges associated with exploration and exploitation in this setting, where the agents have to converge with optimal speed while maintaining awareness of other arms to react to reward distribution changes quickly. In addition, a set of agents is classified as having limited data and, as such, takes longer to converge and should be helped out by similar agents with more abundant data.

Furthermore, no suitable simulator exists for running simulations that reflect the thesis environment. The author introduces an initial simulator for this setting.

1.5 Thesis Outline

Chapter 2 describes the preliminaries. Chapter 3 discusses the operational aspects besides model simulations and algorithm tests. Chapter 4 covers the methods of how the experiments were done and what was used in the process. Chapter 5 shows the experiment setup and the test cases. Chapter 6 shows the result. Chapter 7 analyses the work done. Chapter 8 presents possible future work. Chapter 9 concludes the work.

The chapters build upon each other. When reading through the document for the first time, it is recommended to read it chapter by chapter.

2 Background

This chapter covers the two primary paradigms used to implement solutions to the stated problem.

2.1 Reinforcement Learning

The real-time ML models discussed in the paper are placed in the RL subgroup whose interactions with the environment are described in Figure 3 [13]. The model chooses an action from the available actions $a \in A(s)$, which results in one state of the available states $s \in S$ with a numerical reward from the possible rewards $r \in R \in \mathbb{R}$, which are restricted to problem-specific finite sets. The dynamics of the environment are given by the probability function $p(s', r|s, a)$.

The goal is to determine an optimal target policy $\pi_*(a|s)$ out of all the policies $\pi_N(a|s)$. In the thesis setting, the RL models can use a subset of data because not all actions give rewards, which can be used to adjust tuning. Multiple models are run based on context, and some models get much less data than others, sometimes taking days to converge instead of minutes or hours optimally. This makes the environment constrained as data flow to some models is limited. The models are near-real-time, with rewards given at intervals by a service separate from the models. Appendix 2 shows a general overview of ML paradigms.

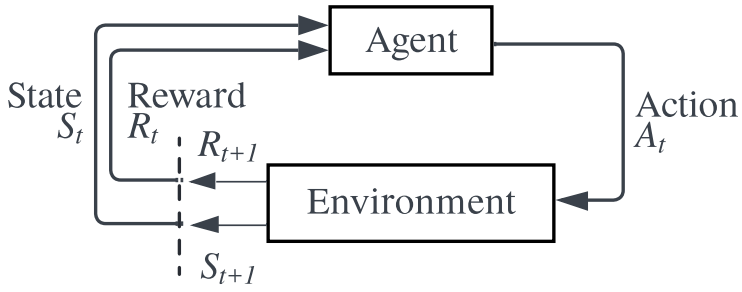

Figure 3. The model-environment interactions in reinforcement learning.

RL is an ML paradigm in which an agent learns to make decisions by performing actions and receiving rewards or penalties. Appendix 4 shows a generalised overview of the methods. It involves learning what to do—how to map situations to actions to maximize a numerical reward signal [14]. The agent is not told which actions to take but must discover which actions yield the greatest reward by trying them. Most RL methods are well

described with GPI (Generalized Policy Iteration), which aims to find the best policy π_* and the value function v_* such that in each iteration the policy is interpreted and evaluated until the best policy is achieved $\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} ... \xrightarrow{I} \pi_* \xrightarrow{E} v_*$ [15, 13].

Agents can be model-based or model-free. A model is anything the agent uses to predict the environment's response to its actions. An example is an estimate of $p(s', r|s, a)$. Model-based methods use the model to plan actions before they are taken. Model-free methods learn action-to-return associations. They evaluate action values q^* instead of state values v^*. Given samples under π, estimate q_π. When doing model-free evaluation, it's possible to express q_π-estimation as v_π-estimation [16].

Agents can be off-policy or on-policy. Behavioural policies b generate experiences or samples from the environment. It defines the actions the agent takes while exploring. Target policies π try to optimize and learn, dictating the agent's actions to achieve the highest expected return.

Off-policy methods involve learning a target policy π that is different from the behaviour policy b used to generate experiences, $\pi \neq b$. This distinction allows off-policy methods to leverage data collected from various sources or past experiences, enhancing sample efficiency. They usually experience larger variance in their estimates and can face stability issues. On-policy methods learn the value of the policy used to make decisions, and the same policy is used to generate the agent's behaviour $\pi = b$. They are also evaluated and improved based on the experiences gathered. A function approximation would look like $\hat{v}(s, w) \rightarrow \hat{q}(s, a, w)$ [17]. RL algorithm groupings can be seen in Appendix 6.

The agent interacts with its environment, which is formalized as a Markov decision process (MDP) and learns a policy to maximize cumulative reward over time. The trade-off between exploration (of uncharted territory) and exploitation (of current knowledge) is central to RL. The algorithms include methods like Q-learning, where agents learn the quality of actions denoting how good an action taken in a particular state will be [18]. RL methods cannot evaluate the loss function for different hypothesis choices compared to supervised and unsupervised ML methods.

Non-stationarity is when the environment's dynamics or the agent's policies change over time, making it difficult for the agent to learn stable and optimal strategies. This can arise from dynamic environments, evolving conditions, or interactions with other learning agents. Non-stationarity introduces challenges such as convergence issues, increased variance in value estimates, and a more complex exploration-exploitation trade-off. To address these challenges, techniques like adaptive learning rates, prioritized experience replay, ensemble

methods, and meta-learning could enable the agent to adapt continuously to the changing conditions and maintain effective learning [19].

2.1.1 Recommendation Systems

Recommender systems are a subclass of information filtering systems that employ algorithms to predict and suggest items a user might like or find relevant [20]. They are used across various industries and platforms.

Netflix employs a sophisticated recommender system to suggest movies and TV shows to its users. By analyzing viewing history, ratings, and user interactions, Netflix creates a personalized profile for each user and recommends content that aligns with their interests [21].

Amazon's recommender system focuses on product recommendations. It employs collaborative filtering, utilizing purchase history, browsing behaviour, and items in a user's wishlist to identify patterns and relationships between products. This allows Amazon to display relevant recommendations like "Frequently bought together" or "Customers who viewed this item also viewed." [22]

Other examples of such systems are the music recommender on Spotify and the video recommender on YouTube. Recommenders can give multiple or no recommendable items $0 \leq n \leq N$ where N is the total item space. The recommendable item count depends on the system design.

In the thesis work, returning multiple preferred providers ranked by their success rate could be feasible and prove beneficial in cases where the best provider fails the delivery. The system can immediately fall back and try the next best provider in line.

This would complicate manners and require careful system design around the recommender to give correct feedback to the model. The external systems would need to store the recommended paths with eventual statuses so the recommender would know which paths were successful. It would also introduce model updates on multiple paths in cases where n of the best providers fail the message delivery in a given period.

Returning multiple providers doesn't provide much benefit, as the assumption is that retries to send the message would once again ask the recommender for a routing decision with the previous status in mind. The models discussed in the work can be considered recommenders that return a single decision.

Contrary to supervised learning, where a model learns from labelled data, recommender systems work well with diverse feedback mechanisms to refine their predictions. This feedback can be explicit, such as user ratings or reviews, or implicit, inferred from user behaviour like clicks, purchases, or dwell time [23].

Traditional recommender systems are predominantly trained offline and involve a cyclical process in which a dataset is initially collected, followed by model training. As time progresses, introducing new items leads to novel user interactions, necessitating periodic retraining of the model [20]. This approach, however, encounters several challenges:

1. Frequency of Retraining: Determining the optimal frequency for retraining the model is tricky. It is common practice to repeatedly retrain the model on the entire historical interaction dataset. However, as the volume of data expands, the computational resources required for retraining escalate, making the process increasingly resource-intensive [24].
2. Integration of New Items: A persistent issue with offline recommender systems is the continuous influx of new items. These items typically lack substantial interaction data, necessitating a period of exploration to accumulate sufficient information. This lack of initial data can hinder the system's ability to effectively incorporate new items into its recommendations until adequate user interaction data is collected.

2.1.2 Multi-Armed Bandit

The thesis mainly focuses on the near real-time reward category within the exploration vs. exploitation problem set using Multi-Armed Bandits seen in Figure 4 [25] also known as k-armed bandits. The estimated value Q of action a at time t is denoted as $Q_t(a)$.

They belong to the semi-supervised ML paradigm, modifying their model through environment interaction rather than training on labelled data. MABs do not directly alter their environment while making decisions in the thesis setting. However, the choices may indirectly influence it, depending on the agent's objective and the environmental properties. The MAB problem can be viewed as a simplified RL scenario, where the agent itself is the bandit, as shown in Figure 5.

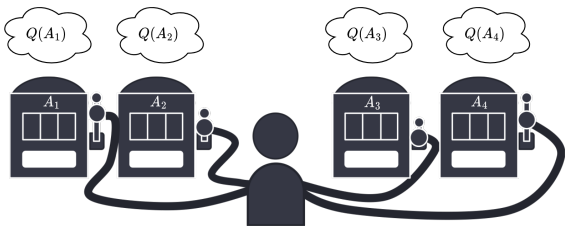

Figure 4. Multi-Armed Bandits.

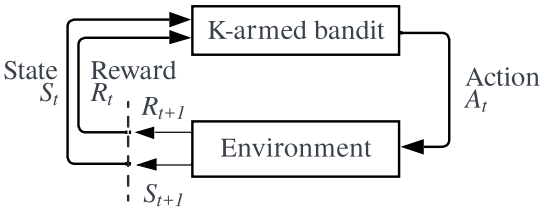

Figure 5. Model environment interaction with MAB.

MAB is a classic framework for sequential decision-making in uncertain environments. A learner repeatedly chooses among a set of arms (options), each associated with an unknown reward distribution. The goal is to maximize the cumulative reward over time, balancing exploring new arms with exploiting known, high-reward arms. The learner cannot see future observations when making current decisions [13]. These algorithms are suitable for problem cases that:

- Involve choosing an action from a set of actions;
- Have a feedback loop that provides a reward for a chosen action;
- Does not require planning for the future. Available choices and rewards in upcoming steps are not affected by the decisions that have been made.

The bandit algorithms have a large-scale utilization in practice. Prominent examples include the following:

- Online Advertising and Recommendation Systems: Choosing which ad or product to display to maximize clicks, purchases, or engagement [26];
- Clinical Trials: Identifying the most effective treatment [27];
- A/B Testing and Website Optimization: Determining the best website design, layout, or call-to-action button to improve conversion rates [28];
- Network Routing and Resource Allocation: Selecting the best path for data packets or allocating resources among users [29];
- Adaptive Game AI: Designing non-player characters that learn and adapt their strategies [30].

The classic Multi-Armed Bandit problems can be viewed as a particular case of an RL problem with a single state. MABs belong to the general class of MDP with a horizon H=1 where optimal policy selection is derived from a single-step reward maximization, $\pi^*(s) = \arg\max_{a \in A} Q(s, a)$.

Extended models such as contextual bandits relax these assumptions and involve multiple states or contexts in their formulation [31]. MABs differ from the complete RL setting in that the actions taken by the agent do not cause the environment to change the set of available actions in the future, making the environment seemingly stateless.

They are widely used in online experimentation, recommendation engines, and traffic routing problems. MABs are often used to explore all options while minimizing suboptimal decisions [2]. It is an actively researched and continuously evolving field, as the solved problem sets are foundational solutions to more complicated real-life problems. They are distilled to a level where relevant features are kept, and the problem is marginally simplified. Appendix 5 describes a subset of MAB paradigms. MABs consist of the following:

1. Arms: Each timestep t actions available a_t are referred to as "arms." The arms have differing reward distribution, which are typically unknown to the decision-maker at the start;
2. Reward Distribution: Arms have unknown reward distributions, which can be for example stochastic or adversarial. Distribution specifics are usually unknown and require the decision maker to estimate them by experimentation [32];
3. Decision Strategy: The approach used to decide the next pullable arm, balancing the trade-off between exploring untested arms to find those with potentially higher rewards (exploration) and exploiting known arms that have given high rewards in the past (exploitation) [33];
4. Regret: Quantifies the difference between the rewards obtained by the chosen strategy and the rewards that would have been obtained by the best possible strategy in hindsight. The goal is often to minimize this regret over time [34]. r_* is the highest possible reward and $R(a)$ is the unknown average reward of a. The expected cumulative regret for n time steps would be $\text{Loss}_n = \sum_{t=1}^{n} \text{loss}(a_t)$.

The difference between many other RL approaches is that the environment dynamics $p(s', r|s, a)$ are unknown. Instead, merely interactions of state, action and reward $S_0, A_0 R_1 S_1 A_1, ..., R_T, S_T$ are received after running some policy through the MDP.

The objective is to choose actions that lead to the highest possible cumulative reward in all n rounds. This task is not an optimization problem mainly because the learner does not know the distribution for each arm. In other words, the bandit instance $v = (P_a : a \in A)$ is unknown to the learner. Another reason a bandit problem is not an optimization problem is that the value of n is unknown. This could be, however, overcome by designing a policy with a fixed horizon and then adapting it for the unknown horizon while proving that the

performance loss of this operation is minimal.

2.1.3 Control Theory

Control Theory offers a structured and mathematical framework for managing and guiding the behaviour of dynamic systems. It provides methodologies for designing and analyzing algorithms that operate effectively in RL systems operating in limited data environments [18].

CT offers a framework for directing the behaviour of systems to achieve desired outputs through feedback loops. It is for systems that perform specific functions under varying conditions. It is increasingly applied in modern applications [35]. CT and MAB problems intersect in the domain of decision-making and resource allocation. CT principles, particularly those concerning feedback and dynamic system behaviour, apply to the strategies employed in MAB problems.

- Control theory can inform exploration vs. exploitation strategies in MAB problems and optimize resource allocation over time for maximum return [36];
- Multi-Armed Bandit problems are akin to a type of optimal control problem. They are particularly suited for situations where resources must be allocated under uncertainty, exemplified by clinical trials [37];
- The concept of regret in MAB problems parallels the conflict in CT between immediate output and future states, mirroring the trade-off between stability and performance [16].

2.1.4 Upper Confidence Bounds

Upper Confidence Bound (UCB) is one of the simulation baselines. It refers to a statistical upper bound on the estimated performance of each strategy. The idea is to construct a confidence interval around estimating a strategy's performance and then consider the upper end of this interval as the optimistic estimate of how well the strategy might perform. Hoeffding's inequality, as seen in Appendix 3, is a fundamental result in probability theory that provides an upper bound on the probability that the sum of random variables deviates from its expected value by a certain amount. It helps estimate the sum of bounded independent random variables. Algorithms similar to UCB are described in Appendix 7.

2.1.5 Contextual Multi-Armed Bandit

Contextual Multi-Armed Bandit (CxMAB) extends the MAB framework by incorporating additional information that may affect the expected reward of each action. They are beneficial in personalized services such as recommendations and online advertising, where the context includes user preferences or situational variables [38]. CxMABs have gained significant traction in recent years in the sphere of recommendation systems. Such models use contextual information to derive the correct data set and decide to obtain the best results for a given optimization function. Context is represented by feature vectors that are categorized as follows:

1. Shared: Common across all actions, time, device, location, etc.
2. Action-specific (parametric actions): A feature vector can specify each action.

The reward in CxMAB problems is a function of the chosen arm and the given context. This complexity adds a layer to the exploration-exploitation dilemma about which arms to explore and how the context changes the arm's effectiveness [39]. The value Q now depends on the context: $Q(s, a)$. As a result, estimating the Q becomes trickier. We can no longer maintain the mean of rewards we've seen. It's possible to learn a Q model using online linear regression: $Q(s, a) = Q_0(s, a) + \phi(s, a)^T p$ [32].

Algorithms for CxMAB problems, like contextual UCB and Thompson Sampling (TS), incorporate context into their learning process to more accurately estimate the value of each arm and manage the exploration-exploitation trade-off. Some CxMAB variants also consider objectives beyond reward maximization, such as minimizing regret or achieving specific goals while respecting resource constraints or fairness [40]. Table 2 describes some differences between A/B Testing, MAB and CxMAB.

Table 2. Differences Between A/B Testing, MAB, and CxMAB.

	A/B Testing	MAB	CxMAB
Dynamic traffic allocation	No	Yes	Yes
Traffic allocation based on feature vectors	No	No	Yes

2.2 Federated Learning

Federated learning enables multiple agents to benefit from collaborative learning without exposing themselves to privacy issues. Such a technique provides the user with benefits

from the learned experience of other users.

Federated Learning (FL) is generally a privacy-preserving technique that trains ML models on devices such as mobile phones without transferring personal data to the cloud. It uses a process in which model updates are shared instead of raw data, maintaining privacy and utilizing the computational power of individual devices. A classic algorithm is FedAvg [41], a fundamental algorithm aggregating local updates to improve a global model. QFedAvg [42] and FedProx [43] are improvements of FedAvg.

FL covers many approaches and research directions. It is an algorithmic setting for distributed machine learning and analytics without centralized data collection and with default privacy. It has federated cross-device learning to train global models on the decentralized data stored across different devices. It also contains local device learning to train custom models for each user on their own devices. It incorporates federated analytics to compute statistics from decentralized data and federated computation for broad-spectrum operations on decentralized data [41].

It is a beneficial technology for personalizing models. Algorithms train a separate model for each local data set. In FL, a whole network exists with nodes being local datasets, each node having its own local feature matrix, label vector, and weight vector. These three can be linked together, forming a network. For the scope of the thesis, we will simulate a network on a single machine.

The main mathematical object is an undirected empirical graph with nodes indexed by natural numbers. Nodes have symmetric (non-negatively) weighted relations. The weight represents the strength of the coupling between nodes. For this to be useful, the edges reflect the statistics of the data set, so it only makes sense to couple the training of two personalized models if they are similar [44].

2.2.1 Privacy Protection

Privacy protection can be addressed through differential privacy, which adds noise to data or gradients to obscure individual contributions and secure multi-party computation, allowing collective model training without exposing personal data. These methods ensure that sensitive information remains confidential while benefiting from aggregated insights during model training [45, 46].

2.2.2 Data Poisoning

Data poisoning in FL involves malicious participants manipulating their local data or model updates to degrade the global model performance [47]. Various strategies exist for poisoning FL, including model update poisoning and targeted data manipulation [48]. Several defence mechanisms have been made to counter these threats, ranging from anomaly detection in model updates to sophisticated schemes such as reputation-based node evaluation and generative adversarial networks to detect poisoned data [49].

2.2.3 Federated Multi-Armed Bandits

Federated MAB (FMAB) systems are a fusion of FL and the MAB framework. This combination allows for decentralized decision-making across multiple agents or nodes, each able to learn and adapt based on their local data while contributing to a global model or strategy. In an FMAB system, multiple agents (each with their local environment) pull the arms of their local bandits and observe rewards. These agents then share their findings in a privacy-preserving manner, typically through aggregated updates or a central server coordinating the learning process. The central server updates the global model or strategy based on the insights received from all agents and then distributes the updated model or decision-making strategy back to the agents as seen in Figure 6 [50, 51].

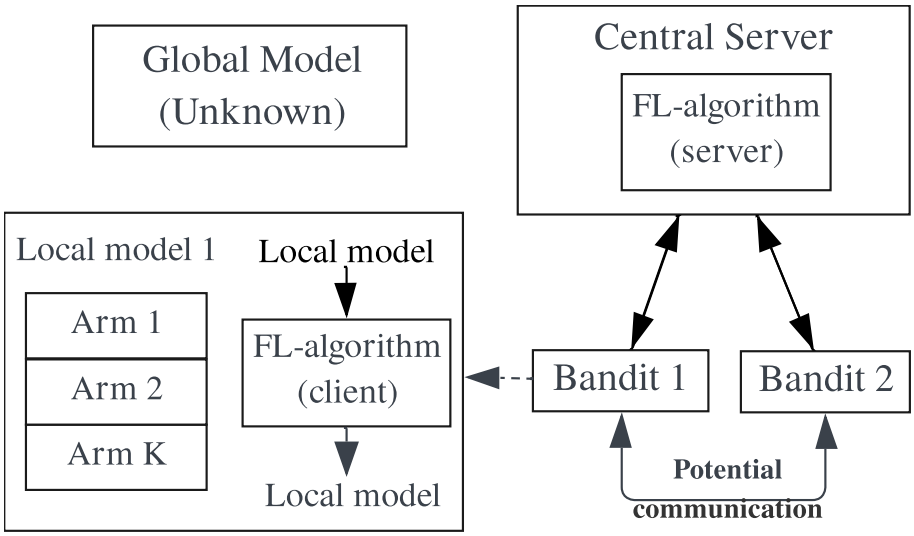

Figure 6. Generalized Federated Learning setup.

FMAB represents an advanced approach in machine learning, balancing the need for personalized, efficient decision-making with privacy and scalability across distributed environments. Table 3 shows a set of algorithms.

Table 3. Federated Multi-Armed Bandit learning algorithms

Algorithm	**Description**	**Decision**
FedUCB [52]	Nodes train models on their local data, prioritizing user privacy. Ensures privacy by incorporating differential privacy techniques into how devices share model updates with a central server.	Custom variant implemented.
FedTS [53]	Ideal for applications where privacy is crucial and the environment may change over time.	TS will not be considered in the FL context
FedLinUCB [50]	A linear contextual bandit algorithm for federated learning environments, which incorporates the benefits of UCB in linear settings with federated data	Custom model building, will be investigated in future work.
Fed2-UCB [54]	Gradually samples new clients while performing arm sampling, and thus simultaneously explores and balances both types of uncertainty.	Client sampling in and out isn't suitable for this setting.
GossipUCB [55]	FNetwork of agents, where each agent can only communicate with its neighbours. Agents make decisions based on local data and share information with their neighbours to achieve global understanding and reduce overall regret in the system.	Considered, eventually disregarded due to slow learning and convergence.
Fed-PE [56]	Federated Phase elimination is proposed to cope with client heterogeneity without exchanging local characteristic vectors or raw data. Fed-PE relies on a novel multi-client G-optimal design and achieves near-optimal regrets for disjoint and shared parameter cases with logarithmic communication costs.	Got implemented; unfortunately, performance was poorer than expected.

2.2.4 Horizontal Federated Learning

Horizontal FL, often referred to as sample-based federated learning, is a paradigm in which multiple parties collaborate to train a shared machine-learning model without exchanging their local data [57, 58]. Each participant has a data set with different samples with the

same feature space. The process of horizontal FL can be outlined as follows:

1. Data Distribution: Each participant has its own local data set. These datasets have the same features but differ in the samples they contain;
2. Local Model Training: Clients independently train models on their local datasets. This step ensures that raw data remains on the client side, preserving privacy;
3. Model Aggregation: After local training, clients send their model updates to a central server. Actual data remains local;
4. Central Aggregation: The central server aggregates the updates to produce a new global model;
5. Global Model Distribution: The updated global model is then distributed back to the clients for further training or inference;
6. Iterative Process: The cycle of local training, model aggregation, and global model distribution repeats until the global model achieves the desired performance.

Horizontal FL is particularly beneficial in scenarios where the same type of data is collected across different entities or devices, such as healthcare, finance, or mobile device applications [59].

2.2.5 Vertical Federated Learning

Vertical federated learning (VFL) is a paradigm in FL where entities collaborate to train a model by contributing different features for the same set of samples. Unlike horizontal FL, which integrates data between different entities based on the same characteristics but different samples, VFL combines data vertically, which means that entities have other attributes or features for the same data samples [58, 60].

The key characteristics of VFL:

- Data Structure: Different entities have different feature sets for the same set of samples. This scenario is common in banking, healthcare, and retail industries, where various organizations collect different data types on the same individuals.
- Privacy Preservation: VFL allows participating entities to collaboratively train a model without sharing their raw data, thus preserving privacy and complying with data protection regulations [61].
- Model Training: In VFL, each participant trains a local model on their feature set and exchanges intermediate computation results rather than raw data, using secure multi-party computation (SMPC) or other privacy-preserving mechanisms to aggregate the

updates.

An example use case of VFL is in the financial sector, where banks and retail companies can combine their customer data (financial history and purchasing behaviour, respectively) to build more accurate credit scoring models without compromising customer privacy [60].

2.3 Prior Work and Literature Review

The problem in the Master's thesis can be categorized as non-stationary stochastic constrained reinforcement learning, which is only concerned with the next decision and balances between exploration and exploitation. To satisfy one of the research questions, the qualities of making such a solution work in a distributed fashion are examined. The field of FL is included in the literature review, which looks at distributed variants of exploration-exploitation. Although FL is a new field, research incorporating MABs with FL exists.

2.3.1 Multi-Armed Bandits with Cost Subsidy

The paper by Sinha et al. [62] explores a variant of the MAB problem where an agent incurs costs to play an arm and aims to optimize rewards and costs. The authors demonstrate that traditional MAB algorithms like UCB and TS do not adapt well to this problem due to their dual-objective nature, leading to suboptimal cost and reward outcomes. They establish a fundamental lower bound for online learning algorithms in this context, highlighting the problem's complexity compared to classical MAB scenarios. The paper introduces a novel Explore-Then-Commit (ETC) algorithm, which achieves near-optimal regret bounds and confirms its effectiveness through numerical simulations. This work broadens the understanding of MAB problems by incorporating cost considerations, offering new insights for applications where the balance of cost and reward is important.

2.3.2 Collaborative Multi-Agent Heterogeneous Multi-Armed Bandits

The paper by Chawla et al. [63] investigates a setting where multiple agents collaboratively learn from different stochastic MABs to minimize group regret. It presents decentralized algorithms for two scenarios: context-unaware, where agents share information randomly, and partially context-aware, where each agent knows some peers learning the same bandit. The study provides theoretical regret bounds, demonstrating the near-optimality of the algorithms, and validates their performance through simulations. This research contributes to understanding how agents in varying contexts can improve decision-making through

collaboration.

2.3.3 Multi-Armed Bandit with Budget Constraint and Variable Costs

The paper by Ding et al. [64] extends the classical MAB framework, incorporating variable costs and a budget constraint. In MAB with Budget Constraints and Variable Costs (MAB-BV), each arm pull yields a random reward and incurs a random cost to maximize the expected reward within the budget limit. This model is particularly relevant for internet applications like online advertising and cloud computing, where actions have variable costs and are budget-constrained. Research in MAB-BV has led to the development of UCB variants that adapt to these variable costs, optimizing the cost-reward ratio within the given budget.

2.3.4 Multi-objective Contextual Multi-Armed Bandit With a Dominant Objective

The paper by Tekin and Turgay [40] introduces a novel approach to the MABs by focusing on scenarios with multiple objectives where one objective is dominant. This approach, known as the Multi-objective CxMAB With a Dominant Objective (CMAB-DO), aims to optimize the dominant objective while considering a secondary one. The study proposes an algorithm and introduces two new performance metrics: 2D regret and Pareto regret, both showing sublinear growth, which indicates the algorithm's effectiveness in long-term learning. The research is pertinent to fields like wireless communication, medical diagnosis, and recommender systems, where decision-making involves balancing multiple objectives.

2.3.5 Federated Multi-Armed Bandits

Federated Multi-Armed Bandits (FMAB), introduced in a paper by Chengshuai Shi and Cong Shen [54], adapts the MAB framework to an FL context. This approach incorporates collaborative learning of a global bandit model among decentralized clients, each with its bandit problem, without direct data sharing. The study delineates two specific models within the FMAB framework: the approximate model, dealing with client sampling uncertainty, and the exact model, which posits the global model as the precise average of local models. The research proposes the Federated Double UCB (Fed2-UCB) algorithm to handle uncertainties from both arm and client sampling, advocating a phased client inclusion strategy to optimize communication costs and achieve an $O(\log T)$ regret.

2.3.6 Federated Linear Contextual Bandits

The paper by Huang et al. [56] presents a novel approach to addressing the challenges in FL through the MAB framework. The proposed model allows individual clients to learn collaboratively without sharing raw data, maintaining privacy and reducing communication costs. The authors introduce the Federated Phased Elimination (Fed-PE) algorithm, which achieves near-optimal regret for both disjoint and shared parameter cases, underpinned by a novel multi-client G-optimal design. Theoretical analysis and experiments validate the effectiveness of Fed-PE, demonstrating its optimal performance on synthetic and real-world datasets compared to existing methods. The study also introduces collinearly-dependent policies, providing a tight minimax regret lower bound for the disjoint parameter case.

2.3.7 Harnessing the Power of Federated Learning in Federated Contextual Bandits

The paper by Shi et al. [50] presents Federated Inverse Gap Weighing (FedIGW), a novel approach for Federated Contextual Bandits (FCB) that integrates FL protocols to enhance decision-making. Using inverse gap weighting (IGW) for contextual bandit tasks, FedIGW updates reward function estimates through FL, improving learning efficiency and adaptability. Theoretical and empirical analyses demonstrate its effectiveness, with the potential for incorporating advanced FL features like personalization and privacy. This work advances FCB research, offering a flexible and robust framework for future developments in FL systems.

2.4 Considered Alternative Approaches

Supervised learning (SL) was considered a viable option to solve the stated problems. Its goal is to learn a mapping from inputs to outputs based on labelled training data. SL aims to make accurate predictions based on historical data. Learning occurs primarily in the training phase. The thesis problem deals with historical data to the extent of maintaining a trail of historical reward-decision mappings while continuously updating the model with new decisions and training the model continuously on live data. It does not base the entire model on historical data to fit a curve. As a metaphor, the curve constantly changes, and the models must readjust to it as near to real-time as possible. This would make maintaining a near-real-time SL model computationally much more costly than the MAB approach [65].

Genetic algorithms are good at exploring a vast search space and finding near-optimal solutions over generations but can be computationally intensive and slow, especially in

real-time scenarios [66].

Game theory and Multi-Agent systems can handle competitive and cooperative interactions systematically but often require comprehensive modelling of other agents, which can be complex and data-intensive. Such systems may not adapt quickly to real-time changes without continuous learning mechanisms [67].

Swarm Intelligence is robust and flexible in dynamic and uncertain environments, but individual agent behaviours can be simple, potentially limiting complex problem-solving capabilities. Global behaviour can be unpredictable and difficult to control precisely [68].

Bayesian Inference requires careful selection of prior distributions, which can influence outcomes significantly [69].

Decision Trees and Random Forests are prone to overfitting in complex environments with noisy data. The lack of continuous learning makes this approach unviable since it is not inherently designed for real-time updates or exploration [70].

Large Language Models (LLMs) are trained using SL on a large corpus of text data and are designed to understand, generate, and manipulate human language. They predict the next word in a sequence, enabling them to create coherent and contextually relevant text based on the input they receive. The SL-based training invalidates this approach based on the prior SL analysis [71].

Control theory (CT) was considered as a viable option but is not well suited for dynamic and non-stationary environments. Since the action space can be regarded as finite, other approaches, such as MAB, offer a more natural decision-making framework. Furthermore, alternatives excel in the exploration-exploitation space and are intuitively more straightforward. In advanced applications, elements of control theory can be integrated. For example, an algorithm can be used for learning and exploration within a control system designed using the principles of control theory. This hybrid approach can take advantage of the strengths of both methodologies to address complex decision-making problems, especially in adaptive and learning-based control systems [72].

3 Operations

This is covered alongside the algorithms to better understand how the methods could be applied in systems and how they are composed to make real-time decisions based on the data and requests flowing in.

3.1 Infrastructure

Figure 7 gives a theoretical high-level example of a system where the RL agents can be placed. Constructing the theoretical architecture falls out of the thesis scope. Notably, the proposed system is similar to recommendation system architectures. Before designing an architecture with similar RL agents, it's necessary to establish how the algorithms operate, what data is needed, and what the data volume is.

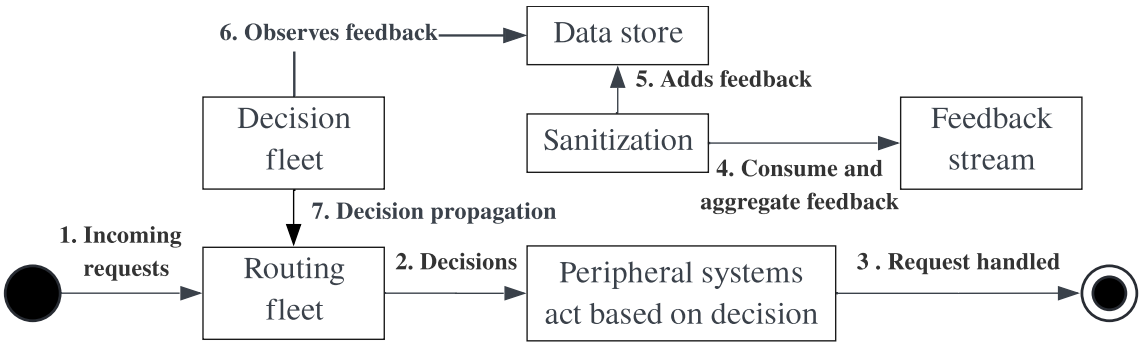

Figure 7. Theoretical system architecture example of a message route recommender.

The researchable algorithms would be placed within the decision fleet. The decision and routing fleets could be merged depending on system requirements.

Production-grade Multi-Armed Bandit (MAB) models are run in Docker [73] and deployed to Amazon Web Services (AWS) Elastic Compute Cloud (EC2) instances. The build system that packages the models is BuildKite [74].

An alternative serverless variant for running the models is described in Appendix 8. Amazon SageMaker Pipelines is an AWS service that creates, automates, and manages end-to-end ML workflows. It covers the entire ML lifecycle, from data preparation and feature engineering to model training, evaluation, and deployment. SageMaker creates a directed acyclic graph that customers can visualize in Amazon SageMaker Studio.

3.2 Cold Start and Data Limitations

The challenge of making optimal decisions is when there is little or no historical data for the arms. This situation is prevalent in newly deployed instances where the lack of data makes estimating the success rates of different actions less accurate. This increases the odds of making suboptimal decisions for a brief period after a deployment, requiring once again the exploration phase. At the same time, the previous model already had information about decision success rates.

This data limitation can also occur when an agent runs into situations where the agent clears its memory per arm or for all arms. This could happen when the reward distribution has changed, and the model has difficulty adjusting. One option to address this could be transferring the instance memory between deployments, as the instance could request historic batches from a data catalogue [75]. Addressing the cold start problem isn't the focus of the thesis.

Data limitations can also arise when the models apply data degradation techniques to the decision-making process. Doing so will devalue past results, effectively reducing the number of messages sent through each path as time passes. This technique addresses the environment's non-stationarity and should be carefully calibrated if applied.

3.3 Data Drift

Data drift in MAB refers to the phenomenon where the probability distributions of the rewards change over time. Restless bandits are relevant for adapting to non-stationary or changing environments. Techniques such as weighted least squares can be employed to adapt to these changes [76]. Time-varying contextual bandit problem, where the reward function evolves, requires dynamic models that can track and adapt to these changes [77].

Data drift in the work setting indicates that an arm has variance in the mean reward distributions per client. In such a case, the relevant MAB agent would ideally identify the drift and adjust its behaviour depending on the volume each arm gets and the severity of the variance.

3.4 Monitoring

As the models run, it's essential to ensure the agents work correctly. The following metrics can be helpful in addition to A/B test metrics:

- Time to compute the decisions is seconds: measures whether the service computational speed is fast enough to calculate the decisions for the next iteration. This would ideally be upper bounded to some threshold, which is not crossed;
- Success rates per destination: measures how well certain regions perform. Ideally, this should be partitioned by destination networks and the vendor delivering messages to that destination;
- Variance in decision making: in order to track algorithm and environment stability;

In addition to the monitoring solutions, the instances themselves should log down relevant events to some log store, which can be looked at afterwards for analysis. If the system starts to perform poorly, it can also be detected through peripheral systems and analytics down the line, in addition to direct monitoring. That knowledge would arrive with latency, making reliance on that variant suboptimal for system health.

3.4.1 Distribution of Arm Pulls

It's beneficial to maintain an overview of agent decision-making within a period. One way to do this is to examine the difference between exploratory and exploitative metrics. For instance, if uniform exploration is done, each option should be pulled with equal frequency. An example of an agent pulling arms over a period is displayed in Figure 8.

3.4.2 Take Rate

The take rate describes the proportion of times an arm is selected by the algorithm in relation to the total number of trials conducted. This metric determines the frequency with which an arm is chosen and can indicate the algorithm's confidence in that arm's potential to provide the highest possible reward. Suppose an arm has a higher take rate. In that case, the bandit algorithm considers it more likely to be optimal based on its past performance and the exploration-exploitation balance strategy employed.

4 Methodology

Methodologies are systems of practices, techniques, procedures, and rules used by those who work in a discipline. This chapter covers the methods, describing how and with what the work was done.

4.1 Technology

Python was the main programming language used to implement the different algorithms during the thesis development. The primary libraries used in the project are the following:

- *Numpy* [78]: for scientific computing, especially beneficial when working with matrices, better performance than Python arrays;
- *Pandas* [79]: for high-performance data structures and tools for data analysis, mainly used for displaying data frames;
- *Matplotlib* [80]: for creating static, animated, and interactive visualizations. Used for plotting algorithm performances in a time series;
- *Streamlit* [81]: for building interactive and deployable data applications. Useful for building an interactive overview dashboard;

4.2 Simulation Framework

The simulator's User Interface (UI) is built using Streamlit, which was chosen since it enables fast prototyping and has good developer experience. The webpage is illustrated in Appendix 9.

Streamlit is an open-source app framework designed for data scientists and engineers to create interactive, web-based data applications. It allows users to turn data into shareable web apps using Python scripts [81].

The UI proved useful primarily for debugging algorithms and quickly comparing different algorithms against each other and with various data. The success rate graphs and algorithm decisions were displayed since the work can simulate different success rates per provider per client. The reward and regret graphs with the final results were also shown.

Streamlit runs on a single thread, so eventually, it became slow, running all the algorithms

for the entire simulation length. To maintain development speed, a configuration-based simulator interface was created using Python. Although not as convenient as Streamlit, it was faster in the end. The next iteration of the UI is considered to be written using Taipy [82], which claims to be more performant and feature-complete.

4.3 Federated Learning

Table 4 shows common libraries for handling Federated Learning in Python. Although these libraries offer helpful utility, the simulation problem space doesn't necessarily require including a standalone library for FL in the simulator. The libraries will be used as inspiration to integrate FL functionalities into the currently built simulation system. This will provide more flexibility in adjusting the functionality as requirements change.

Table 4. Pros and Cons of Federated Learning Libraries.

Library	**Pros**	**Cons**
TensorFlow Federated (TFF) [83]	Integrates with TensorFlow, robust and well-supported	It can be complex, steep learning curve
PySyft [84]	Enhances privacy, supports PyTorch and TensorFlow	Less mature, can be challenging to deploy
FATE [85]	Designed for industrial use, supports secure computing	Primarily used in the financial sector, less documentation is available
Flower [86]	Framework agnostic, flexible	Still evolving, less established community

4.4 Generating Data Sets

To successfully run experiment simulations, the agents making decisions need to be put in a seemingly continuous environment, as in real life. The model decision results are given as feedback to the model at the end of each timestep so it can readjust for the next step, if necessary.

The mean success rate μ and cost of sending the messages per timestep are pre-determined for every option before the simulation run alongside the simulation length. The mean success rate value is bounded as $0 \leq \mu \leq 1$ and represents the percentage of messages that successfully terminate. Although cost isn't discussed further in this section, and the algorithms do not consider it within the scope of the thesis, it exemplifies an additional dimension to consider when constructing algorithms.

The models are run through many simulations with varying data sets. Each client will have its probability distribution when pulling from any given arm. This means that multiple clients sampling the same arm could have varying results. Each client sends a predefined number of messages in a timestep, which varies.

The work makes a distinction between synthetic and real-world data. Real-world data is observed and monitored in real-running production systems. Synthetic data is artificially generated in the simulator based on template cases for every simulation run to test one or many experiment scenarios that have occurred in reality. All graphs associated with simulations use synthetic data.

At the time of writing, executing experiment simulations on past real-life data is impossible, and the only way to truly test on real data is to run the new model live on production traffic. The live system can only trigger one message-sending path at a time without automated systems, and the data for all possible paths in that time step is never learned. Learning based on past real-life data would require constructing a system where the results of all the possibilities are known, and as such, every option is tried for a single message request.

Running such a system for an extended period is not feasible due to significant cost increases. In SMS routing, it is also not conceivable that the receiving device receives the same message $0 - n$ times if sent through n providers when constructing a test data set for simulations.

The construction of such a data-gathering system is out of the thesis scope, although as a theoretical proposal, it can be built separate from the main decision flow. Such a system could be used to construct real-life data sets for simulations and test specific providers before including them in the decision pool for a destination network. Testing providers with a standalone system is not discussed further.

Most general-purpose simulations in this thesis use seasonal traffic patterns to more accurately reflect real-life scenarios. The seasonality looks like a sine function ranging from 50 to 200, as seen in Figure 8. The figure additionally displays the algorithm's routing decisions throughout the simulation and showcases how the best arm decision can change.

In addition, the algorithms were run through, with the volumes kept steady throughout the simulation episode at around 100 requests per timestep per client. Simulating with steady volumes is not strictly necessary since the general-purpose data sets provide sufficient information for the evaluations. Doing so allowed for a more accurate understanding of algorithm characteristics, such as timesteps needed to converge to an optimal arm at the

start or form an outage. The graphs displaying simulation results are still rendered using seasonal traffic volumes, as seen in Figure 8.

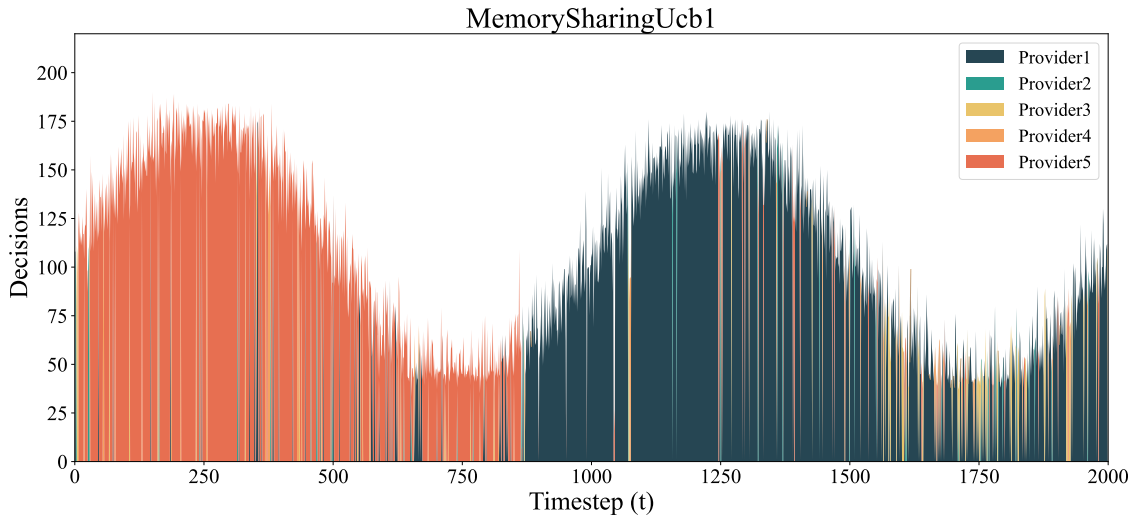

Figure 8. Seasonal data volume with custom algorithm decisions.

Every client has the same predefined set of selectable arms as providers. The arms will remain the same within a simulation episode, but the mean reward in a time step μ_t can vary between clients. For example, the same arm for client 1, $\mu^{a_1}_{t,c_1} = 0.8$ can be degraded for client 2, $\mu^{a_1}_{t,c_2} = 0.15$. The reasons for this can be manifold. The provider might not just handle a certain region well, a cell tower might be broken, or the client is sending bad-quality traffic (e.g., numbers are not in use) that doesn't terminate.

The generatable set for the provider consists of two vectors of length T. One represents the true mean reward μ of a provider at time steps up to T, and the other depicts the cost of using the provider at time steps up to T. The resulting values resemble the ones in Table 5, assuming a reward mean of 0.84 and a mean cost of 0.11.

Table 5. Data represented from the simulated provider perspective.

Step	0	1	2	3	4	5	...	T
Mean success rate μ	0.87	0.89	0.83	0.81	0.88	0.82	...	0.84
Mean cost	0.11	0.11	0.11	0.11	0.11	0.11	...	0.11

Figure 9 shows five providers who can be selected to deliver the message. Their mean success rates are relatively similar, but one performs better than the rest. At one of the timesteps, the best provider's mean success rate drops significantly. This is called a provider outage, which can typically last from a few minutes to multiple hours.

Numerous reasons could have caused the outage. It is even possible that the given platform sends so many messages to the provider that the provider gets congested and stops receiving traffic altogether after some time. Provider overflow detection is not within the thesis scope, and it would be challenging to implement reliably since other entities use the same providers.

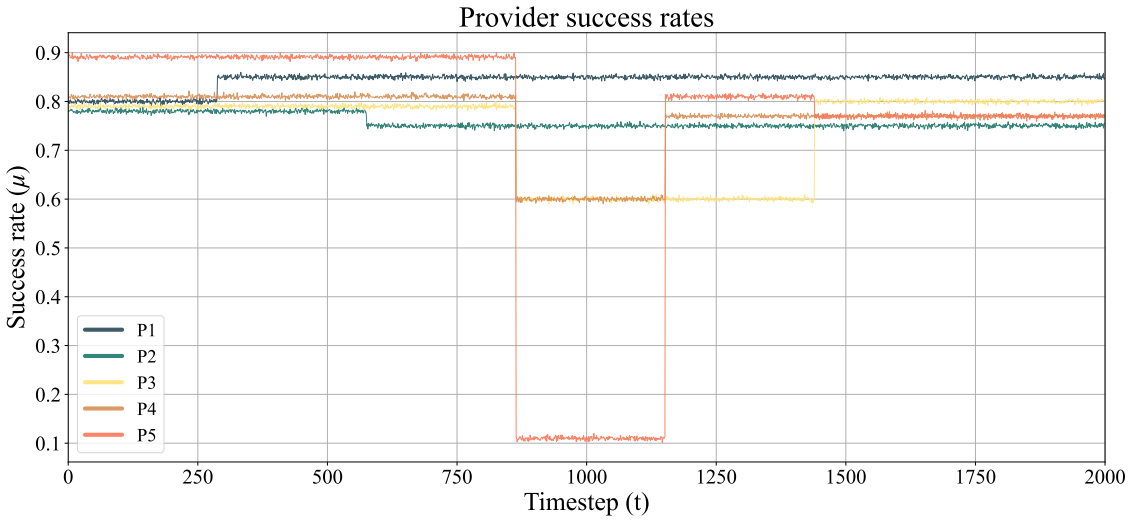

Figure 9. Best provider experiencing an outage.

5 Experiments

The ML models and parameter configurations are compared and evaluated empirically through experiments. The process involves training multiple models on datasets and measuring their performance based on reward and regret.

The reward is a direct measure of algorithm performance, representing the immediate benefit of selecting an arm [13]. In the thesis, the reward is measured as the number of successfully delivered messages out of the total volume sent $\mu_c^m = \sum_{t=0}^{T} \frac{success_{c,t}}{total_{c,t}}$. The experiment results are graphed using cumulative reward.

Regret is a measure of performance loss due to suboptimal action selections. It's defined as the difference between the expected rewards of pulling the best arm and pulling the arm decided by the algorithm. μ^* represents the mean reward of the best arm and $\mu_t(a_t)$ is the mean reward of the arm chosen at time t, the regret after T steps is given by $\rho = T\mu^* - \sum_{t=0}^{T} \mu_t(a_t)$. Minimizing regret is the primary objective in the design of bandit algorithms, as it quantifies the efficiency and effectiveness of the decision-making process [87].

The thesis defines regret as the number of undelivered messages due to suboptimal routing decisions to providers. It is the only regret property applied. It'd be possible to penalise algorithms further for not making the best decision. In practice, the global best decision isn't known, cannot be easily determined and wouldn't add significant value since it doesn't impact algorithm behaviour. Instead, it is an indirect measure reflecting how much worse an algorithm performs than an optimal strategy that always selects the best arm.

Regret is calculated by comparing the running algorithm results against the results of the theoretical best algorithm as $\rho_c = \sum_{t=0}^{T} \pi_{t,c}^* - \pi_{t,c}^m$. Regret cannot be measured only from the running algorithm itself $\rho_c = \sum_{t=0}^{T} \pi_{t,c,total} - \pi_{t,c,success}$ as it would be its own baseline and in the analysis, the algorithm would appear to have no regret which would only be valid if the model is the theoretical best decider.

The aim is to identify models with the highest overall reward across the test cases. This will give higher confidence that the models perform as desired on real traffic. Experiments with the same setup are run n times. The cumulative reward and regret are averaged across simulation episodes to correct for the noise and to have higher confidence in the result.

The simulation procedure is the following for all time steps $t \in T \subseteq \mathbb{Z}$ and $T \neq \varnothing$:

1. An algorithm selects a provider to send messages to for the given step t;
2. The reward is sampled from the provider's reward distribution by the simulation framework and revealed to the algorithm at the end of step t.
   (a) The success rate of the selected provider p at time t is determined by its mean μ_t^p;
   (b) This outcome is revealed after all messages are sent at time t by all clients;
   (c) The algorithm uses the feedback to estimate and recalculate the best provider for the next timestep $t + 1$.

During a simulation run, the agent's timesteps are bounded to T steps $\sum_{t=0}^{T}$. In real-life settings, they are expected to run for as long as a new version is deployed or the process is shut down. This makes the total number of timesteps T unbounded, and it can be considered an indefinite process $\sum_{t=0}^{\infty}$.

Table 6 exemplifies a simplified overview of what information an algorithm operates with when starting to evaluate timestep 5. The data is represented as a multidimensional matrix. Row_1 shows the decision made on that timestep. Row_4 shows the success rate of routing to that provider $Row_4 = \frac{Row_2}{Row_3}$. Since the algorithms do batch inference, the data amounts are stored to determine how much data every timestep has. Batch inferencing means that multiple uniform decisions will be made at every timestep. Row_5 displays cost, which is ignored since the work focuses primarily on quality optimizations. It becomes relevant when the algorithm's objective is to optimize cost.

Table 6. Algorithm decisions on step 5 for client c on time steps T.

0	Timestep	0	1	2	3	4	5	$\sum_{t=6}^{T-1} \pi_{t,c}$	T
1	Arm pulled	1	2	3	1	2	0	...	0
2	Total successful	219	233	242	221	320	0	...	0
3	Total sent	270	274	278	280	400	0	...	0
4	Mean success rate	0.81	0.85	0.87	0.79	0.8	0	...	0
5	Mean cost per message	0.11	0.09	0.11	0.11	0.09	0	...	0

Around 800 - 2000 timesteps are executed per simulation, depending on how many cases a simulation is testing. This proved to be the optimal horizon to observe algorithm convergence with sufficient certainty and is relatively quick, assuming a single timestep lasts a few minutes. The entire 2000 timestep experiment simulates roughly $1.3 - 2.7$ days. The MAB simulations do not require vast amounts of data since they have a simpler

problem structure than full RL, a more direct feedback loop, and no dependency on future states compared to the others.

RL models are generally less data efficient. For example, training AlphaGo and AlphaZero systems to achieve superhuman performance required playing millions of games against themselves, generating vast amounts of data to learn from. For instance, AlphaGo Zero required around 4.9 million games over 40 days, while AlphaZero needed about 21 days of self-play to reach top performance in chess, shogi, and Go [88, 89].

In real cases, these models run for an indeterminate amount of time in a non-stationary environment. Algorithms must maintain flexibility in decision-making when experiencing abrupt changes to the mean reward across the model's life span. Some algorithms, like UCB, have trouble adjusting to changes down the line as more decisions are made and samples are gathered. To adapt to this, a set of models is tried with decaying rates.

Decay rates are applicable for balancing exploration and exploitation, adjusting the learning rate, and implementing temporal discounting. r_t is the reward received at time t, and γ is the discount factor, where $0 \leq \gamma \leq 1$, the decay rate γ indicates how quickly the value of rewards diminishes over time [90].

Contrary to decaying expected future rewards like in temporal difference (TD) [13], historical data observed by the model can be decayed using an exponential moving average. This method applies a decay factor γ^t, where γ is the discount factor and t is the time step. The decay rate γ ensures that recent data points are weighted more heavily while older data points are gradually discounted. This helps models adapt to new information while still learning from past experiences.

In graphs which display simulation results (for example, Figure 13) t_0 indicates the time step in which the figure shows the simulation. This is to narrow down the y-axis values for a better overview since the y-value can start much lower than the eventual optimal. T is the total timesteps. c is the client. n amount of executions for each algorithm. For each execution, the dataset generates new noise, which adds to the variance in the result. The noise is upper-bounded at 2%. n is upper-bounded at 50 since higher values started to degrade the document viewing experience.

There is one thick line per algorithm, which is the calculated mean of the execution. Since the iterations are non-skewed and have equal weight on the average, the mean is also the mode and the median. The algorithm has a confidence interval around which is bounded by the min and max results of the total simulation set.

Hyperparameter tuning optimizes ML models. With MAB, the tuning affects the exploration-exploitation balance and, as a result, the algorithm's overall performance. The following are examples of tunable parameters:

1. Exploration-Exploitation Trade-off Parameters – ϵ;
2. Learning rate – α and δ;
3. Decay rates – γ.

Examples of some tuning results can be seen in Appendix 10.

5.1 Test Cases

Test cases are built so that experiments can be run based on them. The settings mimic real-life edge cases or everyday scenarios. The data sets are two-dimensional and hierarchical. There is a general configuration for the provider and configurations for every client. If the client configuration is missing, then the simulation sampling defaults to what is configured on the provider level.

The start of a test case can be imagined as the start of a new model's successful deployment. The models do not have prior knowledge at the start of each test case. Theoretically, transferring history between deployments is possible, but that isn't discussed in the thesis. The prior state space is too large for it to be included, and adding this info would disrupt algorithm performance and analysis.

Five main cases test different scenarios. These will be referred to in the future sections by their index.

1. Figure 10 displays the normal behaviour of 5 providers. The successful delivery rate is stable, with minimal noise and no outages during the test case period. This tests how much exploration the algorithms tend to do when the reward distributions do not change. There are 2 pairs of providers having similar success rates on different performance levels. In this case, 2 providers deliver with a success rate of 0.82, and the other pair with 0.63. Noting this, the test case also tests how the algorithms choose the best option when there are 2 closely equal decisions, with the best one changing due to added noise.

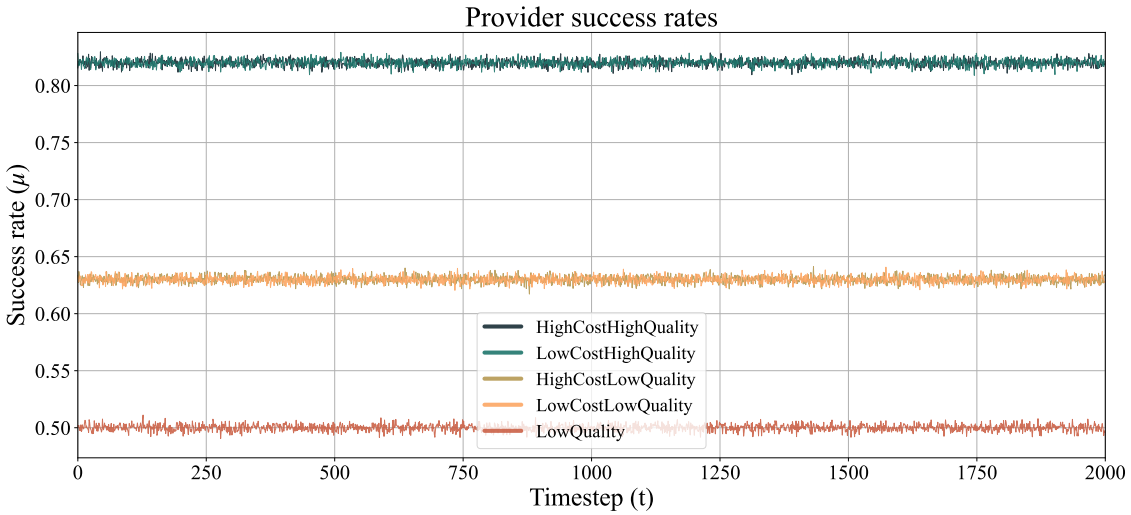

Figure 10. Providers behaving in a normal manner.

2. Figure 9 initially displays the stable behaviour of 5 providers. There are no outages and the suboptimal provider delivery rates drift in time. At a point, the best provider has an outage and after recovering is no longer the best.
3. This is similar to test case 1 as success rates are stable throughout the simulation, and no outages occur. However, there are 3 providers with delivery rates being client dependent, and provider success rates are not close to each other. This case validates whether the contextual algorithms consider the context.
4. This case combines all cases as seen in Figure 11 besides the contextuality test case 3. Everything is stable initially, making the comparison of algorithm convergence more accurate. Then, there is an outage for all options, after which the providers recover one after another. The recovery is a bit different per client, and one of the clients will have another outage afterwards on the best arm.

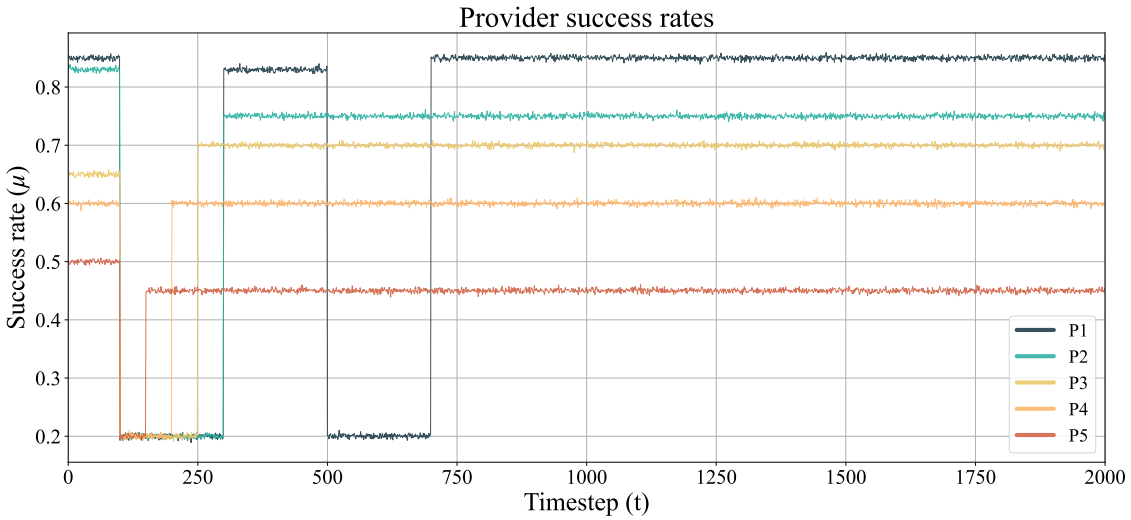

Figure 11. Total provider outage with incremental recovery.

5. The start has stable success rates and no outages or data drift. Then, all providers uniformly have an outage for a short period. There are 3 outage durations tested. Figure 12 shows the smallest outage window. This tests how much a small outage disrupts the algorithms' behaviour.

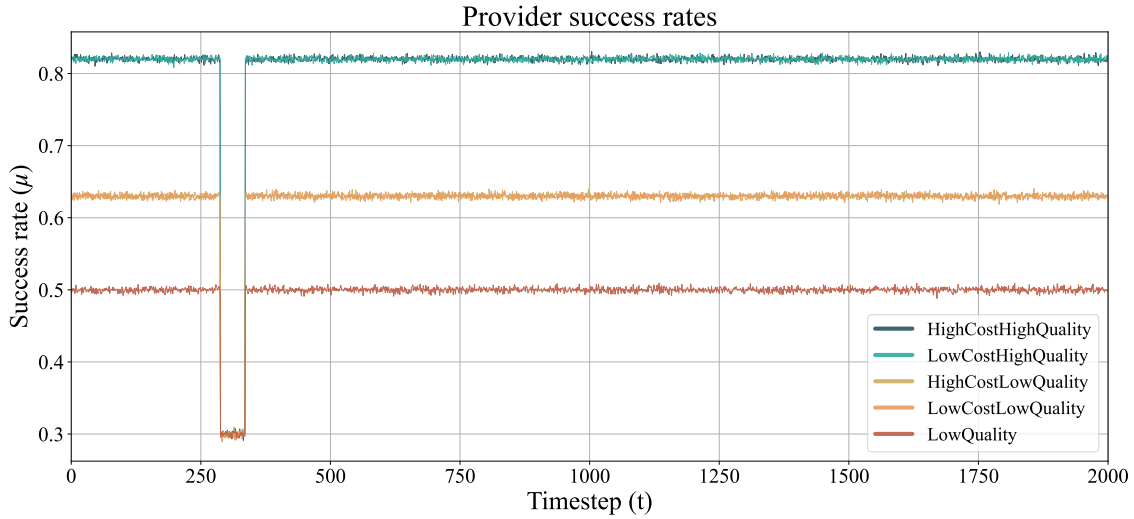

Figure 12. All providers have a small outage.

6 Results

This chapter presents the results of the experiments with various algorithms. Every graph has an accompanying table that provides details about the given simulation.

6.1 Baseline

A baseline refers to a benchmark strategy or algorithm against which the performance of other algorithms gets compared. Baselines are simpler methods that help validate that the proposed approach is better than something naive or simpler in the given environment. The following are common baselines:

- Uniform Sampling: Selects each arm with equal probability, regardless of past outcomes. It provides a way to measure the performance of algorithms against an uninformed strategy.
- ϵ-Greedy Algorithm: Selects the best-known arm with probability $1-\epsilon$ and a random arm with probability ϵ. Introduces some exploration while still exploiting.
- Upper Confidence Bound: Samples arms based on the mean reward and the estimates' variance favouring arms with higher uncertainty to ensure enough exploration.
- Thompson Sampling: Maintains a probability distribution over the potential reward of each arm and samples from these distributions to decide which arm to pull.

All of the above algorithms have been implemented. The goal is to demonstrate that new algorithms perform better than these baselines regarding cumulative reward and regret. The chosen baseline is UCB with $\alpha = 2$.

From the base algorithms, it's best suited for the environment. Other baselines are also tried for every experiment, but they aren't rendered in graphs if their performance isn't highlightable.

The α parameter was determined empirically via hyperparameter tuning, and the value is supported by the literature [91] under certain environment assumptions where it gives theoretical guarantees for the regret bound. However, the paper doesn't claim to achieve the best possible regret bound or that $\alpha = 2$ is optimal for all scenarios. More optimal variants could be found for large time horizons and non-stationary settings. Given the non-stationarity to the thesis environment, the alpha parameter could be dynamically adjusted

throughout time. This would introduce another dimension that needs to be monitored if placed to make real decisions.

Figure 13 is run on test case 5, and simulation results are in Table 7. Uniform random is not added to the graph to give a better view of closely performing algorithms. Executed algorithms behave similarly. The main observational point here is to determine which algorithm recovers the quickest. The TS algorithm outperforms UCB by an insignificant margin, having regret ρ differences under 100, which is a minimal difference in the thesis context.

Table 7. Baseline algorithms with test case 5 details.

Method	Mean %	Max %	Min %	Delivery volume (k)
TS	80.08	80.17	79.99	400.3/500
UCB1	80.03	80.11	79.86	400.3/500
ϵ-greedy	78.62	78.90	78.51	394.5/500
Uniform random	66.57	66.73	66.41	333.5/500

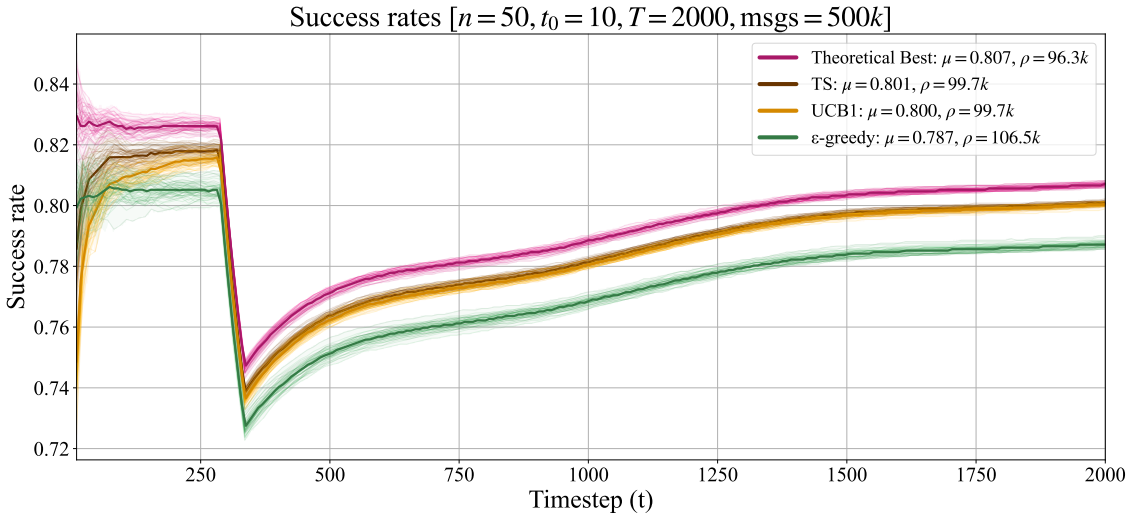

Figure 13. Baseline algorithms with test case 5.

Figure 14 is run on test case 2, and simulation results are in Table 8. The best provider has an outage. This determines how well the algorithms readjust themselves to the new best option. UCB1 outperforms others.

Table 8. Baseline algorithms with test case 2 details.

Method	Mean %	Max %	Min %	Delivery volume (k)
UCB1	84.77	84.93	84.56	424.5/500
TS	83.70	84.68	81.42	423.2/500
ϵ-greedy	83.18	83.59	82.65	417.8/500

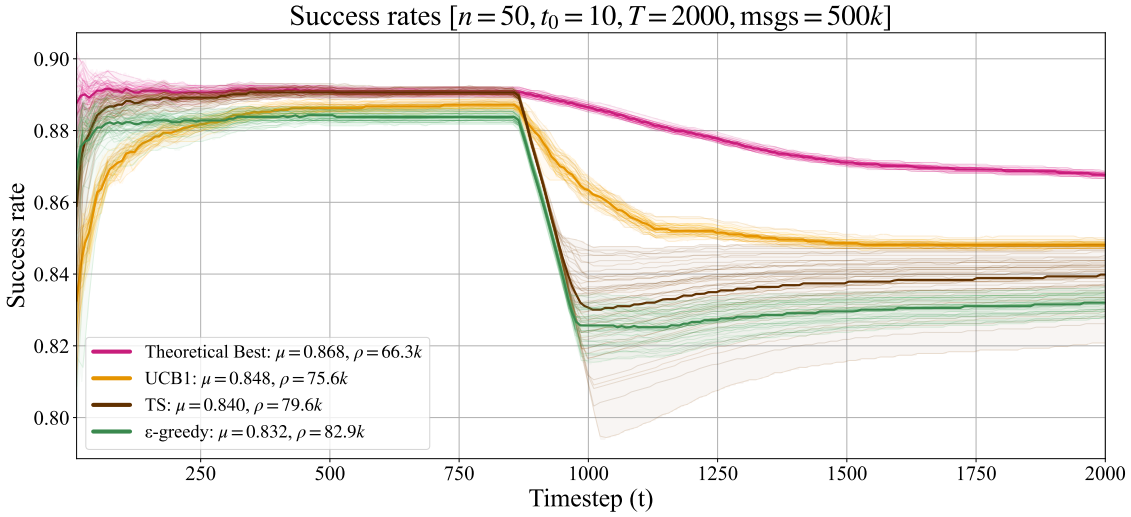

Figure 14. Baseline algorithms with test case 2.

6.2 Algorithms Considered

The MAB setting has a wide variety of algorithms. This section examines some algorithms, determines whether to include them in the simulations and gives reasons for the inclusion or exclusion.

6.2.1 Explore then Commit

Although Explore then Commit (ETC) is a classic category, it will be excluded from further experiments and will not be investigated very profoundly in the thesis. This is because the algorithm's base case halts exploration after the algorithm converges on a single arm. It would be suboptimal in the non-stationary environment's indefinite setting since it is not adaptable to changes in provider success rates down the line. The simulation is run with test case 2 and seen in Figure 15. Experiment details are in Table 9.

Table 9. ETC simulation metrics compared to UCB1 details.

Method	Mean %	Max %	Min %	Delivered volume (k)
UCB1	84.72	84.90	84.44	424.6/500
ETC	82.19	82.63	82.09	411/500

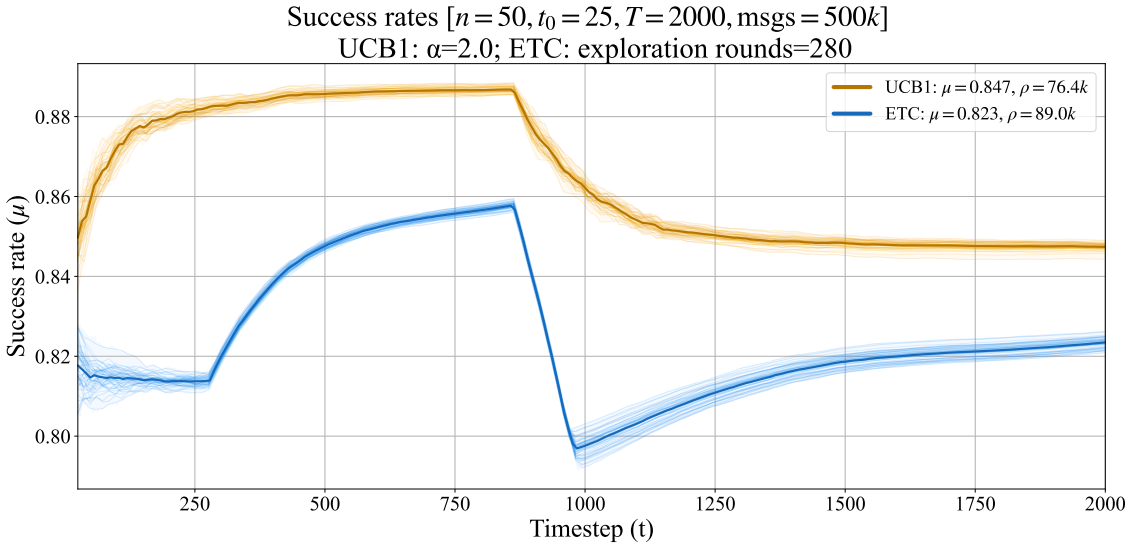

Figure 15. ETC simulation with test case 2.

This is a part of the algorithm design and does not mean the algorithm is terrible. Even in this setting, if ETC were orchestrated carefully, with a change detection system that re-triggers the exploration phase, it could provide more optimal results. As a standalone algorithm, it doesn't suit the environment.

6.2.2 Thompson Sampling

Regarding computational efficiency, while TS's directed exploration can sometimes lead to faster convergence in stationary environments, it doesn't necessarily translate to better performance in non-stationary settings. Figure 16 and Table 10 show that UCB1 outperforms TS in cases where provider success rates change. UCB1's adaptability is preferred over TS's potential conversion speed advantage for the specified environment.

TS may converge faster in stationary environments due to directed exploration based on posterior probabilities. It can be less adaptable to non-stationary environments, where reward distributions change, as seen in Figure 16. This can lead to suboptimal performance when provider success rates fluctuate.

UCB was chosen over TS for this thesis primarily due to its better adaptability in non-stationary environments, ability to recover from provider outages, strong theoretical guarantees, and simplicity for real-world implementation. While TS has potential advantages in stationary settings, UCB's strengths align more closely with the dynamics and practical requirements of the SMS routing problem.

Table 10. TS simulation metrics compared to UCB1 details.

Method	Mean %	Max %	Min %	Delivered volume (k)
UCB1	84.88	85.06	84.71	424.4/500
TS	82.42	84.88	81.94	412.1/500

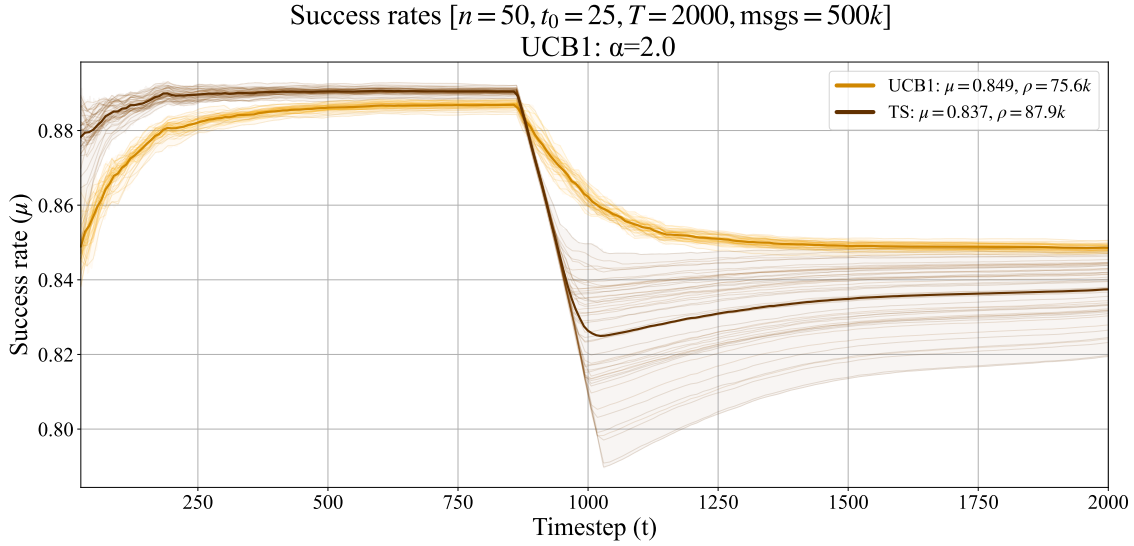

Figure 16. TS simulation on test case 2.

Despite the omission, TS proved to have fast convergence towards the correct arm, which could prove useful in some settings.

6.2.3 Softmax

Although the Softmax algorithm in Equation 6.1 [92]

$$\pi(a) = \frac{e^{\frac{Q(a)}{\tau}}}{\sum_b e^{\frac{Q(b)}{\tau}}} \tag{6.1}$$

can exhibit fast convergence in stationary environments, its performance is less robust when arm rewards are closely clustered and in non-stationary settings with changing provider success rates. This is due to its reliance on accumulated rewards and a tendency to

over-explore already favoured arms. These limitations, observed in theoretical analysis and supported by existing literature [13, 93], show Softmax's shortcomings in the SMS routing environment, where adaptability to change is needed. This method was only looked at theoretically, and no simulations were conducted.

6.3 Upper Confidence Bound Tuned

The tuned UCB algorithm [94] is an enhanced version of the classic UCB1 algorithm. It adjusts the confidence bounds based on the average reward and the variability of each arm, potentially leading to more efficient learning and decision-making.

The main drawback of the standard UCB1 approach is that the UCB1 estimate is derived to be distribution-independent. In our setting, the rewards are from the Bernoulli distribution. Therefore, adopting UCB for a Bernoulli setting can improve confidence and minimize regret.

UCB-Tuned is an approach that considers the estimated variance of rewards and is tuned for Bernoulli MAB problems. UCB-Tuned has been shown to outperform UCB1, but there is no formal proof of having a smaller regret than UCB [95]. Figure 17 runs the algorithm with the test case 2 in Figure 9.

Table 11. Tuned Upper Confidence Bound details.

Method	Mean %	Max %	Min %	Delivery volume (k)
UCB Monitored	86.36	86.51	86.27	432.3/500
UCB1	84.73	85.01	84.59	424.9/500

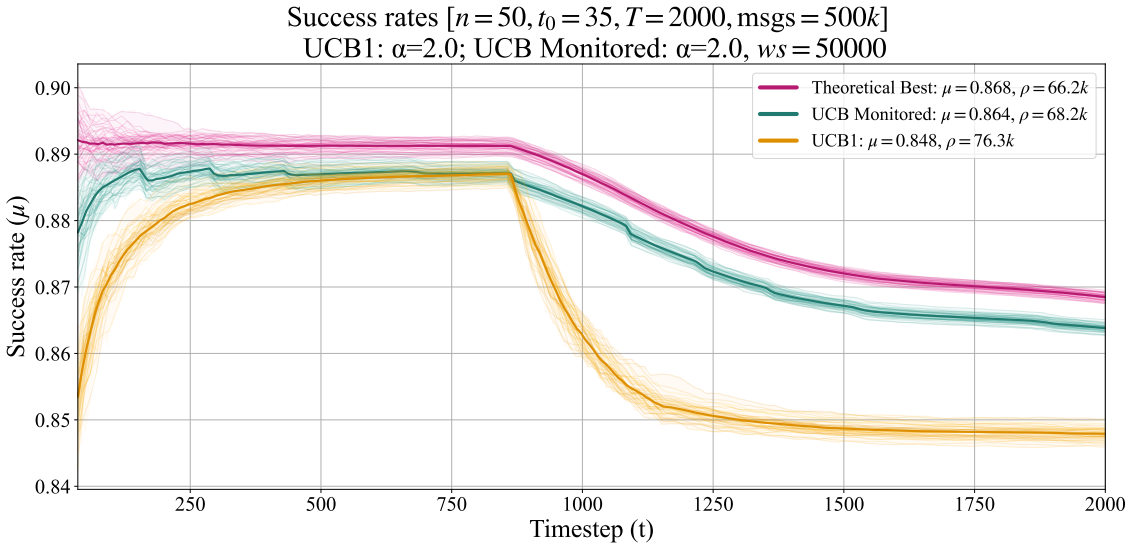

Figure 17. Tuned Upper Confidence Bound.

Adding to UCB tuned, we get a custom algorithm termed "UCB Monitored". The simplified form of this approach is seen in Algorithm 1. It combines multiple methods to improve change detection and handle outages.

Multiple sliding windows were added to monitor the message request flow. Sliding windows offer an overview of the determined time window data. This allows to add custom handling depending on the windows content. For example, there are three windows. Two windows of the same size next to each other to detect outages and a longer window to maintain quality over time. The longer window maintains quality since it has a more stable overview of the options. The values within the smaller windows can deviate a lot more. There are two ways to configure these windows: the amount of time passed or the volume passing by. For this algorithm, the latter was chosen.

Memory resetting is also added. A memory reset causes the algorithm to effectively forget all its observed rewards for all paths or specific paths. In this case, it is done for all arms to re-explore the changed success rates. This is ideally avoided at all costs since it could significantly increase regret. The decision to reset memory comes based on the sliding windows. If the shorter-ranged pair detects an anomaly and it persists in the next window, then the algorithm wipes out all memory it has, and essentially, cold starts again.

Algorithm 1 Monitored Upper Confidence Bound

Initialize $n_t(a) \leftarrow 0$ for all a ▷ Counts the number of times action a is taken
Initialize $Q_t(a) \leftarrow 0$ for all a ▷ Estimated value of action a
Initialize exploration factor: α
Initialize window types and sizes ψ ▷ Largest window is Ψ
Initialize memory reset threshold ω ▷ As a percentage of toleratable difference
Configure memory reset policy ζ ▷ All arms at once or the degraded arm
for each round $t \in T$ **do**
  if $\Delta\psi_{small} > \omega$ **then** ▷ Δ is the success rate difference
    Reset arms according to set policy ζ
  end if
  for each arm $a = 1, 2, \ldots, K$ **do**
    if $a \notin \Psi$ **then** ▷ Arm not explored in largest window
      Increase weights for the arm in selection.
    end if
    $UCB_t(a) \leftarrow Q_t(a) + \sqrt{\frac{\alpha \ln t}{n_t(a)}}$ ▷ Calculate UCB
  end for
  Choose action A_t where $A_t = \arg\max_a UCB_t(a)$ ▷ Adjusted to weights
  Observe reward R_t
  $n_t(A_t) \leftarrow n_t(A_t) + 1$ ▷ Update count of chosen action
  $Q_t(A_t) \leftarrow Q_t(A_t) + \frac{1}{n_t(A_t)}(R_t - Q_t(A_t))$ ▷ Update estimate of action value
end for

6.4 Contextual Bandits

In real cases, the mean reward from the providers can depend on the sender. Contextual MAB introduces additional dimensionality in the context of the sending client. It exposes this knowledge to the algorithm as a context vector linked to the sending client ID.

For example, consider a client with an ID of 592 who could have a vector mapping of $592 \rightarrow [32, 190, 1]$ or $592 \rightarrow [32]$. Different vector values represent specific properties, such as price preferences or a target destination. Contextual algorithms, like LinUCB, connect the received rewards with the context vector to provide a more client-oriented approach than non-contextual variants.

The context vectors can vary in size. It has been observed that longer vectors often give better results, even if the same vector value is duplicated, such as $592 \rightarrow [32, 32]$. This observation suggests that including more features, or even reinforcing existing features, can improve the algorithm's ability to make accurate predictions.

However, the effectiveness of longer context vectors depends on the quality and relevance of the additional features. Adding irrelevant or noisy features can degrade performance. In LinUCB, the algorithm leverages the context vectors to estimate the expected reward for each action by fitting a linear model. The more informative the context vector, the better the model can capture the underlying relationships between context and reward. While longer context vectors can provide better results by offering more information, they must be carefully constructed to ensure that the additional features contribute positively to the model's performance.

To consider this in the simulations, providers can be configured to respond from different reward distributions. Figure 18 shows the simplest context in which only one sender exists. Details are seen in Table 12. This makes LinUCB and UCB act similarly. In particular, LinUCB requires a bigger reward step than just the mean reward float value, i.e. 0.87. To adjust for this, the reward is multiplied by a coefficient δ, which in this case is 10.

Table 12. LinUCB with test case 2 details.

	mean %	max %	min %	Delivered volume (k)
LinUCB	85.37	85.54	85.10	427.5/500
UCB1	84.83	85.05	84.67	425/500
LinTS	84.62	84.74	83.85	423.5/500

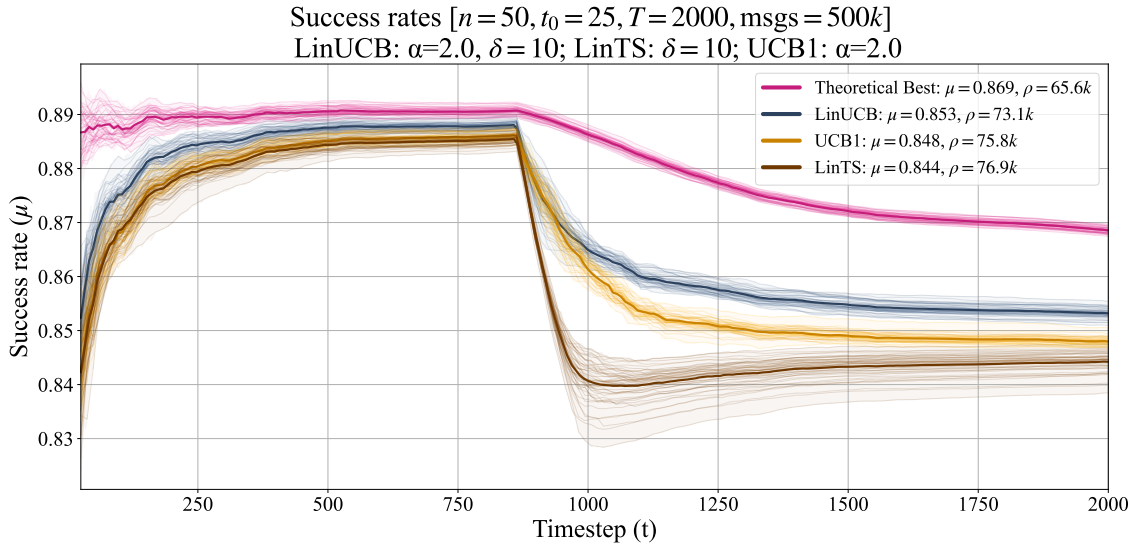

Figure 18. LinUCB with test case 2.

When the mean reward distribution differs by client, but the sendable providers match, as seen in Figure 19, the contextual variants significantly outperform UCB1. Details are seen in Table 13. UCB1 is omitted from the graph to examine contextual performances more closely for the stationary test case 3.

Table 13. Contextual bandits handling differing client-specific feedback distribution details.

	success rate %	max %	min %	Delivered volume (k)
LinUCB	82.02	82.08	81.87	410/500
LinTS	81.94	82.06	81.79	410/500
UCB1	72.50	72.66	72.38	362.5/500

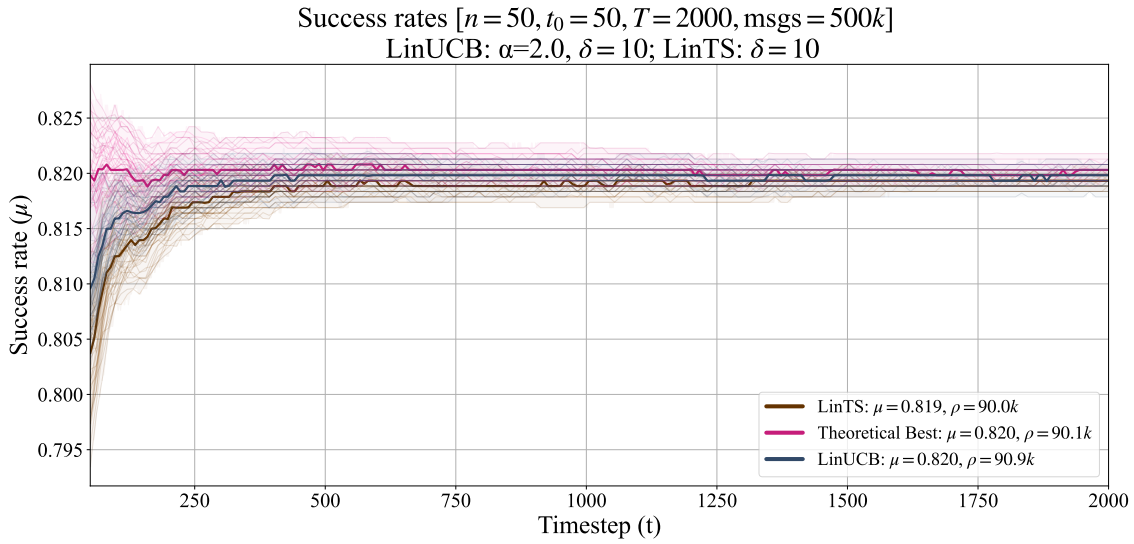

Figure 19. Contextual bandits handling differing client-specific feedback distribution.

Contextual bandits are a promising approach for future use cases. In the thesis scope, the main contributor to the context vector is the sending client ID, but further developments can start to incorporate additional features such as client preferences, regional peculiarities, or different personalized routing strategies.

6.5 Cooperative Algorithms

Cooperative MAB introduces FL elements to the algorithm, like the central server shown in Figure 20, which orchestrates MAB instances under its control and averages client results to give added anonymity. The dotted lines are left for future work to find out ways to make the edge bandits as autonomous as possible while maintaining the benefits of having a federated server.

This is similar to the contextual bandit approach in case the partitioning is done on the client level. Incorporating contextual variants into the mix allows for finer tuning.

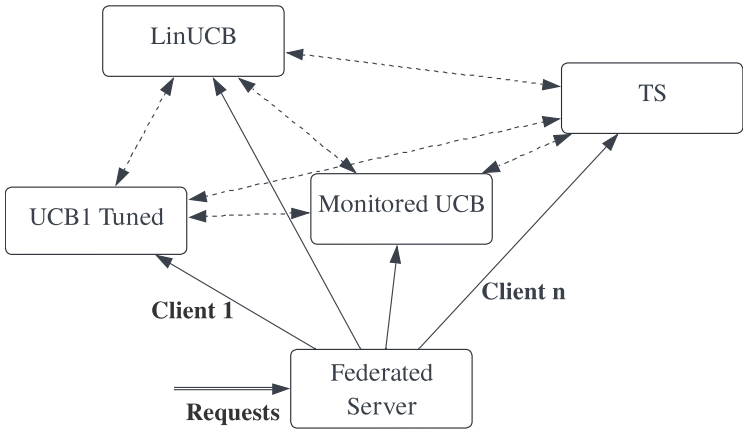

Figure 20. Bandit Mesh.

It is important to consider network communication constraints when deploying agents. It depends on whether the algorithms run on separate machines or a single node. In this work's setting, the networking cost is negligible and can be ignored. Notably, this is different from classical FL, where there is a communication cost with the edge devices.

In real-life scenarios, it will be more optimal from a network communications point of view to host all MAB agents on a single machine and vertically scale the instance until financially viable; after a single instance stops to meet load requirements, the architecture becomes distributed and starts to look more like FL.

6.5.1 Collaborative Bandits

The custom variant combines many different showcased approaches and adds more to it. The simplified version is described in Algorithm 2. The simulation results are in Figure 21. Similarly to the UCB Monitored algorithm, sliding windows are also used here for outage detection.

Every client has a set of configurable sliding windows covering the observed mean rewards μ for timesteps t. One set of windows measures provider success rates, and the other measures volume within the windows. This variant differs from the monitored variant as window types are temporal instead of volume-based. This is, in part, to experiment further with temporal windows, but additionally, temporally emptying windows would indicate that the algorithm arm hasn't been sampled and should be tested again.

This forces the system to maintain an overview of all arms within a period. It would be beneficial in very high-volume scenarios since suboptimal arms will get sampled less over a determined period and limited data settings since all arms are sampled within a determined window. Both options work; it's more of a manner of tuning and seeing what works best.

The algorithm employs three distinct windows to track results, each with a different size and purpose. The small window is most adaptable in length and focuses on the most recent data. The medium window is four times larger and serves as a secondary check on the findings of the small window, capturing a broader perspective to ensure the accuracy and consistency of the analysis. Lastly, the large window is the most expansive at forty times the size of the small window and offers the broadest view, providing context and revealing long-term trends.

The central server ensures that all clients have explored all possible arms within the longest timeframe. The agent looks at other clients with correlating arm results to optimise out unnecessary exploring. From these correlating clients, a heuristic function determines how many will get requested, and the aggregated result is returned to the client result that has started to age out. A skewed variant of the sampled timestep is also added to the requesting client's status matrix.

It seems that this can cause an infinite cascade of borrowing the same data between agents. This is mitigated as the original request sapling timestep remains in a similar range between agents. After the timestep exceeds the sample, one of the clients will sample again.

The algorithm itself keeps track of how much data each arm has available. And asks the federated server to add data into the arm from the same arms of other clients if possible.

The algorithms are assumed to run indefinitely, and the environment has success rate drifts. It is noted that when outages are simulated in the last section, then the algorithms struggle to adjust to the new values. The algorithms use a degrading function on the historical data to adjust for this. This means that historical results impact the algorithm decision less. In the normal status, it is tuned to use $\gamma = 0.999$. Minor changes to the degradation number significantly increase exploration, which would like to be avoided.

An outage is detected when the smallest sliding window observes a degradation of the best arms that receive traffic. Then, the federated agent enters the outage mode for that client, which is computationally much more demanding.

The state of arms before the outage is stored. All arms are resampled sequentially, and this process will continue until all arms have a similar degraded success rate. When an option improves, it is locked in for a short period to force the algorithm to adjust its state.

The pre-outage two best arms are mandated to have a sample within the medium window. The best arm in the medium window is pulled with a heuristic probability. The more data points agree with the value, the higher the likelihood of the arm being pulled.

The degradation factor is changed $\gamma = 0.999 \rightarrow \gamma = 0.9$. The degraded arm is paused for n timesteps. Then, the previous best arm is sampled again. This repeats a few times; if the arm is still experiencing issues, the memory of that arm will be reset. The confidence bound-based algorithms will start using Lower Confidence Bounds (LCB).

Outages are resolved when all arms have returned to normal status, and the success rates are similar to those before the outage.

Table 14. Collaborative Bandits compared to other variants with test case 3 details.

Algorithm	Mean %	Max %	Min %	Delivery volume (k)
Theoretical Best	77.68	77.74	77.51	388.6/500
MemorySharingUcb1	75.34	75.85	73.41	374.1/500
UCB Monitored	72.98	73.67	72.88	368.2/500
LinUCB	70.30	71.25	67.62	356.1/500
UCB1	57.08	72.42	56.61	362.0/500

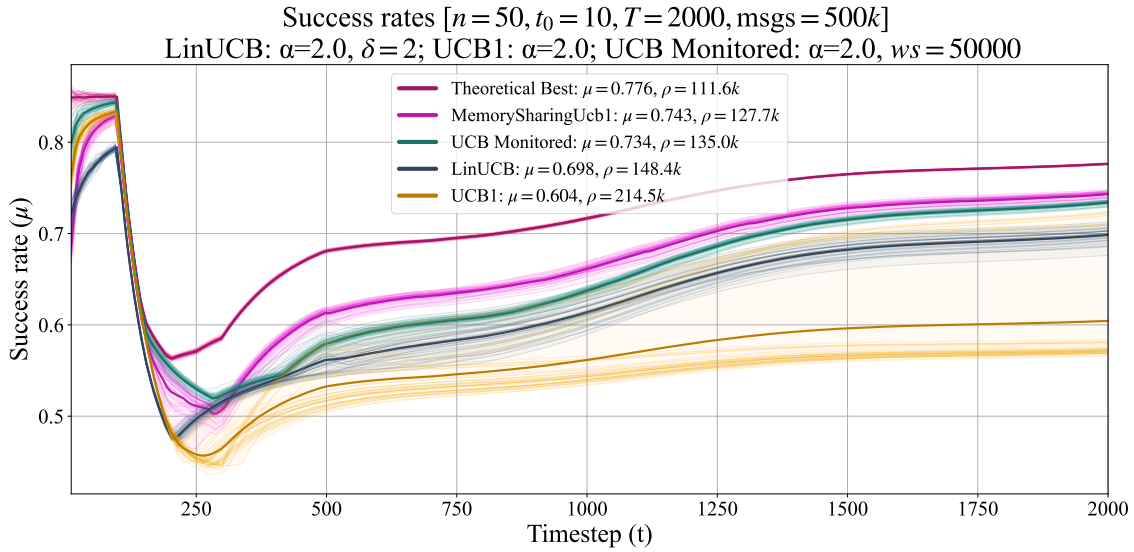

Figure 21. Collaborative Bandits compared to other variants with test case 3.

The collaborative bandits and the UCB monitored are compared against the test case 2 in Figure 22 with details in Table 15. The performance difference is 0.23%, which is insignificant.

Table 15. Collaborative Bandits and UCB monitored with test case 2 details.

Algorithm	Mean %	Max %	Min %	Delivery volume (k)
Theoretical Best	86.81	87.01	86.76	434.1/500
UCB Monitored	86.34	86.51	86.26	431.8/500
MemorySharingUcb1	86.11	86.22	85.97	430.5/500

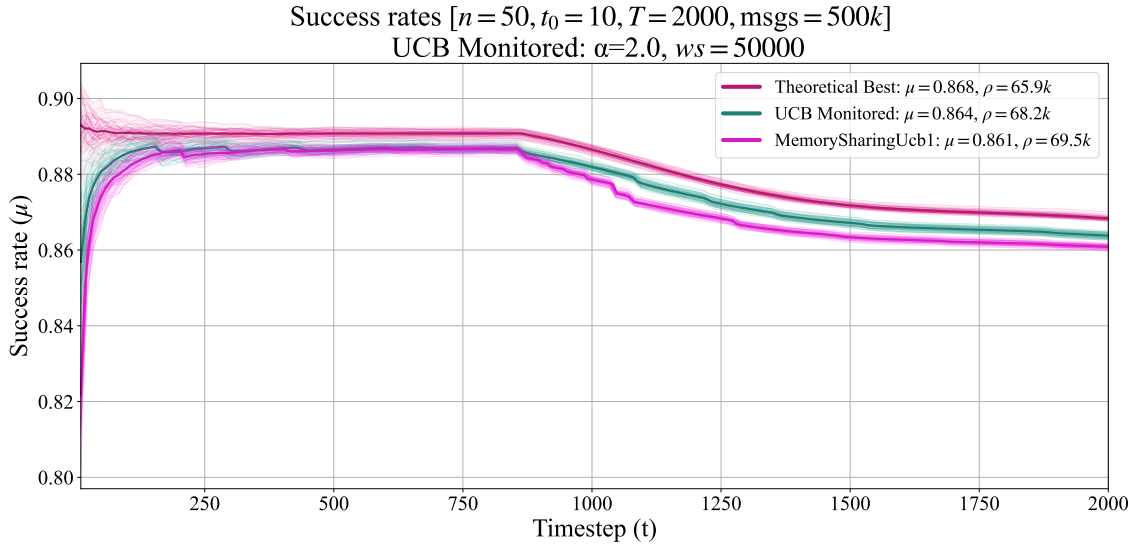

Figure 22. Collaborative Bandits and UCB monitored with test case 2.

Algorithm 2 Collaborative Bandit algorithm

Initialize agent variables: χ_μ, χ_ν ▷ client reward and volume
Initialize global variables: χ_μ^G, χ_ν^G ▷ global reward and volume
Initialize status board: ξ ▷ keeps global and arms state
Initialize window sizes: ψ ▷ Ψ notes the largest window
Initialize sensitivity to outages ω. ▷ noted as a percentage
Initialize starting pause duration ι
for each client $c \in C$ **do** ▷ initialize values in agents
    Initialize degradation factors γ, γ_{outage}.
    Initialize exploration factor α.
end for
for each round $t \in T$ **do**
    for each window $n = 5, 20, \ldots, N$ **do**
        $\psi_{\mu,n} \leftarrow \frac{1}{n} \sum_{t=t-n}^{t} \chi_{\mu,t}$ ▷ Calculate mean reward of all sliding windows
    end for
    $\Psi_\nu \leftarrow \sum_{t=t-n}^{t} \chi_{\nu,t}$ ▷ Volume sum for largest sliding window
    $\xi \leftarrow \chi_\mu, \chi_\nu$ ▷ Update arms states
    Check arms needing exploration in similar agents and share insight if allowed.
    Increase weights for arms needing exploration
    if $\Psi - \psi_{small} > \omega$ **then** ▷ Δ is the success rate difference
        $\xi_{outage} \leftarrow true$
        $\gamma \leftarrow \gamma_{outage}$
        $\xi_{arm_state} = \psi$ ▷ Snapshot pre outage window values
        Re-sample all arms.
        Pause degraded arms for ι timesteps
    end if
    if $\xi_{outage} = true$ **then**
        Unpause degraded arms and mandate their exploration
        if Arms still degraded **then**
            $a_{paused-until} \leftarrow \iota^{times-attempted-to-recover}$ ▷ Pause length is upper bounded
        end if
    end if
    for each client $c \in C$ **do** ▷ Each client has it own agent
        Observe context vector η associated with c
        $\Omega \leftarrow$ Evaluate arms with γ and α applied.
        Choose action A_t from allowed arms, where $A_t = \arg\max_a \Omega$
        if $a \in A$ has recent data below treshold **then**
            Ask the global agent to share memory.
        end if
        Send messages to option A_t
    end for
end for

In general, the collaborative bandit approach is very flexible and can be adjusted in many ways to suit different settings depending on the environment where it is used.

7 Analysis

This chapter evaluates the findings, reviews the experiments, and discusses the methodologies' strengths and limitations. Final algorithm performances, executed through all of the test cases, can be seen in Table 16.

Table 16. Algorithm results over the test cases.

Method	Hyperparams	Mean %	Spread %	Max %	Min %	Converted volume out of 2500 (k)
Theoretical Best	$\varnothing$	82.54	0.25	82.64	82,39	2063
Collaborative UCB	$\alpha = 2$; $\gamma = 0.9999$	80.80	0.51	80.97	80.46	2020
UCB Monitored	$\alpha = 2$	79.43	1.15	79.97	78.82	1986
LinUCB	$\alpha = 2; \delta = 10$	79.23	0.83	79.53	78.70	1980
LinTS	$\delta = 10$	77.22	1.66	77.55	75.89	1930
PF-MAB	$\alpha = 0,5$	76.22	3.97	78.54	74.57	1905
ϵ-greedy	$\epsilon = 0,1$	75.23	3.07	75.63	72.56	1880
UCB with Cost Subsidy	$\alpha = 2$	74.36	4.13	78.02	73.89	1859
UCB1	$\alpha = 2$	74.09	4.16	77.99	73.83	1852
TS	$\varnothing$	73.13	7.61	77.83	70.23	1828
ETC	100 explore rounds	72.22	1.39	72.49	71.10	1805
DLinUCB	$\delta = 0.01$; $\alpha = 2$; $\gamma = 0.9999$; $\sigma = 1$	67.34	1.15	67.92	66.77	1683
Uniform random	$\varnothing$	67.28	0.30	67.43	67.13	1682

7.1 Evaluation Criteria

In addition to the requirements noted in the Goals and Requirements section in Chapter 1, there are specific algorithm criteria. An algorithm or a system of algorithms should consider client-specific success rates. The performance should approach the theoretical best option and converge at most on a 5% worse result than the best possible.

Currently, all algorithms covered in the thesis have the goal of maximizing the reward. Hence, that requirement is fulfilled for all cases and will not be looked at separately.

7.2 Base Methods

Base methods mainly behaved as expected. These variants are logically and computationally the easiest and fastest methods. These algorithms are most eligible if speed is critical and desired over achieving optimal regret. Contextual variants and UCB Monitored also perform quickly. The most computationally demanding algorithm is the Collaborative algorithm, but the computational cost can be tuned according to environment dynamics and resource availability.

Base algorithms optimally converge under the required threshold in a stationary setting; they converge suboptimally when the success rates largely vary. The models might not maintain an overview of the 2 best arms during more stationary periods. Applying a degrading function to the algorithms can improve this.

7.3 Contextual Bandits

Notably, the contextual bandit variants are sensitive to the provided context. The passed context vectors' dimensions and numeric values should be carefully considered when developing the algorithm variant. For example, DLinUCB, LinUCB, and LinTS are all sensitive to the numeric size of the reward received. It must be large enough for matrix calculations, having it within the bounds of the success rate values $0 \leq \mu \leq 1$ made the contextual algorithms perform on the same level with the base algorithms such as UCB and TS even when the reward distributions varied per client. It was concluded from hyperparameter tuning that the optimal reward values for contextual algorithms are $\mu \cdot (8 \leq x \leq 12)$.

Scalar context vectors filled with zeros $[0, \ldots, 0]$ will break contextual algorithm matrix calculations and, as a result, degrade the algorithm's ability to explore and exploit possible paths properly. Furthermore, increasing the feature count inside the LinUCB context boosted the method's performance.

Contextual algorithms converge under the required threshold. The models might not maintain an overview of the 2 best arms during more stationary periods. This can be improved with degraded linear variants, which require more fine-grain tuning compared to what was done in this work. Tuning the degraded models too harshly would result in

aggressive explorations. Too little degrading would ideally give similar results to LinUCB or LinTS, depending on the underlying method.

The algorithms have quite a small variance and handle noise well. The reward level adjustment was introduced as an additional hyperparameter for the contextual algorithms. UCB algorithms also have the configurable α factor, which satisfies the requirement for the algorithm to be adjustable to the environment.

7.4 UCB Monitored

The UCB Monitored is the first custom algorithm implemented. It is based on existing research, combining sliding windows and memory resetting. This algorithm has multi-objectivity built into it, with the goal of eventually considering cost and additional client preferences.

This algorithm has had the chance to be run on production traffic. It has shown noticeable improvement in the regret bounds and has reduced the number of undelivered messages. Unfortunately, this method has not yet had a live run with the memory sharing setting, and the memory sharing needs more testing before being released into a live system.

UCB Monitored converges within the required threshold. Sliding windows allow the algorithm to keep track of all options within a period, sacrificing exploitation rounds and incurring regret. It has many adjustable parameters, the α factor, window sizes and types, and memory resetting threshold, to name a few. It outperforms baselines and handles outages remarkably well. It does more exploration rounds than the collaborative variant, which is expected.

7.5 Client-Centric Multi-Armed Bandits

Considering client specificity in the algorithms poses an interesting case. Assuming N clients, it'd be possible to create N agents, each handling a single client's data. This is likely to give more optimal results per client, assuming the request volume for the client is big enough to allow for accurate decision-making.

Isolated algorithm groups per client for every client are suboptimal since request volumes change in time, and some clients do not send enough requests to calculate their request volume in an isolated group.

Having N client-specific agents is quite similar to the FL paradigm. It likely proves beneficial since a global model representing the entire data volume would exist and can be leveraged by the separate agents as different MAB instances could adjust their models based on that. Since the client cardinality can be quite extensive, to simplify MAB communications, the agents could be categorized based on their respective objective function or geographic region, allowing agent communications within a category.

7.6 Collaborative Memory Sharing UCB

The Collaborative Memory Sharing UCB algorithm, will fall under the termed category of Collaborative Memory Sharing Bandits. Since this class is mostly independent of the underlying collaborative algorithms, it can be tested with various types of algorithms. The only thing the algorithms must have in common is the communication interface, initially with the central collaboration orchestrator.

The collaborating algorithms do not have to be from the same class to communicate. This enables, for example, choosing underlying algorithms that are even more tunable depending on the client's behaviour. This opens the door to more possibilities, like selecting an underlying algorithm based on what is currently happening in the environment or having an ensemble of algorithms that all make decisions and propagate the group result. These options would add to the computational complexity but could prove beneficial in some cases. The environment this algorithm is placed in allows for 0 collaboration cost, unlike classic FL cases.

The collaborative UCB converges within the required period and shows to outperform isolated agents, especially in outage situations. It rigorously keeps an overview of the best two arms within a specified period and re-explores options if they haven't been explored lately to maintain an adequate overview of the possible reward space every action yields. As such, it is set up so that regret is taken to maintain a better overview of the environment during the indefinite setting where the algorithms run.

The algorithm is very adjustable for various environments if need be. In addition to tunable hyperparameters that the underlying algorithms can have, the collaborative orchestration can adjust its sliding window count and size. Smaller windows and time-frames make the algorithm more sensitive to changes but are suboptimal in more stationary settings, suffering more significant regret. The frequency of memory sharing and updating the global values can also be adjusted, the global model is similar to how FedAvg operated in FL apart from the fact that there is currently no noise being added to the received values before aggregating the values. The duration of paused outaged arms can be tuned in cases

where exponential backoff with an upper bound isn't suitable.

The algorithm has quite a small variance and handles noise well. In addition, it can ignore agents whose data doesn't match the current clients' distribution, defending against data poisoning that can occur within the system due to the environment's non-stationarity.

7.7 Simulation Setup

The simulation setup was designed to mirror typical scenarios these algorithms might encounter in operational settings. Parameters like the number of arms, reward variability, and experiment length were adjusted to assess algorithm responses under diverse conditions. The simulator, built in Python and using Streamlit as a UI, offered various ways of testing different cases.

However, it's important to acknowledge that the evaluation relies solely on synthetic data, potentially limiting the generalizability of the findings to real-world SMS routing due to differences between simulated and real-world environments, although this work attempts to minimize the differences by matching the amount of noise observed in real cases and incorporating scenarios witnessed in the real live environment.

7.8 Research Question

This section examines the research questions stated in section 1.2 and gives answers.

1. The research question regarding the creation of novel variants outperforming or being on par with existing solutions is answered. The Collaborative Memory Sharing UCB and UCB Monitored variants outperform other examined variants in the given environment. Collaborative Memory Sharing Bandits answer the question of agents trying to learn from adjacent models;
2. The optimal methods for the given environment are variants which, by nature, are dynamic and offer adjustability to make them more suitable for any given environmental peculiarity. Variations of the sliding window can help with abrupt changes, and degrading the result history could help with slower data drifts. Positive changes can be leveraged by trying out the latest best-performing arm and, as such, behaving greedily.

8 Future Work

This chapter outlines potential future directions for research and development on the thesis topic.

1. Improve change detection in general, but focusing on positive changes;
2. Further improve data drift handling primarily for sudden changes;
3. Incorporate more information into the context vector to improve contextual bandit behaviour;
   (a) Test out contextual bandits with various context vector designs.
4. Look into ways that make the edge bandit algorithms autonomous from the central federated server;
5. Formalize the collaborative memory sharing bandits to be suitable in the federated learning setting where there are communication constraints;
6. Look into other underlying algorithms for collaborative memory-sharing bandits like contextual bandits, TS, ETC;
   (a) Investigating personalized algorithm selection per client per destination and considering the current environment setting;
   (b) Looking into the viability of ensemble methods.
7. Enhancing the scalability of the proposed algorithms to handle larger datasets and more complex environments;
   (a) Taking the cost of sending messages into consideration as one of the objectives;
   (b) Looking further into multi-objective approaches for this setting.
8. Exploring additional applications of the cooperative reinforcement learning framework in other domains, such as e-commerce and healthcare.

Existing research points out possible future works in federated learning. Advances and Open Problems in Federated Learning [96].

9 Summary

This thesis implements and evaluates reinforcement learning methods within a constrained data environment. It focuses on developing cooperative real-time methods and optimizing decision-making processes in telecommunication networks, with examples in short message and multimedia message routing.

A combination of theoretical analysis and practical experimentation explores challenges and opportunities for using reinforcement learning techniques. The research leverages Multi-Armed Bandit algorithms and adapts Federated Learning approaches to improve the cumulative success rate achieved by these methods.

The work examined cooperative learning behaviour among multiple agents to identify strategies for optimizing performance and overcoming data scarcity barriers.

9.1 Objectives

The work aimed to design, simulate, and evaluate reinforcement learning algorithms that can operate effectively in non-stationary environments with limited data.

1. Develop novel reinforcement learning variants under Multi-Armed Bandits that outperform or are on par with existing algorithms;
2. Create a simulator to test the algorithms using synthetic datasets;
3. Demonstrate the benefits of cooperative strategies over isolated approaches regarding service quality improvement.

9.2 Contributions

The executed experiments showed that cooperative strategies outperform isolated approaches, particularly in handling environments with limited data or unstable success rates.

Under the Multi-Armed Bandit class, new reinforcement learning algorithms are introduced. The first algorithm is termed "Collaborative Memory Sharing UCB". It incorporates Federated Learning principles with outage and state management.

The second algorithm is termed "UCB Monitored." It combines different techniques from the literature and adjusts the sliding window technique. Compared to base methods, it handles outages remarkably well. Both methods perform well in the environment and are optimal candidates for real cases.

A simulator is built to help study the reinforcement learning strategies. In addition, extensive experimentation and validation of the models are conducted using synthetic data, and reasoning is given as to why using historical real-life data is not feasible if a dedicated system is not constructed for that purpose.

9.3 Conclusion

This thesis demonstrates that cooperative real-time reinforcement learning methods, supported by federated learning principles, offer a robust solution for optimizing decision-making processes in data-constrained environments. The findings show the potential of these methods to significantly improve message delivery success rates in the real world.

References

[1] Nícollas Silva et al. “Multi-Armed Bandits in Recommendation Systems: A survey of the state-of-the-art and future directions”. In: *Expert Systems with Applications* 197 (2022), p. 116669. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2022.116669. URL: https://www.sciencedirect.com/science/article/pii/S0957417422001543.

[2] Aditya Mahajan and Demosthenis Teneketzis. “Multi-Armed Bandit Problems”. In: Oct. 2007, pp. 121–151. ISBN: 978-0-387-27892-6. DOI: 10.1007/978-0-387-49819-5_6.

[3] Aditya Narayan Ravi, P. Poduval, and Sharayu Moharir. “Unreliable Multi-Armed Bandits: A Novel Approach to Recommendation Systems”. In: *2020 International Conference on COMmunication Systems NETworkS (COMSNETS)* (2019), pp. 650–653. DOI: 10.1109/COMSNETS48256.2020.9027470.

[4] James McInerney et al. “Explore, exploit, and explain: personalizing explainable recommendations with bandits”. In: *Proceedings of the 12th ACM Conference on Recommender Systems* (2018). DOI: 10.1145/3240323.3240354.

[5] Dong Woo Kim, T. Lai, and Huanzhong Xu. “MULTI-ARMED BANDITS WITH COVARIATES:THEORY AND APPLICATIONS”. In: *Statistica Sinica* (2020). DOI: 10.5705/ss.202020.0454.

[6] Anne Laura Penning. “Netflix Recommends: Algorithms, Film Choice, and the History of Taste, by Mattias Frey”. In: *Alphaville: Journal of Film and Screen Media* (2022). DOI: 10.33178/alpha.24.18.

[7] Guoju Gao et al. “Combination of Auction Theory and Multi-Armed Bandits: Model, Algorithm, and Application”. In: *IEEE Transactions on Mobile Computing* 22 (2023), pp. 6343–6357. DOI: 10.1109/TMC.2022.3197459.

[8] D. I. Mattos, J. Bosch, and H. H. Olsson. “Multi-armed bandits in the wild: Pitfalls and strategies in online experiments”. In: *Inf. Softw. Technol.* 113 (2019), pp. 68–81. DOI: 10.1016/J.INFSOF.2019.05.004.

[9] D. Abensur et al. “Productization Challenges of Contextual Multi-Armed Bandits”. In: *ArXiv* abs/1907.04884 (2019).

[10] Robert C. Gray, Jichen Zhu, and Santiago Ontañón. “Multiplayer Modeling via Multi-Armed Bandits”. In: *2021 IEEE Conference on Games (CoG)* (2021), pp. 01–08. DOI: 10.1109/CoG52621.2021.9618892.

[11] Guoju Gao et al. “Auction-Based Combinatorial Multi-Armed Bandit Mechanisms with Strategic Arms”. In: *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications* (2021), pp. 1–10. DOI: 10.1109/INFOCOM42981.2021.9488765.

[12] Chengshuai Shi et al. “Reward Teaching for Federated Multiarmed Bandits”. In: *IEEE Transactions on Signal Processing* 71 (2023), pp. 4407–4422. DOI: 10.1109/TSP.2023.3333658.

[13] R. S. Sutton and A. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018. ISBN: 978-0262039246. URL: https://consensus.app/papers/reinforcement-learning-introduction-sutton-barto-2nd/94d8d982a3b52e13392c301982e291a2/?utm_source=chatgpt.

[14] Richard S Sutton and Andrew G Barto. “Reinforcement Learning: An Introduction”. In: *IEEE Trans. Neural Networks* 9 (1998), p. 1054. DOI: 10.1109/TNN.1998.712192.

[15] M Stanković. “Multi-agent reinforcement learning”. In: *2016 24th Telecommunications Forum (TELFOR)*. IEEE. 2016, pp. 1–1. DOI: 10.1109/NEUREL.2016.7800108.

[16] Steven L. Brunton and J. Nathan Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. 1st ed. Cambridge University Press, 2019. ISBN: ISBN number. DOI: DOInumberifavailable.

[17] Florentin Wörgötter and Bernd Porr. “Reinforcement learning”. In: *Scholarpedia* 3 (2019), p. 1448. DOI: 10.4249/scholarpedia.1448.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518 (2015), pp. 529–533. DOI: 10.1038/nature14236.

[19] L. N. Alegre, A. Bazzan, and Bruno C. da Silva. “Minimum-Delay Adaptation in Non-Stationary Reinforcement Learning via Online High-Confidence Change-Point Detection”. In: (2021), pp. 97–105. DOI: 10.5555/3463952.3463970.

[20] Francesco Ricci, Lior Rokach, and Bracha Shapira. *Recommender Systems Handbook*. Springer, 2015.

[21] Carlos A Gomez-Uribe and Neil Hunt. “The Netflix recommender system: Algorithms, business value, and innovation”. In: *ACM Transactions on Management Information Systems (TMIS)*. Vol. 6. 4. ACM New York, NY, USA, 2016, pp. 1–19.

[22] Brent Smith and Greg Linden. “Two decades of recommender systems at Amazon.com”. In: *IEEE internet computing* 21.3 (2017), pp. 12–18.

[23] James Bennett and Stan Lanning. “The Netflix Prize”. In: *Proceedings of KDD Cup and Workshop*. Vol. 2007. 2007, pp. 3–6.

[24] Yang Zhang et al. “How to Retrain Recommender System?: A Sequential Meta-Learning Method”. In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2020). DOI: 10.1145/3397271.3401167.

[25] James Delorey. *The Multi-Armed Bandit Problem*. Accessed: 11.05.2024. 2023. URL: https://people.stfx.ca/jdelamer/courses/csci-531/topics/multi-armed-bandit/the-problem.html.

[26] Olivier Chapelle and Lihong Li. “An empirical evaluation of thompson sampling”. In: *Advances in neural information processing systems*. Vol. 24. 2011.

[27] Sofia S Villar, Jack Bowden, and James Wason. “Multi-armed bandit models for the optimal design of clinical trials: benefits and challenges”. In: *Statistical science: a review journal of the Institute of Mathematical Statistics* 30.2 (2015), p. 199.

[28] Alekh Agarwal et al. “Taming the monster: A fast and simple algorithm for contextual bandits”. In: *International Conference on Machine Learning*. PMLR. 2014, pp. 1638–1646.

[29] Baruch Awerbuch and Robert D Kleinberg. “Adaptive routing with end-to-end feedback: Distributed learning and geometric approaches”. In: *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*. 2004, pp. 45–53.

[30] Alain Haurie and Georges Zaccour. *Games and dynamic games (Vol. 1)*. World Scientific Publishing Company, 2005.

[31] Guojun Xiong, Jian Li, and Rahul Singh. “Reinforcement Learning Augmented Asymptotically Optimal Index Policy for Finite-Horizon Restless Bandits”. In: (2022), pp. 8726–8734. DOI: 10.1609/aaai.v36i8.20852.

[32] Sébastien Bubeck and Nicolo Cesa-Bianchi. “Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems”. In: (2012).

[33] Keqin Liu and Qing Zhao. “Distributed Learning in Multi-Armed Bandit With Multiple Players”. In: *Signal Processing, IEEE Transactions on* 58 (Dec. 2010), pp. 5667–5681. DOI: 10.1109/TSP.2010.2062509.

[34] Joannès Vermorel and Mehryar Mohri. “Multi-armed Bandit Algorithms and Empirical Evaluation”. In: Oct. 2005. ISBN: 978-3-540-29243-2. DOI: 10.1007/11564096_42.

[35] Ania-Ariadna Baetica, Alexandra M. Westbrook, and H. El-Samad. “Control theoretical concepts for synthetic and systems biology”. In: *Current Opinion in Systems Biology* (2019). DOI: 10.1016/J.COISB.2019.02.010.

[36] A. Mandelbaum. “Discrete multi-armed bandits and multi-parameter processes”. In: *Probability Theory and Related Fields* 71 (1986), pp. 129–147. DOI: 10.1007/BF00366276.

[37] S. Villar, J. Bowden, and J. Wason. “Multi-armed Bandit Models for the Optimal Design of Clinical Trials: Benefits and Challenges”. In: *Statistical science: a review journal of the Institute of Mathematical Statistics* 30.2 (2015), pp. 199–215. DOI: 10.1214/14-STS504.

[38] Chunqiu Zeng et al. “Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit”. In: (2016).

[39] Eric Schulz, E. Konstantinidis, and M. Speekenbrink. “Learning and decisions in contextual multi-armed bandit tasks”. In: *Cognitive Science* (2015).

[40] Cem Tekin and E. Turgay. “Multi-objective Contextual Multi-armed Bandit With a Dominant Objective”. In: *IEEE Transactions on Signal Processing* 66 (2017), pp. 3799–3813. DOI: 10.1109/TSP.2018.2841822.

[41] Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *Artificial Intelligence and Statistics* (2017), pp. 1273–1282.

[42] Tian Li, Maziar Sanjabi, and Virginia Smith. “Fair Resource Allocation in Federated Learning”. In: *ArXiv* abs/1905.10497 (2019).

[43] Anit Kumar Sahu et al. “Federated Optimization in Heterogeneous Networks”. In: *arXiv: Learning* (2018).

[44] Y. Sarcheshmehpour, M. Leinonen, and A. Jung. “Federated Learning from Big Data Over Networks”. In: *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), pp. 3055–3059. DOI: 10.1109/ICASSP39728.2021.9414903.

[45] Kang Wei et al. “Federated Learning With Differential Privacy: Algorithms and Performance Analysis”. In: *IEEE Transactions on Information Forensics and Security* (2019).

[46] Viraaji Mothukuri et al. “A survey on security and privacy of federated learning”. In: *Future Gener. Comput. Syst.* (2021).

[47] Vale Tolpegin et al. “Data Poisoning Attacks Against Federated Learning Systems”. In: (2020).

[48] Lei Shi et al. “Data poisoning attacks on federated learning by using adversarial samples”. In: 2022.

[49] Jiale Zhang et al. “PoisonGAN: Generative Poisoning Attacks Against Federated Learning in Edge Computing Systems”. In: *IEEE Internet of Things Journal* (2021).

[50] Chengshuai Shi et al. *Harnessing the Power of Federated Learning in Federated Contextual Bandits*. 2023. arXiv: 2312.16341 [stat.ML].

[51] Chengshuai Shi and Cong Shen. "Federated Multi-Armed Bandits". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 35.11 (May 2021), pp. 9603–9611. DOI: 10.1609/aaai.v35i11.17156. URL: https://ojs.aaai.org/index.php/AAAI/article/view/17156.

[52] Abhimanyu Dubey and A. Pentland. "Differentially-Private Federated Linear Bandits". In: *ArXiv* abs/2010.11425 (2020).

[53] Zhongxiang Dai, K. H. Low, and Patrick Jaillet. "Federated Bayesian Optimization via Thompson Sampling". In: *ArXiv* abs/2010.10154 (2020).

[54] Chengshuai Shi and Cong Shen. "Federated Multi-Armed Bandits". In: (2021). DOI: 10.1609/aaai.v35i11.17156.

[55] Zhaowei Zhu et al. "Federated Bandit: A Gossiping Approach". In: *ACM SIGMETRICS Performance Evaluation Review* 49 (2020), pp. 3–4. DOI: 10.1145/3543516.3453919.

[56] Ruiquan Huang et al. *Federated Linear Contextual Bandits*. 2021. arXiv: 2110.14177 [stat.ML].

[57] Jakub Konečnỳ et al. "Federated optimization: Distributed machine learning for on-device intelligence". In: *arXiv preprint arXiv:1610.02527* (2016).

[58] Qiang Yang et al. "Federated machine learning: Concept and applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 10.2 (2019), pp. 1–19.

[59] Tao Li et al. "Federated learning: Challenges, methods, and future directions". In: *IEEE Signal Processing Magazine* 37.3 (2020), pp. 50–60.

[60] Stephen Hardy et al. "Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption". In: *IEEE 30th International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE. 2017, pp. 447–454.

[61] Kewei Cheng et al. "SecureBoost: A Lossless Federated Learning Framework". In: *IEEE International Conference on Big Data (Big Data)*. IEEE. 2019, pp. 2597–2606.

[62] Deeksha Sinha et al. "Multi-Armed Bandits with Cost Subsidy". In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Ed. by Arindam Banerjee and Kenji Fukumizu. Vol. 130. Proceedings of Machine Learning Research. PMLR, 13–15 Apr 2021, pp. 3016–3024. URL: https://proceedings.mlr.press/v130/sinha21a.html.

[63] Ronshee Chawla et al. *Collaborative Multi-Agent Heterogeneous Multi-Armed Bandits*. May 2023.

[64] Wenkui Ding et al. “Multi-Armed Bandit with Budget Constraint and Variable Costs”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2013. DOI: 10.1609/aaai.v27i1.8637.

[65] Zhixiang (Eddie) Xu. “Supervised Machine Learning Under Test-Time Resource Constraints: A Trade-off Between Accuracy and Cost”. In: (2014).

[66] Leo Budin, Domagoj Jakobovic, and Marin Golub. “Genetic algorithms in real-time imprecise computing”. In: 1999.

[67] Xiaosong Wu et al. “Research on Game Learning Model for Multi-agent System”. In: 2021.

[68] José García-Nieto, Enrique Alba, and Alejandro Olivera. “Swarm intelligence for traffic light scheduling: Application to real urban areas”. In: (2012).

[69] Vishnu Raj and S. Kalyani. “Taming Non-stationary Bandits: A Bayesian Approach”. In: *ArXiv* abs/1707.09727 (2017).

[70] Lior Rokach and Oded Z. Maimon. “Top-down Induction of Decision Trees Classifiers-A Survey”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 35.4 (2005), pp. 476–487. DOI: 10.1109/TSMCC.2004.843247.

[71] Muhammad Usman Hadi et al. *Large Language Models: A Comprehensive Survey of its Applications, Challenges, Limitations, and Future Prospects*. July 2023. DOI: 10.36227/techrxiv.23589741.

[72] Guanya Shi. “Reliable Learning and Control in Dynamic Environments: Towards Unified Theory and Learned Robotic Agility”. PhD thesis. California Institute of Technology, 2023. DOI: 10.7907/8rz4-7b35. URL: https://resolver.caltech.edu/CaltechTHESIS:08052022-231458463.

[73] *Docker Documentation*. https://docs.docker.com/. Accessed: 2024-04-14.

[74] *Buildkite*. Accessed: 2024-05-02. URL: https://buildkite.com/.

[75] Bastian Oetomo et al. “Cutting to the Chase with Warm-Start Contextual Bandits”. In: *2021 IEEE International Conference on Data Mining (ICDM)*. 2021.

[76] Giuseppe Burtini, Jason L. Loeppky, and Ramon Lawrence. “Improving Online Marketing Experiments with Drifting Multi-armed Bandits”. In: *2015 International Conference on Information and Knowledge Management*. 2015. DOI: 10.5220/0005458706300636.

[77] Chunqiu Zeng et al. “Online Context-Aware Recommendation with Time Varying Multi-Armed Bandit”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. DOI: 10.1145/2939672.2939878.

[78] Travis Oliphant. *Numpy: Library for Large, Multi-Dimensional Arrays and Matrices*. Accessed on April 7, 2024. 2024. URL: https://numpy.org/.

[79] The Pandas Development Team. *Pandas: Library for Data Manipulation and Analysis*. Accessed on April 7, 2024. 2023. URL: https://pypi.org/project/pandas/.

[80] Alex Johnson et al. *Plotly: Creating Interactive and Visually Appealing Data Visualizations and Charts*. Accessed on April 7, 2024. 2024. URL: https://plotly.com/dash/.

[81] *Streamlit Documentation*. https://docs.streamlit.io/. Accessed on April 7, 2024.

[82] Taipy. *Taipy - Build Python Data & AI Web Applications*. Accessed: 2024-05-05. 2024. URL: https://taipy.io/.

[83] *TensorFlow Federated: Open Source Framework for Machine Learning and Other Computations on Decentralized Data*. Accessed on April 7, 2024. URL: https://www.tensorflow.org/federated.

[84] *PySyft: A Python Library for Encrypted, Privacy Preserving Machine Learning*. Accessed on April 7, 2024. URL: https://openmined.github.io/PySyft/index.html.

[85] *FATE: Federated AI Technology Enabler*. Accessed on April 7, 2024. URL: https://fate.fedai.org/.

[86] *Flower: A Friendly Federated Learning Research Framework*. Accessed on April 7, 2024. URL: https://flower.ai/.

[87] Yasin Abbasi-Yadkori, A. György, and N. Lazic. “A New Look at Dynamic Regret for Non-Stationary Stochastic Bandits”. In: *ArXiv* abs/2201.06532 (2022).

[88] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550 (2017), pp. 354–359.

[89] David Silver et al. “A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play”. In: *Science* 362 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404.

[90] Haoran Wang, T. Zariphopoulou, and X. Zhou. “Reinforcement Learning in Continuous Time and Space: A Stochastic Control Approach”. In: *J. Mach. Learn. Res.* 21 (2020), 198:1–198:34.

[91] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. “Finite-time analysis of the multiarmed bandit problem”. In: *Machine learning* 47.2 (2002), pp. 235–256.

[92] Yuxi Liu. *PyTorch 1.x Reinforcement Learning Cookbook*. Accessed: 2024-05-02. O’Reilly Media, 2020.

[93] Aleksandrs Slivkins. “Introduction to Multi-Armed Bandits”. In: *Foundations and Trends® in Machine Learning* 12.1-2 (2019), pp. 1–286. ISSN: 1935-8237. DOI: 10.1561/2200000068. URL: http://dx.doi.org/10.1561/2200000068.

[94] Yang Cao et al. *Nearly Optimal Adaptive Procedure with Change Detection for Piecewise-Stationary Bandit*. 2019. arXiv: 1802.03692 [stat.ML].

[95] Djallel Bouneffouf and Raphaël Féraud. “Multi-armed bandit problem with known trend”. In: *ArXiv* abs/1508.07091 (2015). DOI: 10.1016/j.neucom.2016.02.052.

[96] Peter Kairouz et al. *Advances and Open Problems in Federated Learning*. 2021. arXiv: 1912.04977 [cs.LG].

Appendix – 1 Non-Exclusive License for Reproduction and Publication of a Graduation Thesis[1]

I Oskar Pihlak

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Cooperative real-time reinforcement learning in a limited data environment", supervised by Gert Kanter, Riivo Kikas and Ilja Samoilov.
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

23.05.2024

[1]The non-exclusive license is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in the case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive license, the non-exclusive license shall not be valid for the period.

Appendix 2 – Machine Learning Approaches

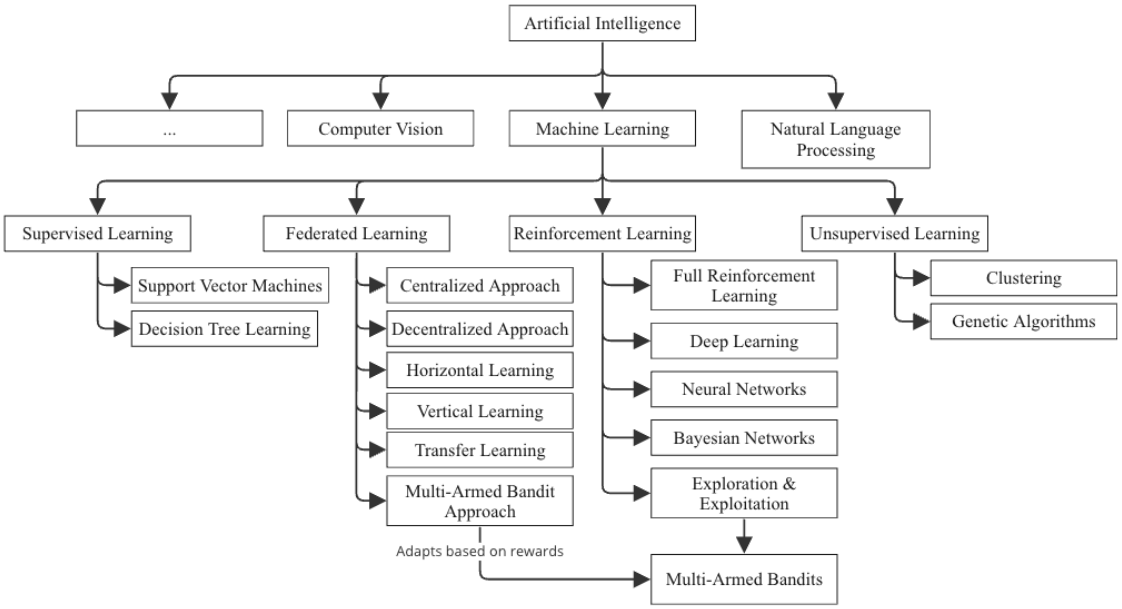

Figure 2.1. Machine Learning Approaches.

Supervised Learning: Algorithms learn from labeled data, pairing inputs with known outputs. It excels at tasks like classification and regression, predicting outcomes for new data. The process involves training the model, and then validating its accuracy on unseen data.

Unsupervised Learning: This approach works with unlabeled data to uncover hidden structures. Techniques such as clustering and dimensionality reduction are common, helping to discover groups or patterns. It's often used for exploratory data analysis and understanding complex datasets.

Reinforcement Learning: Agents in this paradigm learn to make decisions by trial and error, receiving rewards for successful actions. It's widely used in areas like robotics and gaming, where the environment provides feedback to the learner. The key goal is to develop a strategy that maximizes the cumulative reward.

Federated Learning is a machine learning approach that trains algorithms across multiple decentralized devices or servers holding local data samples, without exchanging or centralizing the data. It enables collaborative model training while preserving data privacy and security, as the training data remains on the local devices, and only model updates are shared. This technique is particularly useful in scenarios where data privacy is paramount.

Appendix 3 – Hoeffding Inequality

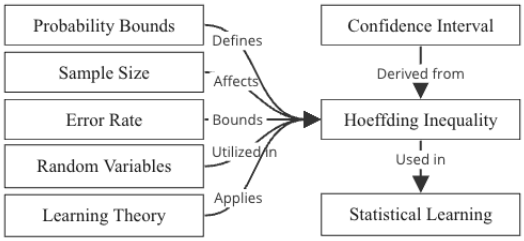

Figure 3.1. Hoeffding Inequality's components.

Hoeffding's Inequality provides a bound on the probability that the sum (or mean) of independent, bounded random variables deviates from its expected value by a certain amount. It is a very useful result in statistics and machine learning, particularly for the analysis of algorithms, as it offers a way to understand the concentration of measure for random variables. Here is a breakdown of what Hoeffding's Inequality states:

- Consider n independent random variables $X_1, X_2, \ldots, X_n$, each bounded by an interval $[a_i, b_i]$.
- The random variables do not need to be identically distributed, just bounded and independent.
- Let $\overline{X}$ be the sample mean of these random variables, and let μ be the expected value of $\overline{X}$.
- Hoeffding's Inequality gives us an upper bound on the probability that $\overline{X}$ deviates from μ by more than a specified amount t, regardless of the individual distributions of the random variables, relying solely on the independence and boundedness assumptions.

$$P\left(\left|\bar{X} - \mu\right| \geq t\right) \leq 2\exp\left(-\frac{2n^2t^2}{\sum_{i=1}^{n}(b_i - a_i)^2}\right)$$

This tells us that the probability of the sample mean being far from the expected mean decreases exponentially as n increases or as the range $b_i - a_i$ of the random variables decreases. In practical terms, Hoeffding's Inequality gives confidence bounds for the true mean of a random variable based on empirical observations. It's widely used in areas like empirical process theory, statistical learning theory, and in the analysis of random algorithms where it provides guarantees for the convergence of empirical means to their true means as more observations are gathered.

Appendix 4 – Reinforcement Learning Methods

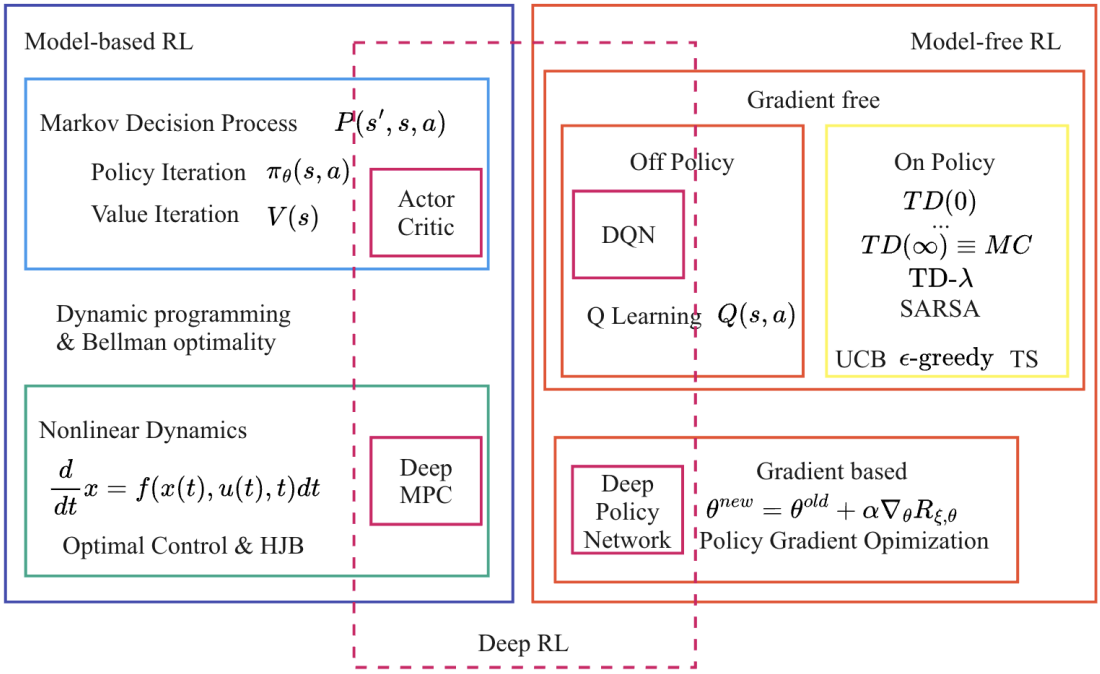

Figure 4.1. Reinforcement Learning Methods.

Appendix 5 – Multi-Armed Bandit and Federated Learning Methods

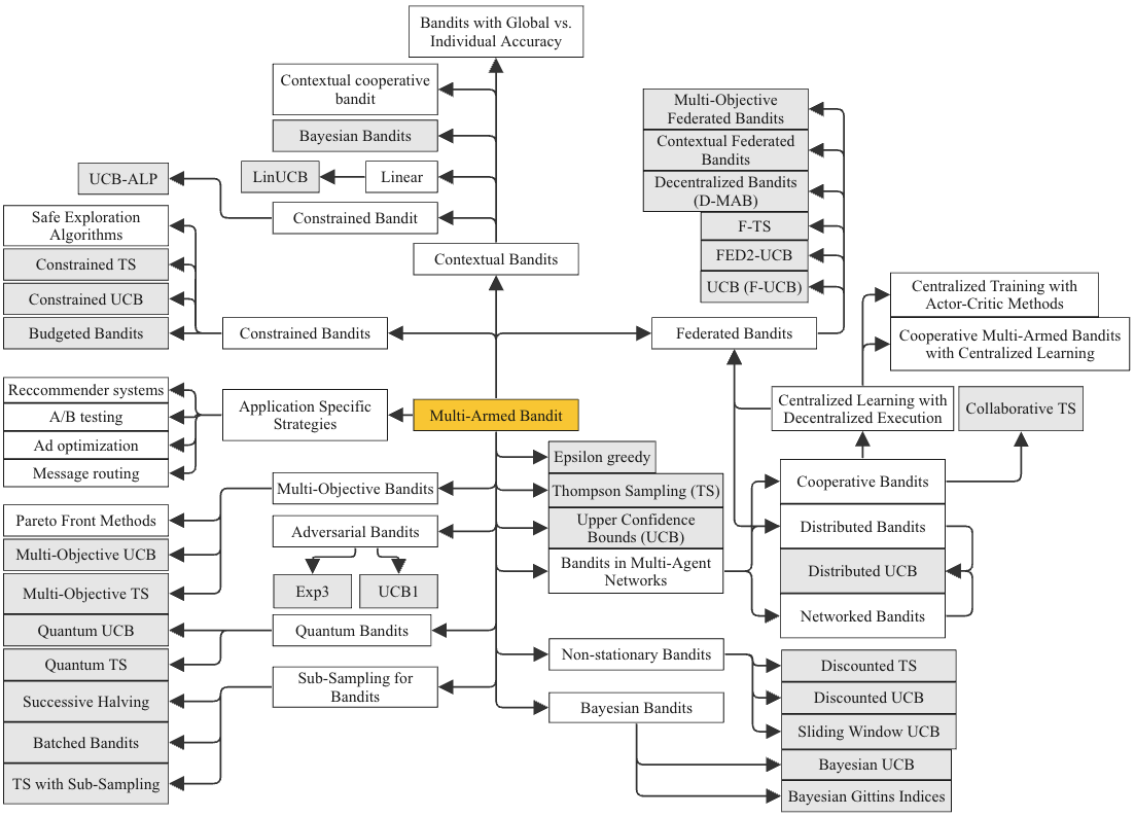

Figure 5.1. Multi-Armed Bandit and Federated Learning Methods.

Appendix 6 – Reinforcement Learning Algorithm Groupings

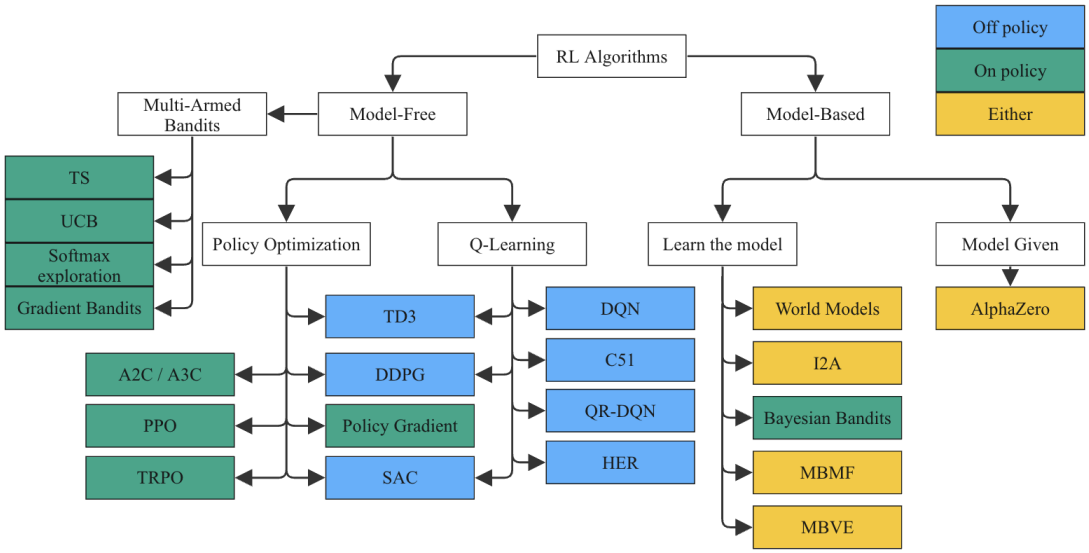

Figure 6.1. Reinforcement Learning algorithm groupings.

Appendix 7 – Algorithms

UCB1 is designed to balance exploration (trying out each arm to gather more information about its reward distribution) and exploitation (choosing the arm that currently seems to offer the best reward). Algorithm: It selects the arm with the highest upper confidence bound, calculated as the sum of the empirical mean of the arm's reward and a term that grows with the logarithm of the total number of plays divided by the number of times the arm has been played. This term ensures that arms not yet explored enough have a higher chance of being selected.

Algorithm 7.1. Upper Confidence Bound (UCB) Algorithm.

Initialize $n_t(a) \leftarrow 0$ for all a ▷ Counts the number of times action a is taken
Initialize $Q_t(a) \leftarrow 0$ for all a ▷ Estimated value of action a
for $t = 1, 2, 3, \ldots$ **do** ▷ Main loop
    for each action a **do**
        if $n_t(a) = 0$ **then**
            $UCB_t(a) \leftarrow \infty$ ▷ Ensure all actions are tried at least once
        else
            $UCB_t(a) \leftarrow Q_t(a) + \sqrt{\frac{2 \ln t}{n_t(a)}}$ ▷ Calculate UCB
        end if
    end for
    Choose action A_t where $A_t = \arg\max_a UCB_t(a)$
    Observe reward R_t
    $n_t(A_t) \leftarrow n_t(A_t) + 1$ ▷ Update count of chosen action
    $Q_t(A_t) \leftarrow Q_t(A_t) + \frac{1}{n_t(A_t)}(R_t - Q_t(A_t))$ ▷ Update estimate of action value
end for

UCB2 refines the exploration-exploitation balance by varying the amount of exploration over time, based on a parameter that controls the degree of exploration. Algorithm: UCB2 introduces a more complex formula for calculating the confidence bounds, including an exploration parameter that can be adjusted. It uses a mechanism of "epochs" where each arm, once selected, is played a certain number of times, with this number increasing as the algorithm progresses.

UCB Tuned adapts its exploration strategy based on the observed variance in rewards, potentially leading to more efficient exploration than UCB1, which uses a fixed exploration term. UCB Tuned is more complex than UCB1 because it requires additional computations to estimate the variance of rewards for each arm. In practice, UCB Tuned can outperform UCB1, particularly in environments where the reward distributions have signif-

icant variance, as it more effectively allocates exploration efforts. UCB Tuned needs to track the reward variance for each arm, which can increase the storage and computational requirements compared to UCB1. While UCB Tuned can offer advantages in specific scenarios, its performance benefit needs to be weighed against the increased complexity and computational cost, especially in resource-constrained environments.

LinUCB

Algorithm 7.2. Linear Upper Confidence Bound (LinUCB).

Initialize parameters: α (confidence level)
for each arm $a = 1, 2, \ldots, K$ **do**
    Initialize $A_a \leftarrow I_d$ ▷ I_d is the d-dimensional identity matrix
    Initialize $b_a \leftarrow \mathbf{0}_d$ ▷ $\mathbf{0}_d$ is a d-dimensional zero vector
end for
for each round $t = 1, 2, 3, \ldots$ **do**
    Receive context $x_{t,a}$ for each arm a
    for each arm $a = 1, 2, \ldots, K$ **do**
        $\theta_a \leftarrow A_a^{-1} b_a$ ▷ Compute parameter vector
        $p_{t,a} \leftarrow \theta_a^\top x_{t,a} + \alpha \sqrt{x_{t,a}^\top A_a^{-1} x_{t,a}}$ ▷ Compute UCB
    end for
    Choose arm $a_t = \arg\max_a p_{t,a}$
    Observe reward r_t
    $A_{a_t} \leftarrow A_{a_t} + x_{t,a_t} x_{t,a_t}^\top$
    $b_{a_t} \leftarrow b_{a_t} + r_t x_{t,a_t}$
end for

Appendix 8 – Multi-Armed Bandit System Architecture in the Cloud

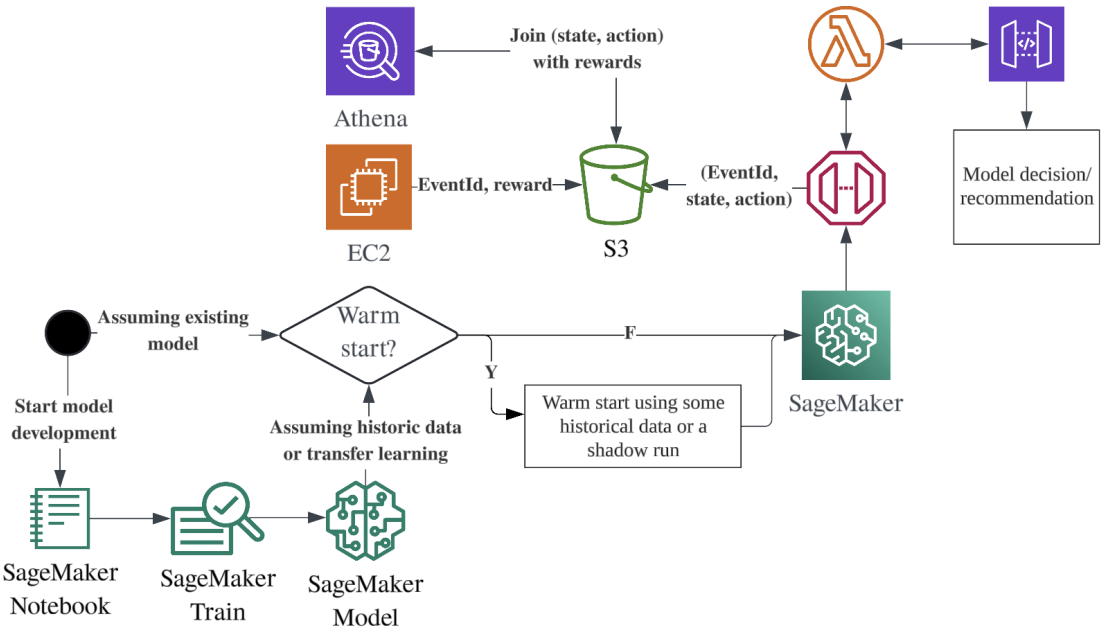

Figure 8.1. AWS architecture for RL systems.

To implement the explore-exploit strategy in Amazon SageMaker RL, there is an iterative training and deployment system that does the following:

1. Presents the recommendations from the currently hosted contextual bandit model to the user, based on her features (context);
2. Captures the implicit feedback over time;
3. Continuously re-trains the model with incremental interaction data.

Appendix 9 – Simulator UI

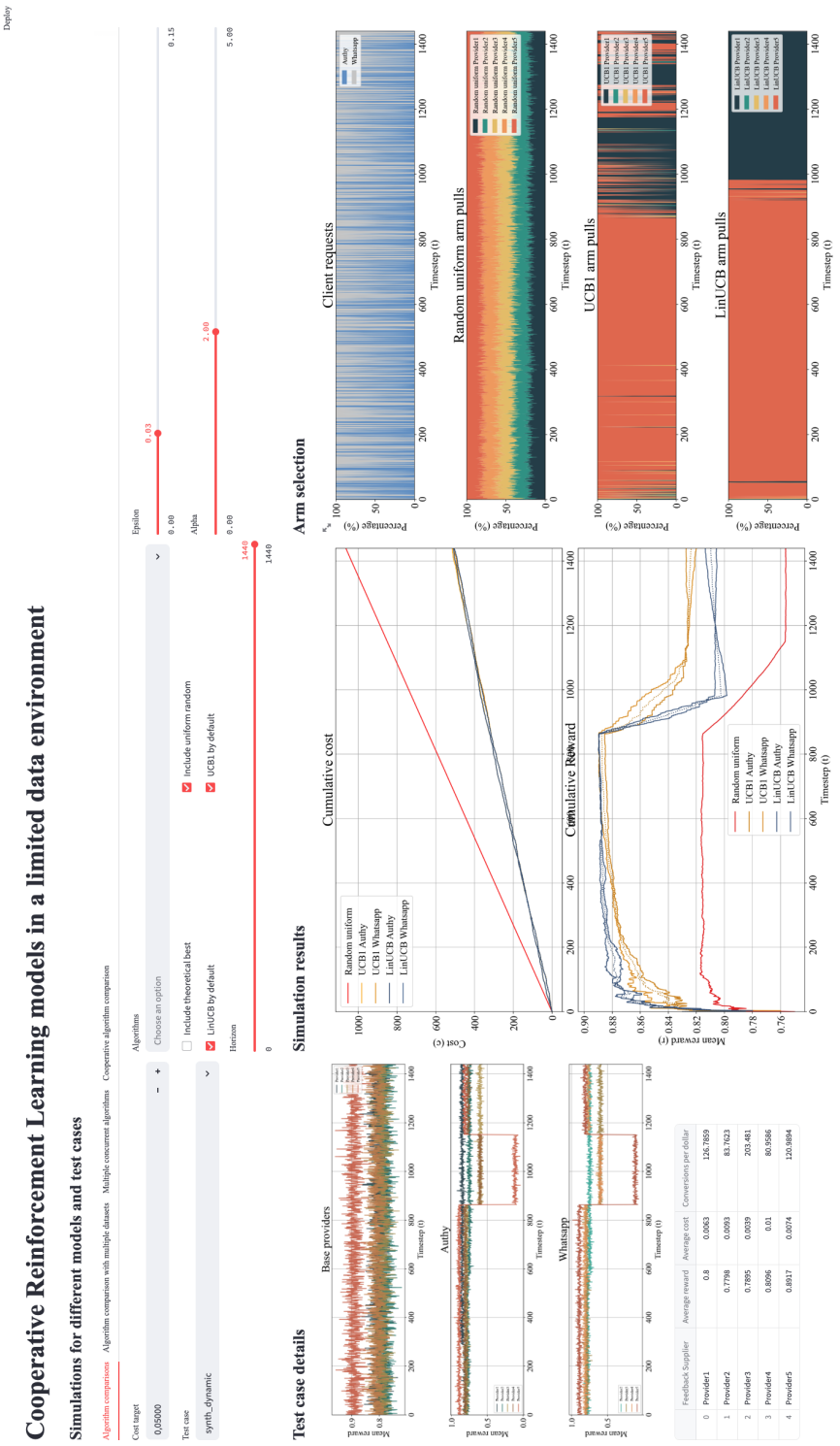

Figure 9.1. Simulator UI.

Appendix 10 – Hyperparameter Tuning

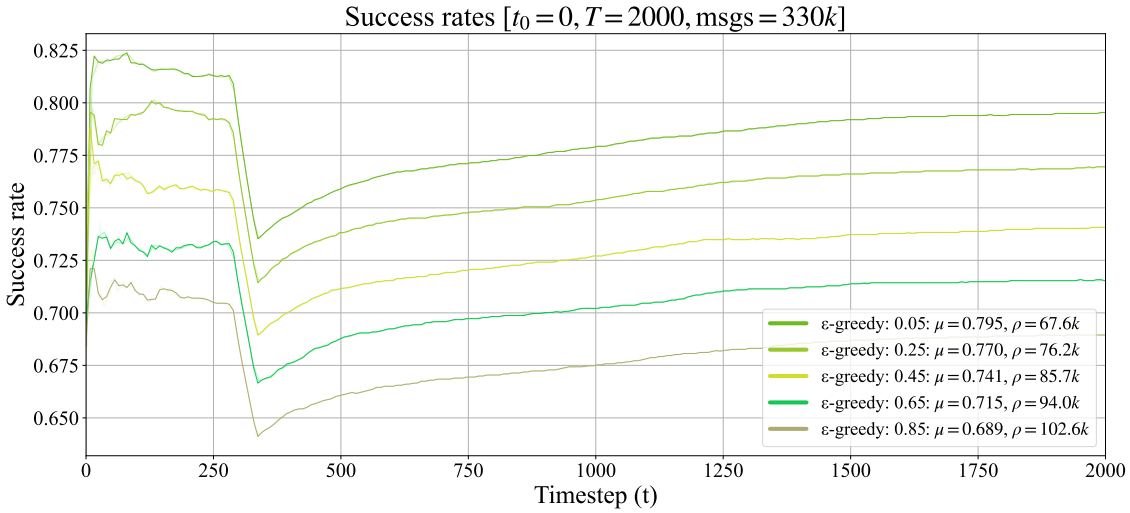

Figure 10.1. $\epsilon-$greedy tuning.

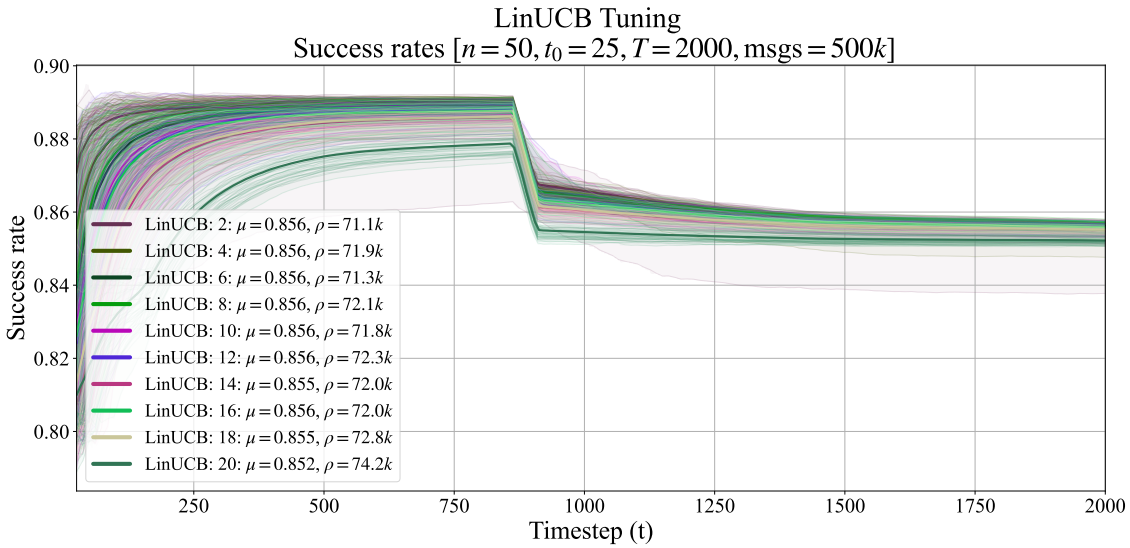

Figure 10.2. Performance metrics for LinUCB with different δ.

Table 10.1. Performance metrics for LinUCB with different δ details.

δ	Mean	Max	Min	Spread	Delivered volume
2	0.857769	0.857973	0.837751	0.020222	428820 / 499806
6	0.857389	0.857459	0.850696	0.006763	428563 / 499806
10	0.856388	0.856926	0.853201	0.003725	428297 / 499806