Karlsruhe Institute of Technology

Leveraging Context Information in Budgeted Multi-Armed Bandits

Bachelor's Thesis of

Benedikt Engel

at the Department of Informatics
Institute for Program Structures and Data Organization (IPD) Böhm

Reviewer:	TT-Prof. Dr. Pascal Friederich
Second reviewer:	Prof. Dr.-Ing. Ina Schaefer
Advisor:	Marco Heyden

11. November 2024 – 11. March 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Benedikt Engel)

Abstract

This thesis investigates the Budgeted Contextual Multi-Armed Bandit (BCMAB) problem, where a decision-maker selects one of N arms in each round t, receiving both a stochastic reward and an associated cost. The reward and cost distributions depend on contextual information x_t, which is revealed at the beginning of each round. The objective is to maximize rewards while adhering to a predefined budget, requiring an efficient trade-off between cost and reward in the decision-making process.

While existing research primarily addresses either contextual bandit algorithms or budgeted bandit frameworks in isolation, their combination remains underexplored. In this thesis, we propose novel algorithms that integrate both aspects by extending established approaches from Contextual and Budgeted Multi-Armed Bandits. These algorithms are systematically evaluated across diverse environments, focusing on cumulative regret and variance as the primary performance metrics. Furthermore, we analyze their behavior under varying conditions, provide empirical insights and hypotheses into their performance, and outline potential strategies for mitigating their limitations.

Zusammenfassung

Diese Arbeit untersucht das Budgeted Contextual Multi-Armed Bandit (BCMAB)-Problem, bei dem eine Entscheidungsinstanz in jeder Runde t einen von N Armen auswählt und sowohl eine stochastische Belohnung als auch damit verbundene Kosten erhält. Die Verteilungen von Belohnung und Kosten hängen von kontextueller Information x_t ab, die zu Beginn jeder Runde offengelegt wird. Das Ziel ist es, die Belohnungen zu maximieren, während ein vorgegebenes Budget eingehalten wird, was eine effiziente Abwägung zwischen Kosten und Belohnung im Entscheidungsprozess erfordert.

Während sich die bestehende Forschung hauptsächlich entweder mit kontextabhängigen Bandit-Algorithmen oder mit budgetierten Bandit-Frameworks separat befasst, bleibt deren Kombination weitgehend unerforscht. In dieser Arbeit werden neue Algorithmen vorgeschlagen, die beide Aspekte integrieren, indem sie etablierte Ansätze aus den Bereichen der Contextual Multi-Armed Bandits und der Budgeted Multi-Armed Bandits erweitern und kombinieren. Diese Algorithmen werden systematisch in verschiedenen Umgebungen evaluiert, wobei der kumulative Regret und die Varianz der summierten Regret's als zentrale Leistungsmetriken im Fokus stehen. Darüber hinaus analysieren wir ihr Verhalten unter unterschiedlichen Graden von Unsicherheit, liefern empirische Erkenntnisse und Hypothesen zu ihrer Performance und skizzieren potenzielle Strategien zur Minderung ihrer Einschränkungen.

Contents

List of Figures

List of Tables

1 Introduction

Every day, companies face decision-making problems under uncertainty. Online advertisers must decide which ads to show to users, e-commerce platforms need to determine the best warehouse to optimize shipping times, and financial firms optimize portfolio allocations [8, 20]. In such cases, learning from past interactions to make optimal choices in real-time is crucial.

The Multi-Armed Bandit (MAB) problem is a fundamental framework for decision-making under uncertainty. It models a scenario where the learner repeatedly selects from a set of options, known as arms, each associated with an unknown reward distribution. The learner's goal is to maximize the cumulative reward by efficiently balancing exploration and exploitation to identify the arm with the highest expected reward.

In its classical form, the MAB problem consists of a learner choosing one of N arms over T rounds. The learner aims to maximize the cumulative reward, which is the sum of obtained rewards. Since each arm's reward follows an unknown probability distribution, the learner faces an exploration-exploitation tradeoff: exploiting the arm with the highest estimated reward or exploring other arms to gather more information about their potential rewards.

Variants of this framework have a wide range of applications, from optimizing online advertisement placements to improving resource allocation in scheduling tasks. However, this type of decision-making extends far beyond online advertising and scheduling problems. It plays a crucial role in diverse fields such as network routing, dynamic pricing, tree search and medical trials [8]. The following examples illustrate MAB applications.

Example 1 (Online Advertisement)**.** In online advertisement, companies must decide which ads to display to users to maximize engagement, such as clicks or conversions. In MAB's, each ad represents a distinct arm. A new round begins when a user visits the website, prompting the advertisement algorithm, i.e. the learner, to select the ad to display. The reward is yielded if the user clicks on the displayed ad.

Example 2 (Warehouse Allocation)**.** The MAB AG is an e-commerce company that operates multiple warehouses, each with different costs, processing times, and capacities. The learner, i.e., the MAB AG, aims to maximize profit by efficiently routing orders for shipping. In this case, the reward is the profit gained by successfully selling and shipping as many items as possible. A new round is starts if a new order arrives. Whenever a new order arrives, the company must assign it to one of the warehouses while optimizing costs and delivery time to maximize profit.

Classical MAB's are not well suited for decision-making in such environments. This is due to them not including context information into their decision-making. Also, they do not take the cost-efficiency of a decision into account. To see why this is a problem, consider the examples from before:

Example 1 (continued)**.** In online advertisement, different users respond differently to ads. For example, a user with a strong interest in Tennis and a low interest in cars is more likely to click on a sports related ad than on a car related ad, even though the car related ad might have a higher average click rate. This could be due to more car interested people usually visiting the website. The additional information (i.e. that a user is interested in tennis) is called context and can include details such as age, gender, interests, or even recent purchases. Those are provided to the learner at the start of each round. Given this information, a classical MAB would still pick the car ad, since it has a higher expected reward. This is suboptimal since the click probability on the sports ad is much higher for this user. Another problem is that in online advertisement, costs arise for the advertiser whenever a user clicks on an ad. A classical MAB naively chooses the ad with the highest reward even though this might be the least cost-effective one, which leads to an inefficient utilization of advertisement budget.

Example 2 (continued)**.** Classical MAB's have issues in warehouse allocation, since warehouse conditions fluctuate and depend on external information. This information can include details such as an order's start location, destination, distance, mode of transport, warehouse capacity, and margin. Such features are critical because they influence factors like shipping time and order volume, which directly impact the revenue stream of the company. Therefore, past performance does not always implicate future efficiency. A naive strategy—such as always choosing the fastest warehouse—can lead to inefficiencies such as congestions. Also, the fastest warehouse might yield the highest cost, thereby decreasing the net profit.

Two so far separate directions of research tackle each of these challenges in isolation: *Contextual* MAB's and *Budgeted* MAB's.

In contextual MAB's, the rewards differ based on the presented context, which the learner observes at the beginning of each round. The learner observes this context information as a context vector x_t, which is a random variable that is correlated with the reward r_a of arm a through an arbitrary function with noise [8].

In the *Budgeted* MAB setting, each arm incurs a cost whenever it is chosen. The learner's goal remains maximizing the reward, but within a fixed, finite budget. This budget decreases by the cost that arise by playing an arm. And just like the reward, the cost of an arm is a random variable following an unknown distribution [7].

Many real-world applications, including the ones mentioned above, require decision-makers to simultaneously leverage context information and adhere to resource constraints. However, despite advances in both Contextual and Budgeted MAB's [10] as separate research fields, their combination has so far been overlooked. Hence, this thesis integrates both concepts into a unified framework. We refer to this approach as Budgeted Contextual Multi-Armed Bandits (BCMAB). In this setting, both reward and cost depend on contextual factors, requiring the algorithm to learn these dependencies and make cost-effective choices accordingly. In particular, we make the following contributions.

Contributions

We provided a proof of concept, establishing that contextual bandit frameworks can be effectively extended to budget-aware decision-making. This serves as a foundation for the development of novel hybrid algorithms that integrate contextual learning with budget constraints.

To achieve this, we introduced multiple contextualized budgeted bandit algorithms. Specifically, we integrated contextual models into budgeted greedy strategies and vise versa, such as *ϵ-first* [14] and *b-greedy* [16]. We incorporated Mutual Information Gain (MIG) into *GP First*, allowing the algorithm to dynamically reduce the exploration budget and minimize unnecessary exploration without requiring hyperparameter optimization.

Furthermore, we introduced several contextual Upper Confidence Bound (UCB)-based algorithms, each using distinct ways to compute confidence intervals. We extended these approaches to effectively handle budget constraints. Moreover, we incorporated MIG into *GP-based UCB algorithms*, refining their uncertainty estimation. We extended the *ω-UCB algorithm* [7] by integrating different contextual models. This integration enables the direct use of model parameters in uncertainty estimation, improving adaptability. In parallel, we adapted Thompson sampling approaches from the budgeted setting to the contextual domain.

We introduced Thompson Sampling (TS) algorithms that build upon UCB-based approaches, including *GP TS* [13, 3] and *Linear-TS* which we extended to budgeted bandits. The implementation of *Linear-TS* follows an idea proposed in the *LinUCB* publication [11], further expanding its applicability. Additionally, we introduced *β-TS* [17], which utilizes a Beta distribution to sample both rewards and costs. This algorithm is based on a budgeted bandit and is further enhanced through the usage of MIG, leading to faster variance contraction and improved learning efficiency.

To increase robustness across diverse environments, we employed the EXP4 framework [8] to aggregate various bandit algorithms, resulting in a stronger, more adaptive bandit. This ensemble-based approach proved particularly beneficial in scenarios where little prior knowledge about the environment is available.

Finally, we evaluated the proposed algorithms systematically by using default hyperparameters. The evaluation demonstrates that *Linear b-Greedy, GP-First, ω-UCB* variants, *C-LinUCB Xia, LinUCB Tor, GP-TS, and GP β-TS* performed exceptionally well and exhibited high stability across different environments. We further analyzed and interpreted the results, leading to several hypotheses proposed to explain the observed behavior.

These contributions advance the state of research in budgeted contextual bandits by bridging the gap between contextual learning and cost-aware decision-making, paving the way for further research in both theoretical guarantees and industry applicability.

All implementation details, including code and experimental setups, are publicly available on GitHub[1], enabling further research and reproducibility.

[1] github.com/Benedikts-stuff/BCMAB

Thesis Outline

The remainder of this thesis is organized as follows: Chapter 2 introduces the problem setting and defines the evaluation metrics used to measure algorithm performance. Chapter 3 provides an overview of related work. Chapter 4 describes the proposed algorithms and their underlying principles. Chapter 5 details the experimental setup. Chapter 6 presents the results and discusses their implications. Chapter 7 concludes the thesis and outlines potential directions for future research.

2 Fundamentals

This chapter introduces fundamental concepts and theoretical foundations of Multi-Armed Bandits (MAB). It explains the basic mechanics, key challenges, and different variations of MAB problems. These fundamentals provide the necessary background to understand the algorithms discussed in later chapters.

In the classic stochastic Multi-Armed Bandit (MAB) problem, the learner must choose between N arms over T rounds. As we illustrated in figure 2.1, the reward r_a for each arm $a \in \mathcal{A}$ follows an unknown distribution $\mathcal{D}_a$. For any arm, the expected reward is denoted by $\mu_a = \mathbb{E}[r_a]$.

2.1 The Multi Armed Bandit Problem

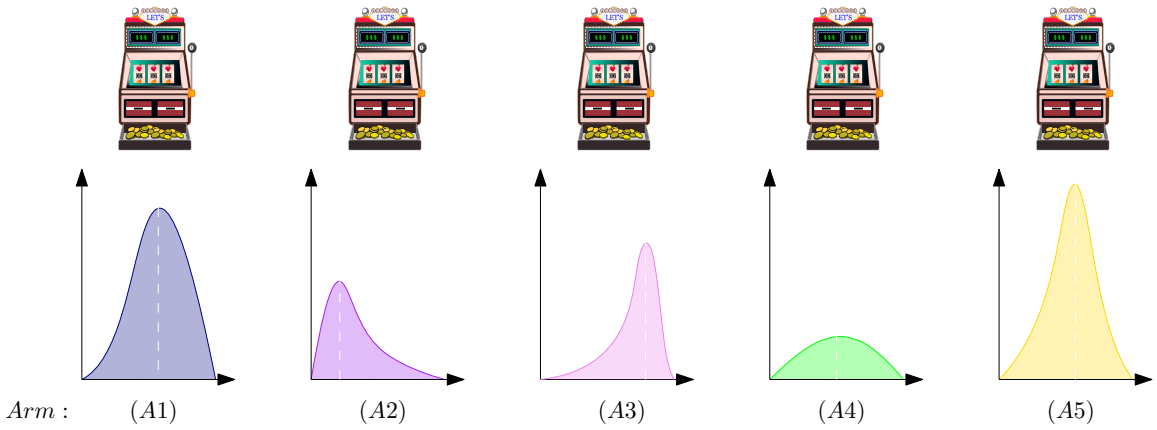

Figure 2.1: The Multi-Armed Bandit Problem, based on [4]

At each round $1 \leq t \leq T$, the learner selects an arm $a \in \mathcal{A}$, where $\mathcal{A}$ represents the set of all available arms. The learner observes a reward $r_a \sim \mathcal{D}_a$ for the chosen arm a, drawn from its unknown distribution.

The learner's goal is to maximize the cumulative reward over T rounds. This is equivalent to minimizing the cumulative regret over T rounds, which we define as the summed difference between the expected reward of the chosen arm μ_a and the best arm $\mu_{a^*} = \max_{a \in \mathcal{A}} \mu_a$, as shown in (2.1):

$$\text{Regret}(T) = T\mu_{a^*} - \sum_{t=1}^{T} \mu_{a_t}. \tag{2.1}$$

If we take a look at Figure 2.1, the arm with the highest expected reward is $A3$. Therefore, the learners' goal is to play $A3$ as frequently as possible while becoming more certain

about its guess. In order to learn the best arm, the learner has to gather data on all other arms as well. This is called the *exploration-exploitation dilemma*: since the true reward distributions of the arms are unknown, the learner must balance exploiting arms that yielded high rewards in the past with exploring other arms, to gain knowledge [8].

2.2 The Budgeted Multi Armed Bandit Problem

A natural extension of the classic MAB problem is the Budgeted Multi-Armed Bandit (BMAB). In BMAB's, the learner observes a reward r_a and cost c_a for each arm $a \in \mathcal{A}$, which both follow an unknown distribution with the expected values denoted as $\mu_a^r = \mathbb{E}[r_a]$ and $\mu_a^c = \mathbb{E}[c_a]$. Unlike classic MAB's where the reward exclusively determines the choice of an arm, the learner must now balance between maximizing the cumulative reward while staying within a fixed total budget B.

The key challenge in the BMAB problem is to select arms that offer the best trade-off between reward and cost, i.e. the most cost-effective arms. Simply choosing the arm that corresponds to $\mu_{a^*}^r = \max_{a \in \mathcal{A}} \mu_a$ is often suboptimal, as higher rewards may come with high costs. The learner has to develop a strategy that minimizes the cumulative regret while ensuring that the summed costs do not exceed the budget B. To formalize this, we introduce the function $T_\pi(B)$ that denotes the number of rounds played under policy π and budget B:

$$T_\pi(B) = \sum_{\substack{t=1 \\ B_t \geq 0}}^{\infty} \mathbb{1}\{B_t \geq 0\}. \tag{2.2}$$

The cumulative regret is then described by equation (2.2) [16]:

$$\text{Regret}(T_\pi(B)) = T_\pi(B)\mu_{a^*} - \sum_{t=1}^{T_\pi(B)} \mu_{a_t}^r$$

Notice that in the budgeted bandit setting, the number of rounds played is not a fixed constant T anymore, since it is directly influenced by the arm selection policy.

A further extension of the MAB problem is to include contextual information, allowing the learner to adapt its decisions based on additional features describing each round's conditions.

2.3 The Budgeted Contextual Multi-Armed Bandit Problem

Algorithms incorporating contextual information and budget constraints are referred to as Budgeted Contextual Multi-Armed Bandits (BCMAB's).

In BCMAB's, the learner observes context information in the form of a context vector $\boldsymbol{x_t} \in \mathcal{X}$ at the beginning of each round t. The context itself is a random variable drawn from an unknown distribution. The reward and cost r_a, c_a are now functions of the context

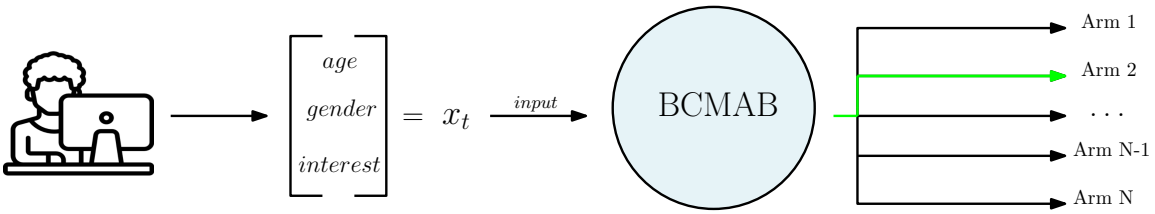

Figure 2.2: Leveraging Context information in Multi-Armed Bandits

that predict the mean reward of arm a. The learner now aims to select the optimal arm for a given context by learning the functions

$$r_a, c_a : \mathcal{X} \to \mathbb{R}, \quad \boldsymbol{x} \mapsto r_a(\boldsymbol{x}),\ c_a(\boldsymbol{x})$$

of each arm that map the presented context to the reward and cost. This connection follows either a linear or a non-linear relationship. Figure 2.2 illustrates this concept using a web-based recommendation scenario: When a user visits a website, attributes such as age, gender, and interests are available. The learner then selects the most cost-effective arm for this context. We find it important to note that the attributes do not need to belong to the same domain. For example, gender and interests are categorical variables, while age is numerical. Our key requirement is the existence of a function that maps the reward based on these attributes.

To efficiently select arms, the learner aims to approximate the functions $r_a(\boldsymbol{x})$ and $c_a(\boldsymbol{x})$, which predict the reward and cost for each arm $a \in \mathcal{A}$ given a context $\boldsymbol{x}$, with high accuracy. Since the context is a random variable and the optimal arm in each round depends on it, the identity of the best arm in each round is itself a random variable.

To measure performance in this setting, we consider the expected cumulative regret, which quantifies the total difference between the reward of the optimal arm and the chosen arm over all rounds. Formally, we define this as [8]:

$$\text{Regret} = \mathbb{E}\left[\sum_{t=1}^{T_\pi(B)} \left(\max_{a\in\mathcal{A}} r_a(\boldsymbol{x_t}) - r_{a_t}(\boldsymbol{x_t})\right)\right]$$

where a_t denotes the arm selected by the learner in round t. The selection process operates under a fixed budget B, where each chosen arm reduces the remaining budget by its respective cost $c_a(\boldsymbol{x_t})$ [8].

2.4 Bandit Algorithm Families

The following section provides an overview of the three main families of algorithms used in this work.

2.4.1 Greedy Algorithms

In Greedy algorithms, the learner wants to select the best arm solely on the model's predictions. Doing this exclusively leads to no exploration, meaning the algorithm always exploits the same arm without knowing if it is the best. To address this, most greedy algorithms, such as ϵ-Greedy, pick the arm with the highest expected reward with a probability of $1-\epsilon$, where ϵ is a parameter that controls exploration. Setting ϵ as a constant would result in linear regret, since the algorithm never stops exploring. Therefore, some algorithms in this family use an adaptive exploration parameter α, which decreases as more rounds are played, encouraging exploration early on and gradually shifting towards exploitation [2]. A similar approach is found in phased based algorithms, such as the ϵ-First algorithm proposed by Tran-Thanh et al. Phase based means that the exploration-exploitation dilemma is dealt with by dividing the algorithm into 2 distinct phases. The

- **Exploration Phase:** uniformly explore all the arms until the exploration budget defined as $\epsilon \cdot B$ is drained.
- **Exploitation Phase:** with the remaining budget $B - (\epsilon \cdot B)$, only play the best arm based on the information from the exploration phase and never explore again.

By following this approach, the algorithm can achieve very low regret when the learner selects the correct arm. However, if the learner chooses an incorrect arm, the regret can rise significantly higher. The advantage of this approach is that, in favorable conditions, it can lead to highly efficient decision-making. However, its downside is the potential for extreme regret fluctuations, making it unsuitable for applications requiring consistent and predictable performance.

2.4.2 Upper Confidence Bound Algorithms

The Upper Confidence Bound (UCB) algorithms are based on the principle of *optimism in the face of uncertainty*, which suggests that the learner should act as if the environment is as favorable as possible [8]. To make this idea clear, let's consider an example adapted from Lattimore et al. [8]:

Example 3 (Optimism in the Face of Uncertainty)**.** Imagine being a medical researcher exploring new treatments for a disease. Some treatments are well-researched with known outcomes, while others are new and untested. An optimistic approach to the untested treatments encourages exploration, as you believe they could hold the key to a breakthrough, even though there is little prior data. A pessimistic approach would focus on the established treatments, potentially missing out on a more effective treatment.

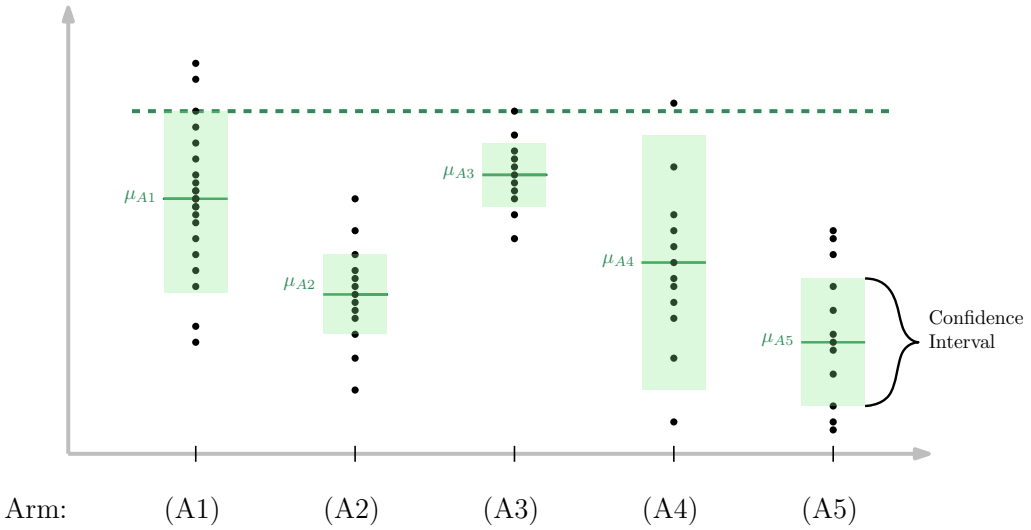

Figure 2.3: Confidence Intervals in Multi-Armed Bandits

The upper confidence bound mathematically captures the concept of optimism in the face of uncertainty. This value overestimates the actual reward of an arm with high probability. In essence, we compute a 1-δ confidence interval for the expected reward μ_a^r of each arm $a \in \mathcal{A}$, and only take the upper bound of this interval into account to decide which arm to chose [8]:

$$UCB_a(t-1,\delta) = \mu_a^r(t-1) + \Delta \tag{2.3}$$

Here, $\mu_a^r(t-1)$ is the estimated reward based on information from the previous $t-1$ rounds. The parameter δ represents the probability that the actual reward r_a is greater than the estimate. In equation (2.3) Δ stands for the confidence. A large value of Δ means high uncertainty and low confidence, whereas a low value means the opposite [8].

To further illustrate this concept, consider Figure 2.3, which represents the state of a MAB at round t in the context of Example 3. In this scenario, the learner selects the medication corresponding to arm $A1$, because it has the highest upper confidence bound (UCB), even though its expected reward is lower than the reward of other arms. It is crucial to understand that these confidence intervals are dynamic; they evolve as the number of observations increases. As the best arm accumulates more evidence through frequent selection, its confidence interval narrows, reducing uncertainty about its true mean and shifting the balance between exploration and exploitation. In this case, the learner tends to select arms with greater uncertainty and a higher UCB. This leads to exploration.

2.4.3 Posterior Sampling Based Algorithms

In posterior sampling-based algorithms, the learner maintains a posterior distribution over the rewards for each arm $a \in \mathcal{A}$, and the rewards follow an unknown distribution. Unlike UCB or greedy algorithms that use point estimates like the expected reward, these algorithms sample from the posterior to make decisions. This sampling induces an implicit

exploration, as the learner is more likely to explore arms with higher uncertainty (greater variance) even if their mean is lower.

Let r_a denote the reward for arm a. Initially (at $t = 0$), we assume a prior belief $p(\theta_a)$ on the parameter θ_a, which governs the reward distribution of arm a. After each round, the learner updated the prior of the selected arm based on the observed rewards $r_{a,1}, \ldots, r_{a,t}$ to form a posterior distribution:

$$p(\theta_a \mid r_{a,1}, \ldots, r_{a,t}) \propto p(r_{a,1}, \ldots, r_{a,t} \mid \theta_a) p(\theta_a).$$

Here, $\mathcal{D}_t = \{r_{a,1}, \ldots, r_{a,t}\}$ represents the data collected up to time t. The posterior reflects the updated belief about θ_a. In round $t + 1$, the posterior from round t serves as prior. By iteratively refining this belief, the algorithm balances exploration and exploitation based on both observed rewards and remaining uncertainty.

2.4.4 Bandits with Expert Advice

The *Bandits with Expert Advice* (BwEA) framework extends the standard multi-armed bandit setting by incorporating multiple experts E from a set of experts $\mathcal{E}$. In each round, every expert provides a prediction for the expected reward of each arm a based on the context $\boldsymbol{x}$. Instead of selecting an arm based solely on past observations, the algorithm maintains a probability distribution Q over the expert set $\mathcal{E}$.

At the beginning of each round, the learner samples an expert from Q and follows the recommendation of the selected expert, choosing the arm with the highest predicted reward. After the learner observes the reward, they update the belief about the experts and the probability distribution Q, gradually increasing trust in experts who provide more accurate predictions.

This framework is particularly useful in scenarios where external knowledge or structured information is available, allowing the bandit to benefit from multiple information sources rather than relying solely on trial-and-error learning.

2.5 Leveraging Context Information

There are various ways to incorporate context into MAB's. To learn the relationship between context and rewards or costs, we can use different models In this section, we explain the models used in this thesis. We focus on ridge regression, Gaussian processes, and neural networks, as these are most frequently used in related work.

Before proceeding, we introduce the concept of feature maps, which provide a method for transforming features from different domains into a common representation.

2.5.1 Feature Maps

We assume that the learner has access to a feature map. A feature map is a function that maps context vectors into the real feature space:

$$\psi : \mathcal{X} \to \mathbb{R}^d$$

Notice that the feature space must not have the same dimension as the original context space. With the now numerical context features, it is easier to learn the reward-cost-context relationship.

In the following, we assume that the context vector denoted as $\boldsymbol{x}$ has already undergone transformation into the feature space.

2.5.2 Linear Model

Both cost and reward depend on the context, either in a strictly linear manner or in a form that a linear model can approximate effectively. A linear model can learn this relationship, where the weight vector $\theta_a \in \mathbb{R}^d$ captures the influence of each feature on the reward r_a for the arm a. The linear model predicts the expected reward for a given context vector $\boldsymbol{x} \in \mathbb{R}^d$ as:

$$\hat{r_a}(\boldsymbol{x}) = \boldsymbol{x}^\top \theta_a.$$

The goal is to iteratively learn θ_a from observed data to improve future decisions. In our case, each arm has a distinct model that the learner updates only for the chosen arm.

To estimate the parameter vector θ_a, the model maintains a data matrix $A_a \in \mathbb{R}^{d \times d}$ and a vector $b_a \in \mathbb{R}^d$. Initially, we set A_a to the identity matrix $A_a = \mathbb{I}_d$ and b_a to a zero vector $b_a = \mathbf{0}$ of the same dimension as A_a. The model then computes the parameter vector θ_a as:

$$\theta_a = A_a^{-1} b_a.$$

At the end of each round, the learner selects an arm and observes a reward $r(\boldsymbol{x})$. With the new observation, the learner updates the model parameters as follows:

- The matrix A_a accumulates the outer product of the context vector:

$$A_a = A_a + \boldsymbol{x}\boldsymbol{x}^\top.$$

- The vector b_a accumulates the context vector scaled by the observed reward:

$$b_a = b_a + r(\boldsymbol{x})\boldsymbol{x}.$$

As the number of rounds increases, the difference between the estimated parameter vector θ_a and the true parameter vector θ_a^*, that defines the true reward relationship $r(\boldsymbol{x}) = \boldsymbol{x}^\top \theta_a^*$, decreases. Under a sufficient number of runs through this cycle, θ_a converges to θ_a^*.

We can visualize this by plotting the frobenius norm of the estimated weight vector θ_a and the actual weight vector θ_a^* of the best arm in the LinUCB algorithm. As the number of rounds increases, the norm quickly approaches 0 which leads to increasingly precise prediction of the reward of the respective arm.

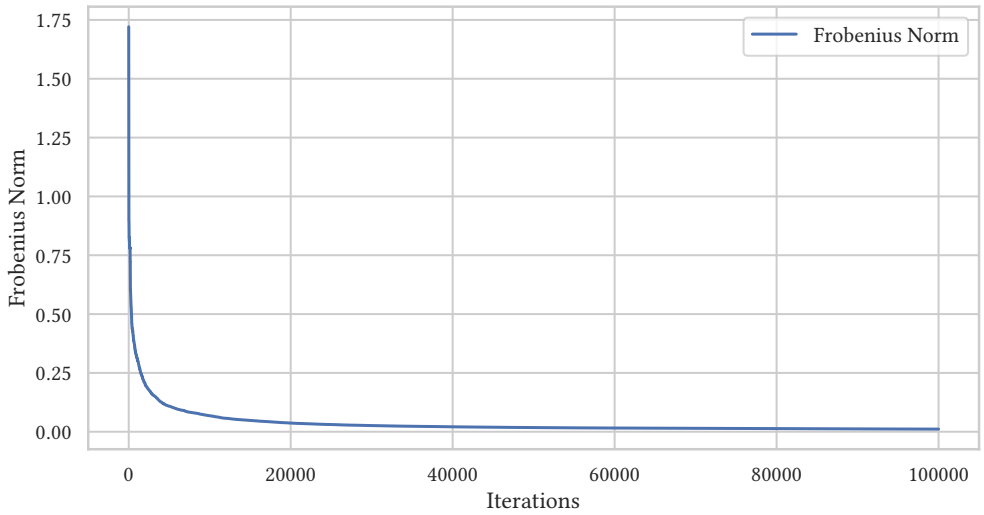

Figure 2.4: Frobenius Norm of θ_a vs θ_a^* in LinUCB

2.5.3 Gaussian Process

A Gaussian Process (GP) is a probabilistic model that maintains a distribution over all possible functions. GP's are particularly useful for learning complex relationships, such as the mapping between contexts and rewards in a Contextual Multi-Armed Bandit setting. The idea behind a GP is that contexts which lay close to each other in the context space have a similar reward probability. The kernel function describes this similarity. For this work, we use the Radial-Basis-Function-Kernel (RBF-Kernel). We use the RBF-Kernel to measure the similarity between contexts in a non-linear way, with the similarity decaying exponentially as the distance between two contexts increases.

For an observed context $\boldsymbol{x}$, a GP outputs

- an expected value $\mu(\boldsymbol{x})$ as a prediction of the reward or cost
- an uncertainty measure $\sigma(\boldsymbol{x})$, which represents how certain the model is about the prediction

Uncertainty is a measure of how 'unfamiliar' the new context is, with higher uncertainty indicating that the model has not seen similar contexts before and is less confident about its predictions for this context. The GP's prediction is based on the covariance structure described by the Kernel. It determines how strong the observed context $\boldsymbol{x}_*$ and the training data, i.e. previously observed contexts, are correlated [12]:

- $k(x_*, X_{\text{train}})$: Vector to describe similarity between x_* and each item of the training set based on kernel.
- $K(X_{\text{train}}, X_{\text{train}})$: Covariance Matrix of the training data. Describes the similarity between all items of the training set.
- y_{train}: Set of observed rewards or costs.

Notice, that a GP makes the assumption that all elements of y_{train} are samples from a Multivariate Gaussian. With these definitions and assumptions, the predicted reward and uncertainty is computed as shown in equations (2.4) and (2.5) [12].

$$\mu(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, X_{\text{train}}) \cdot K(X_{\text{train}}, X_{\text{train}})^{-1} \cdot y_{\text{train}} \tag{2.4}$$

$$\sigma^2(\boldsymbol{x}_*) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - k(\boldsymbol{x}_*, X_{\text{train}}) \cdot K(X_{\text{train}}, X_{\text{train}})^{-1} \cdot k(X_{\text{train}}, \boldsymbol{x}_*) \tag{2.5}$$

After the GP receives a new context-reward pair, it incorporates the new data by updating its covariance matrix. This refines the model's predictions for future contexts [12]:

$$K(X_{\text{train}} \cup \{\boldsymbol{x}_*\}, X_{\text{train}} \cup \{\boldsymbol{x}_*\}),$$

where $\boldsymbol{x}_*$ is the newly observed input (context) and X_{train} represents the existing training data. Additionally, we should re-tune the hyperparameters of the kernel function to account for the new data.

2.5.4 Neural Network

Neural Networks are a class of machine learning models that excel at learning complex, non-linear mappings between inputs and outputs. We describe the components of the neural network in the following.

The architecture of a neural network determines its depth (number of layers) and width (number of neurons per layer). The depth of the network allows it to capture hierarchical and abstract features of the input data, while the width controls the capacity to learn fine-grained patterns. For a given context $\boldsymbol{x}$, the network applies successive transformations through its layers, which we defined as:

$$\boldsymbol{h}^{(l+1)} = \sigma(\boldsymbol{W}^{(l)}\boldsymbol{h}^{(l)} + \boldsymbol{b}^{(l)}), \quad l = 1, \ldots, L,$$

where:

- $\boldsymbol{W}^{(l)}$ and $\boldsymbol{b}^{(l)}$ are the weight matrix and bias vector of the l-th layer.
- $\boldsymbol{h}^{(l)}$ represents the activations of the l-th layer.
- $\sigma(\cdot)$ is the activation function, which introduces non-linearity into the model.

In this thesis, we use the Rectified Linear Unit (ReLU) activation function, which is defined in equation (2.6).

$$\sigma(z) = \max(0, z). \tag{2.6}$$

ReLU is computationally efficient and mitigates the vanishing gradient problem.

We can then get a prediction for the reward $\hat{r_a}(\boldsymbol{x})$ by propagating $\boldsymbol{x}$ through the network layers l as shown above, with the final layer outputting a scalar value:

$$\hat{r_a}(\boldsymbol{x}) = \boldsymbol{h}_{\boldsymbol{a}}^{(L)}.$$

This value serves as the model's estimate of the expected reward for arm a with the given context.

We compute the uncertainty of the prediction by using the covariance matrix of the model parameters:

$$\text{uncertainty} = \sqrt{\frac{\lambda}{m} \cdot \boldsymbol{g}^\top \boldsymbol{C}^{-1} \boldsymbol{g}},$$

where:

- λ is a regularization parameter.
- m is the total number of observations seen by the model.
- $\boldsymbol{g}$ is the gradient of the predicted reward with respect to the model parameters, $\boldsymbol{g} = \nabla_\theta \hat{r}(\boldsymbol{x})$.
- $\boldsymbol{C}$ is the covariance matrix of the model parameters.

This works because it captures the propagation of uncertainty in the parameter space to the output space, using the gradient $\boldsymbol{g}$ as a measure of how sensitive the output is to small parameter changes. The covariance matrix $\boldsymbol{C}$ incorporates information about the variability of the parameters, and regularization ensures that the uncertainty estimation remains robust.

We update the neural network in batches of size k and adjust the model parameters using gradient descent with the Mean Squared Error loss function.

$$\mathcal{L} = \frac{1}{k} \sum_{i=1}^{k} \left(\hat{r_{a,i}}(\boldsymbol{x}_i) - r_{a,i} \right)^2, \tag{2.7}$$

In (2.7) $\hat{r_{a,i}}(\boldsymbol{x}_i)$ represents the predicted reward for context $\boldsymbol{x}_i$, $r_{a,i}$ is the observed reward for arm a and k represents the batch size.

Gradient descent minimizes the MSE loss to optimize the network's weights and biases. We use batch updates with size k to balance computational efficiency and gradient stability, reducing oscillations from frequent updates.

3 Related Work

Research on Multi-Armed Bandits (MABs) has evolved significantly, leading to multiple extensions that incorporate real-world constraints, such as budget limitations and contextual dependencies. In this chapter, we provide an overview of existing algorithms in budgeted and contextual bandits, as well as hybrid approaches that combine both perspectives.

We first introduce budgeted bandit algorithms, which optimize reward while considering cost constraints. Next, we discuss contextual bandits, where the reward of an arm depends on observed contextual features. Finally, we review algorithms that integrate both budget-awareness and contextual decision-making, positioning this thesis within the research landscape.

3.1 Budgeted Bandit Algorithms

Classical Multi-Armed Bandit (MAB) algorithms focus solely on maximizing rewards, without considering the costs associated with each arm. Budgeted Bandit algorithms extend this framework by introducing budget constraints, optimizing the trade-off between rewards and costs. This section summarizes key approaches to budget-aware bandit optimization. A common approach is to maximize the reward-to-cost ratio.

Xia et al. apply this idea in their *b-greedy* algorithm, an extension of the classical ϵ-greedy algorithm. Instead of selecting the arm with the highest expected reward with $1 - \epsilon$ probability, *b-greedy* chooses the arm with the highest $\frac{reward}{cost}$ ratio [16]. To address the issue of linear regret with a fixed ϵ, they propose an adaptive exploration parameter that decreases as the number of rounds increases.

A more robust strategy for budget-aware exploration is provided by Xia et al.'s *Budgeted Thompson Sampling* (BTS) algorithm. BTS models rewards and costs for each arm using Beta distributions and iteratively updated them after an arm is played. The algorithm samples reward and cost estimates from these distributions and selects the arm with the highest sampled $\frac{reward}{cost}$ value [17].

The principle of Optimism in the Face of Uncertainty is also applicable to budgeted bandits. Xia et al. demonstrate this with their *i-UCB* algorithm, which uses an upper confidence bound (UCB) approach based on the expected reward-to-cost ratio rather than the expected reward alone. However, this algorithm does not differentiate between uncertainty in the reward and cost estimates. To address this limitation, they propose *Budget-UCB*, which separately models the uncertainty for both reward and cost [15].

An inherent challenge in budgeted ban dits is the potential for suboptimal decisions when the cost distribution is U-Shaped, i.e. there is a high probability of sampling a value near the interval boundaries. Heyden et al. tackle this issue by introducing asymmetric

confidence intervals for both rewards and costs. Their approach essentially pulls the confidence bounds more towards the mean, leading to more optimal decisions [7].

3.2 Contextual Bandit Algorithms

A further extension of classical Multi-Armed Bandits are Contextual Bandits. We can group these algorithms into two sets: *Linear Bandits* i.e. bandits that utilize a linear model to learn the cost-reward-context relationship and *Non-Linear Bandits* i.e. bandits that utilize more complex models to learn relationships beyond linearity.

3.2.1 Linear Bandits

Many contextual bandit algorithms focus on learning a linear function to model the relationship between context and reward. One prominent example is *LinUCB* by Li et al., which leverages the inverse summed covariance matrix of the observed data to quantify uncertainty in the computation of the upper confidence bound [11].

A similar principle underlies variants of the *Linear Thompson Sampling* algorithm. Li et al. proposed using the inverse summed covariance matrix of the data as the covariance parameter of a multivariate Gaussian distribution. The mean of this distribution corresponds to the weight vector θ of the linear model from the previous round. The learner then samples a new weight vector for the current round from this distribution and uses it to select an arm.

Agrawal et al. followed a similar approach, but directly sampled the expected reward instead of the weight vector itself. In their formulation, the mean is computed as the dot product of the learned weight vector with the observed context, while the standard deviation is derived from the Mahalanobis distance of the context vector using the inverse summed covariance matrix similar to LinUCB [1].

3.2.2 Non-Linear Contextual Bandits

Despite their effectiveness, algorithms based on linear models face a significant limitation: the assumption of a linear relationship between context and reward. In practice, this relationship is often non-linear. To address this, researchers have explored various models capable of capturing non-linear relationships, such as neural networks and kernelized methods like Gaussian processes.

For instance, Zhang et al. introduced *Neural Thompson Sampling*, which combines Thompson sampling with a neural network. In this approach, the expected reward is sampled from a multivariate Gaussian distribution, where the mean is the neural network's predicted reward for the given context $\mathbf{x}_t$, and the variance is derived from the model's gradients [19].

Another powerful framework for modeling non-linear relationships is *Gaussian Processes (GPs)*. Srinivas et al. applied this concept in their *GPUCB* algorithm, which uses a Gaussian process to maintain a probabilistic distribution over possible reward-context functions [13]. Similarly, Chowdhury et al. adapted Gaussian processes for use with Thompson

sampling [3]. To quantify uncertainty, they incorporate the mutual information gain into the computation of the Confidence interval.

3.2.3 Bandits with Expert Advice

The most well known algorithm within this framework is *EXP4* (Exponential-weight algorithm for Exploration and Exploitation with Experts), which efficiently combines expert advice with bandit exploration strategies. EXP4 maintains weighted probabilities for experts, adjusting them using an exponential weighting scheme to favor those who consistently provide high rewards while still exploring uncertain predictions [8].

3.2.4 Handling High-Dimensional Context Spaces

High-dimensional context spaces pose additional challenges, as many algorithms experience increased regret with higher-dimensional data. Yue et al. proposed a solution to this with their *CoFineUCB* algorithm, which leverages a hierarchical feature representation. CoFineUCB uses a lower-dimensional projection of the context data to accelerate convergence. However, it depends on certain environmental constraints, such as a discretized context space and prior knowledge for learning the feature hierarchy [18].

4 BCMAB Algorithms

In this chapter, we introduce algorithms that integrate budgeted contextual bandit policies. To formalize these algorithms, we define a generic framework for Budgeted Contextual Multi-Armed Bandits (BCMAB) in 1.

Algorithm 1 Budgeted Contextual Bandit Framework

Require: Number of arms N, number of features d, context set $\mathcal{X}$, total budget B, model set $\mathcal{M}$

1: Initialize model's
2: Set current budget $b \leftarrow B$
3: **while** $b > 0$ **do**
4: Observe context $\boldsymbol{x} \in \mathcal{X}$
5: Select arm $a \leftarrow \pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t, B, b)$
6: Observe reward $r_a(\boldsymbol{x})$ and cost $c_a(\boldsymbol{x})$
7: Update model M_a
8: Update budget $b_{t+1} \leftarrow b_t - c_a(\boldsymbol{x})$

The policy π and model set $\mathcal{M}$ depend on the specific algorithm employed. We associate each arm with a distinct reward and cost models, denoted by M_a^r and M_a^c, respectively. The contextual models used are detailed in Chapter 2.5, and we will not reintroduce them in this chapter. Instead, this chapter focuses on presenting the modifications and extensions relevant to budget and context aware decision-making.

To minimize redundancy, algorithmic variations with similar structures are grouped within the same sections. Additionally, to ensure clarity in notation, frequently recurring parameters are summarized in Table 4.1.

4.1 Greedy

This section introduces algorithms based on the greedy selection policy introduced in section 2.4.1. These algorithms prioritize immediate reward maximization without additional measures to quantify uncertainty, and are adapted by incorporating contextual information to mitigate long-term regret.

4.1.1 Linear ϵ-First

The contextual adaption ϵ-First is compatible with any set of predictive models $\mathcal{M}$. The algorithm consists of two distinct phases: a pure exploration phase and a pure exploitation

Parameter	Definition
N	Number of arms
d	Context dimension
$\mathcal{X}$	Context space
B	Total budget
$\mathcal{M}$	Set of models
b	Remaining budget
$\boldsymbol{x}$	Context vector
a^*	Selected arm
π	Policy
$r_a(\boldsymbol{x})$	True reward of arm a
$c_a(\boldsymbol{x})$	True cost of arm a
M_a	Model associated with arm a

Table 4.1: Notation overview

phase. The proportion of rounds dedicated to exploration is determined by the exploration budget b_e. We define it as $b_e = \epsilon B$, where $\epsilon \in [0, 1]$ is a fraction of the total budget B. During the exploration phase, the learner collects data to update its models [14]. Once the learner completes the exploration phase, they continue to select the arm with the highest predicted reward-to-cost ratio in every subsequent round based on the data obtained from exploration. Algorithm 2 describes the arm selection policy formally.

Algorithm 2 Linear ϵ-First

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, B)$
  Compute exploration budget: $b_e = \epsilon B$
  if $b_e > B - b$ **then** ▷ b is the remaining budget
    Select arm $a \sim \mathcal{U}(\mathcal{A})$ ▷ Exploration phase
  else
    for $a \in \mathcal{A}$ **do**
      $p_a \leftarrow M_a^r(\boldsymbol{x})$ ▷ Predicted reward
      $q_a \leftarrow M_a^c(\boldsymbol{x})$ ▷ Predicted cost
    $a^* \leftarrow \arg\max_a \frac{p_a}{q_a}$ ▷ Exploitation phase
  return a^*

4.1.1.1 Introducing an adaptive Exploration Budget

The fixed Linear ϵ-First algorithm can lead to unnecessary exploration if the model is already well-trained. To mitigate this issue, we incorporate a Gaussian Process (GP) and dynamically adjust the exploration budget based on the mutual information gain (MIG) obtained from observing context $\boldsymbol{x}$. We call this algorithm *GP-First*. The MIG is defined as [5]:

$$\gamma = \frac{1}{2} \log \det \left(I + \frac{K_{t-1}}{\eta^2} \right) \tag{4.1}$$

where K_{t-1} represents the covariance matrix of the Gaussian Process after $t-1$ rounds. The parameter η^2 is a parameter of the Gaussian Process modeling the noise variance of the data [13, 5]. The intuition behind this is that if the MIG is high, this observation will reduce uncertainty of the model by a large amount. This is because a high MIG means high uncertainty prior to the observation and indicates that the model needs more data. In this case, the learner should decrease the exploration budget slower than normal to explore more. In the counter event where we have a low MIG, the learner can decrease the exploration budget faster since the model is already accurate enough.

To integrate γ into the ϵ-First algorithm, the learner can dynamically adjust the exploration budget by reducing it inverse proportional to the obtained MIG. After selecting arm a in round t and observing its associated cost and reward, the learner updates b_e as shown in (4.2).

$$b_e = b_e - \max(1, \frac{2}{\mu_a(\gamma^r) + \mu_a(\gamma^c)}) \cdot c_a \tag{4.2}$$

The parameter c_a represents the observed cost for arm a in round t. The term $\mu_a(\gamma)$ refers to the mean MIG over all arms. The exploration budget is adjusted adaptively based on the MIG from both the reward model (γ^r) and the cost model (γ^c). Since both models contribute to the overall understanding of the decision process, their respective MIG's are summed to form a combined measure of knowledge acquisition. The factor 2 ensures that if both models exhibit similar levels of information gain, the reduction term approximates $1/\gamma^r$, preventing premature exploitation.

4.1.2 Static Exploration ϵ-Greedy

We introduce a simple greedy policy that incorporates exploration with a fixed probability of $\epsilon = 0.1$. The exploration rate remains constant throughout all rounds but can be adjusted depending on the specific requirements of the environment [8]. Algorithm 3 formalizes the arm selection policy.

Algorithm 3 Linear ϵ-Greedy

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  $\epsilon \leftarrow 0.1$
  if $p \sim \mathcal{U}(0,1) < \epsilon$ **then**
    Select arm $a \sim \mathcal{U}(\mathcal{A})$ ▷ Exploration
  else
    for $a \in \mathcal{A}$ **do**
      $p_a \leftarrow M_a^r(\boldsymbol{x})$ ▷ Predicted reward
      $q_a \leftarrow M_a^c(\boldsymbol{x})$ ▷ Predicted cost
    $a^* \leftarrow \arg\max_a \frac{p_a}{q_a}$ ▷ Exploitation
  return a^*

The policy follows a simple structure: with probability ϵ, the learner selects a random arm for exploration. Otherwise, they choose the arm with the highest predicted reward-to-cost ratio.

A fundamental limitation of this policy is that it results in linear regret, even if the model perfectly captures the reward-cost-context relationship. This issue arises because the exploration rate ϵ is fixed, meaning that exploration continues indefinitely. To address this limitation, Xia et al. introduced an adaptive exploration rate that decreases exploration over time. We discuss this refinement in Section 4.1.3.

4.1.3 Linear b-Greedy

The modification of Linear ϵ-Greedy (section 4.1.2) follows b-Greedy proposed by Xia et al. [17]. This adaptation introduces a dynamically decreasing exploration rate ϵ, leading to logarithmic regret. Xia et al. defines the adaptive exploration probability as [17]:

$$\epsilon = \min\left(1, \frac{\alpha \cdot N}{t}\right) \tag{4.3}$$

In equation (4.3), α is a scaling factor, N is the number of arms, and t denotes the current round. As the number of rounds increases, ϵ gradually decreases, ensuring high exploration in early stages while shifting towards exploitation over time. This approach significantly improves efficiency compared to static ϵ-Greedy, as it balances exploration and exploitation dynamically.

We illustrated equation (4.3) in figure 4.1, which shows the exploration probability over $t = 1000$ rounds for an ($N = 5$) armed bandit with $\alpha = 4$.

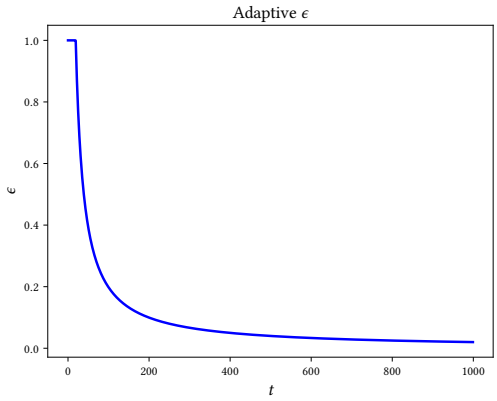

Figure 4.1: Adaptive exploration probability ϵ over $t = 1000$ for an $N = 5$ armed bandit.

4.2 Upper Confidence Bound

This section presents algorithms that utilize Upper Confidence Bound (UCB) policies to balance exploration and exploitation. UCB-based approaches construct confidence intervals around reward and cost estimates to quantify uncertainty, guiding arm selection towards promising arms. These methods are theoretically grounded in the principles discussed in Section 2.4.2.

4.2.1 Budget Constrained LinUCB

The LinUCB algorithm by Li et al. is a contextual bandit approach that estimates an upper confidence bound (UCB) for the expected reward of each arm based on the given context [11]. They achieve this by utilizing the inverse data matrix A_a^{-1}, derived from the linear model explained in chapter 2.5.2. We extend LinUCB to a budget aware algorithm by computing a lower confidence bound (LCB) for cost estimates in the same way as the upper confidence bound for the reward. Algorithm 4 shows the resulting arm selection policy.

Algorithm 4 LinUCB

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  Set α ▷ Equation (4.5)
  for $a \in \mathcal{A}$ **do**
    $ucb \leftarrow M_a^r(\boldsymbol{x}) + \sigma_a$
    $lcb \leftarrow M_a^c(\boldsymbol{x}) - \sigma_a$
  $a^* \leftarrow \arg\max_a \frac{ucb}{lcb}$
  return a^*

In the following, we discuss how to compute the parameter σ_a.

4.2.1.1 Confidence Interval Computation

We can estimate the confidence term σ_a by using three different methods, each based on a different theoretical foundation:

1. **LinUCB [11]** In the original LinUCB algorithm, the authors derive uncertainty from properties of the underlying linear model as:

$$\sigma_a = \alpha\sqrt{x_t^\top A_a^{-1} x_t} \tag{4.4}$$

   During each update step, the learner updates A_a of the selected arm by adding the outer product of the current context vector. Consequently, as the learner selects an arm more frequently, the entries of A_a increase, which in turn results in smaller entries of A_a^{-1} due to the properties of matrix inversion. This reduces σ_a, resulting in tighter confidence intervals as the algorithm gathers more data. LinUCB further controls the exploration-exploitation trade-off by the scaling factor α, defined as:

$$\alpha = 1 + \sqrt{\frac{\ln 2/\delta}{2}} \tag{4.5}$$

where $\delta > 0$ determines the probability $p = 1 - \delta$ with which the confidence bound holds. We chose $\delta = \frac{1}{t}$, allowing the confidence intervals to become tighter as more rounds are played.

2. **LinUCB Tor [8]** We derive an alternative approach from the classical UCB algorithm by Lattimore et al. [8]. They estimate uncertainty based on how often the algorithm has selected an arm relative to the total number of rounds.

$$\sigma_a = \sqrt{\frac{2\ln(1/\delta)}{\mathbb{I}_{\{a=a^*\}}}} \tag{4.6}$$

Instead of utilizing the linear models parameters, this method directly adjusts the confidence interval based on sample counts. As the learner selects arm a more frequently, its associated uncertainty decreases.

Lattimore et al. recommend choosing δ within the range $\frac{1}{t^2} < \delta < \frac{1}{t}$, which we approximate by choosing $\delta = \frac{1}{t\ln(t)}$

3. **m-LinUCB Xia [16]** We use the confidence approximation from the m-UCB algorithm by Xia et al. and incorporate in into our BCMAB [16]:

$$\sigma_a = \alpha\sqrt{\frac{\ln t}{\mathbb{I}_{\{a=a^*\}}}} \tag{4.7}$$

This formulation closely resembles Equation (4.6) with $\delta = \frac{1}{t}$, but introduces a scaling factor α to further adjust the exploration behavior manually.

4.2.2 The c-LinUCB Bandit

The c-LinUCB algorithm is based on LinUCB but incorporates elements of c-UCB by [16]. Instead of computing a lower confidence bound (LCB) for cost estimates, c-LinUCB directly utilizes the predicted mean cost without incorporating uncertainty estimates of the prediction. We show the arm selection policy in Algorithm 5.

Algorithm 5 c-LinUCB

```
1: function π(N, x, M, A, t)
2:     for a ∈ A do
3:         ucb ← M_a^r(x) + σ_a
4:         q ← M_a^c(x)                ▷ Uses predicted cost instead of LCB
5:     a* ← arg max_a ucb/q
6:     return a*
```

This modification is particularly beneficial in settings where cost uncertainty is low. Computing an LCB in such cases leads to an underestimation of the true mean, resulting in overly pessimistic decisions [7]. By directly using the predicted cost mean, c-LinUCB mitigates this issue. However, it introduces the risk of overconfidence in early rounds, as uncertainty is no longer explicitly accounted for [16]. The original c-UCB algorithm computed its confidence bound σ_a using Equation (4.7), but alternative formulations based on Equations (4.4) and (4.6) are also applicable.

4.2.3 Lin ω-UCB

The Lin-ω-UCB algorithm extends ω-UCB proposed by Heyden et al. [7]. This algorithm addresses a critical issue that arises if cost values are close to the boundaries of the predefined interval $[m, M]$. In such cases, confidence bounds according to the standard UCB or m-UCB formulation become overly optimistic or pessimistic, leading to suboptimal arm selection. To mitigate this problem, Lin-ω-UCB shifts both the upper and lower confidence bounds (UCB and LCB) towards the center of the interval. In the original ω-UCB formulation, confidence bounds are computed based on the sample mean and variance. In a contextual bandit setting with a linear model, the learner can compute the UCB and LCB by incorporating the predicted value of the model and its prediction variance. The variance term is based on the linear model and detailed in equation (4.8).

$$\sigma^2 = x_t^\top A_a^{-1} x_t \tag{4.8}$$

This follows the uncertainty computation from Li et al. [11]. With this, we formalize the arm selection policy in Algorithm 6.

Algorithm 6 Lin ω-UCB

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  $\rho \leftarrow \frac{1}{4}$ ▷ Scaling parameter for confidence bound shift
  for $a \in \mathcal{A}$ **do**
    $\mu_r \leftarrow M_a^r(\boldsymbol{x})$, $\mu_c \leftarrow M_a^c(\boldsymbol{x})$ ▷ Predicted reward and cost
    if $\mu_{r,c} \notin \{M, m\}$ **then**
      $\eta \leftarrow \frac{x_t^\top A_a^{-1} x_t}{(M-\mu_{r,c})(\mu_{r,c}-m)}$
    else
      $\eta \leftarrow 1$ ▷ Handles boundary cases
    $z \leftarrow \sqrt{2\rho \ln(t+2)}$ ▷ Exploration bonus term
    $A \leftarrow \mathbb{I}_{\{a=a^*\}} + z^2\eta$
    $B \leftarrow 2\mathbb{I}_{\{a=a^*\}}\mu_{r,c} + z^2\eta(M+m)$
    $C \leftarrow \mathbb{I}_{\{a=a^*\}}\mu_{r,c}^2 + z^2\eta Mm$
    $ci_{r,c} \leftarrow \sqrt{\frac{B^2}{4A^2} - \frac{C}{A}}$
    $ucb \leftarrow \frac{B^r}{2A^r} + ci_r$
    $lcb \leftarrow \frac{B^c}{2A^c} - ci_c$
  $a^* \leftarrow \arg\max_a \frac{ucb}{lcb}$
  return a^*

4.2.4 Budget Constrained (i) GP UCB

The GP UCB algorithm by Srinivas et al. is a contextual bandit that utilizes a Gaussian Process (GP) to estimate the reward function based on contextual information and corresponding costs and rewards [13]. The GP UCB bandit computes an upper confidence bound (UCB) for the expected reward of each arm by incorporating the standard deviation of the GP prediction, scaled by a confidence parameter β_t. We extend the original GP UCB to the budgeted setting. Algorithm 7 shows the resulting arm selection policy.

Algorithm 7 GP UCB

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  for $a \in \mathcal{A}$ **do**
    $\mu_r, \sigma_r \leftarrow M_a^r(\boldsymbol{x})$, $\mu_c, \sigma_c \leftarrow M_a^c(\boldsymbol{x})$
    $ucb \leftarrow \mu_r + \beta_t \cdot \sigma_r$ ▷ Upper Confidence Bound for reward
    $lcb \leftarrow \mu_c - \beta_t \cdot \sigma_c$ ▷ Lower Confidence Bound for cost
  $a^* \leftarrow \arg\max_a \frac{ucb}{lcb}$
  return a^*

We found two different approaches to compute the parameter β_t:

1. **GP UCB [13]** In its original formulation, β_t is a function of the number of rounds t and the size of the decision space $|D|$:

$$\beta_t = \sqrt{2 \ln \left(|D| \cdot t^2 \cdot \pi^2 \cdot \frac{1}{6\delta} \right)}$$

   The parameter δ is a confidence parameter and the corresponding confidence bounds are computed as

$$cb = \mu(\boldsymbol{x}) \mp \sqrt{\beta_t} \cdot \sigma$$

   where $\mu(\boldsymbol{x})$ is the GP mean prediction and σ the corresponding standard deviation.

   Since our setting differs from the original formulation—where every context is a separate decision variable—we approximate $|D|$ as the number of context vectors observed within the budget B. We estimate this number by using the expected cost per round:

$$|D| = \mathbb{E}[c] \cdot B$$

   For example, with uniformly distributed costs in $[0, 1]$ and $B = 1000$, the expected costs are $\mathbb{E}[c] = 0.5$, leading to $|D| = \frac{1000}{0.5} = 2000$. This is an approximation of the total number of distinct contexts the learner observes.

2. **Improved GP UCB (i-GP UCB) [3]** Chowdhury et al. extended the GP UCB approach by incorporating the MIG γ, defined in equation (4.1). Using this, we update the confidence parameter β_t as in equation (4.9).

$$\beta_t = B + R \cdot \sqrt{2 \cdot (\gamma_{t-1}^{r,c} + 1 + \pi^2 \ln(1/\delta))} \tag{4.9}$$

In (4.9) the parameter

- B is an upper bound on the RKHS norm of the approximated function.
- R is the sub-Gaussianity parameter, representing the noise level.

Notably, as $\gamma_t \to 0$, the confidence bound converges to B, meaning exploration diminishes as uncertainty decreases.

4.2.5 GP ω-UCB

The GP-ω-UCB algorithm extends the GP UCB algorithm by integrating the principles of ω-UCB. Similar to Lin-ω-UCB, this algorithm leverages the predictive mean and uncertainty estimates of the utilized model to construct confidence intervals for decision-making. In the case of a Gaussian Process, we utilize both the predicted mean and the standard deviation of the GP's prediction to define confidence bounds. Algorithm 8 details the full arm selection policy.

Algorithm 8 GP ω-UCB

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  $\rho \leftarrow \frac{1}{4}$ ▷ Scaling factor for uncertainty adjustment
  for $a \in \mathcal{A}$ **do**
    $\mu_r, \sigma_r \leftarrow M_a^r(\boldsymbol{x}), \mu_c, \sigma_c \leftarrow M_a^c(\boldsymbol{x})$
    if $\mu_{r,c} \notin \{M, m\}$ **then**
      $\eta \leftarrow \frac{\sigma_{r,c}^2}{(M-\mu_{r,c})(\mu_{r,c}-m)}$
    else
      $\eta \leftarrow 1$ ▷ Prevents division by zero
    $z \leftarrow \sqrt{2\rho \ln(t+2)}$ ▷ Exploration bonus term
    $A \leftarrow \mathbb{I}_{\{a=a^*\}} + z^2\eta$
    $B \leftarrow 2\mathbb{I}_{\{a=a^*\}}\mu_{r,c} + z^2\eta(M+m)$
    $C \leftarrow \mathbb{I}_{\{a=a^*\}}\mu_{r,c}^2 + z^2\eta Mm$
    $ci_{r,c} \leftarrow \sqrt{\frac{B^2}{4A^2} - \frac{C}{A}}$
    $ucb \leftarrow \frac{B^r}{2A^r} + ci_r$
    $lcb \leftarrow \frac{B^c}{2A^c} - ci_c$
  $a^* \leftarrow \arg\max_a \frac{ucb}{lcb}$
  return a^*

4.2.5.1 Neural ω-UCB: Extending Lin ω-UCB with Neural Networks

Additionally, we extend this algorithm with neural networks, referred to as Neural ω-UCB in the experiments. Since neural networks do not provide natural uncertainty estimates, we simplify the model by setting the scaling factor $\eta = 1$.

This simplification is justified because the predicted means lie within the interval $[0, 1]$, and the reward and cost functions follow a Bernoulli distribution. Given that the variance of a Bernoulli variable is

$$\sigma^2 = \mu(1 - \mu),$$

we can compute η as

$$\eta = \frac{\sigma^2}{(M - \mu)(\mu - m)} = \frac{\mu(1 - \mu)}{(1 - \mu)\mu} = 1.$$

While this approximation may not be optimal, it ensures consistency across all ω-UCB-based algorithms, allowing for a fair comparison of their asymptotic behavior across different models.

4.2.6 Budgeted Confidence Ball

The Budgeted Confidence Ball (BCB) algorithm, introduced by Xia et al. [16], is a contextual bandit algorithm designed to handle budget constraints while optimizing arm selection. Unlike standard UCB or Thompson Sampling algorithms, which focus primarily on maximizing cumulative reward, BCB explicitly incorporates a budget-aware confidence set into its decision-making process. BCB extends traditional confidence-bound-based approaches by maintaining a confidence ball around the estimated reward-to-cost ratio of each arm. This confidence ball is dynamically adjusted based on the observed cost and reward, ensuring that the budget is efficiently allocated.

The key components of BCB include:

- A confidence set for each arm, which considers both reward and cost uncertainty.
- A decision rule that selects arms based on the upper bound of the confidence interval of the reward-to-cost ratio.
- A budget update mechanism, ensuring that arms with low expected costs are favored when exploration is necessary.

For a detailed description of the algorithm, including its theoretical guarantees and implementation details, we refer to its publication [16].

4.3 Thompson Sampling Bandits

In Section 2.4.3, we introduced Thompson Sampling as a Bayesian approach to balancing exploration and exploitation in MAB problems. This section extends this foundation by presenting advanced variants that incorporate contextual information and adhere to budget constraints for cost-effective decision-making.

4.3.1 Budget Aware Linear TS

Budget-Aware Linear Thompson Sampling (LinTS) extends the bayesian interpretation of ridge regression in LinUCB, as noted by Li et al. [11]. In their formulation, the weight vector of the linear model follows a Gaussian posterior distribution. The mean of this distribution is the estimated weight vector, $\tilde{\theta}$ and the covariance defined by the inverse data matrix A^{-1} [11]:

$$\hat{\theta} \sim \mathcal{N}(\tilde{\theta}, A^{-1}).$$

This interpretation naturally leads to a Thompson Sampling strategy, where the learner samples a new weight vector $\hat{\theta}$ from this distribution in each round. The variance of this sampling process implicitly induces exploration. This is because higher uncertainty of the model results in higher variance in the sampled parameters. We show the procedure of the Contextual TS policy in Algorithm 9.

Algorithm 9 Contextual TS

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  Initialize v
  for $a \in \mathcal{A}$ **do**
    $\hat{\theta}^{r,c} \sim \mathcal{N}(\tilde{\theta}^{r,c}, v^2 A_a^{-1})$
    $\mu_r \leftarrow \boldsymbol{x}^\top \hat{\theta}^r$, $\mu_c \leftarrow \boldsymbol{x}^\top \hat{\theta}^c$
  $a^* \leftarrow \arg\max_a \frac{\mu_r}{\mu_c}$
  return a^*

Agrawal et al. proposed a similar approach. They introduced an additional scaling parameter v and called it exploration variance [1]. By tuning v, we can control the balance between exploration (high variance) and exploitation (low variance) based on prior knowledge.

4.3.2 Contextual c-TS

The Contextual c-TS algorithm is a modification of Contextual TS (Algorithm 9). It maintains the same sampling procedure for the reward estimation while using a deterministic prediction for the cost. We motivate this adjustment by scenarios where cost estimates exhibit low variance, making sampling unnecessary and potentially misleading. Instead, the learner makes a decision directly based on the predicted cost value. This reduces

variance in decision-making while maintaining adaptive exploration through the sampling of rewards. Algorithm 10 details the procedure.

Algorithm 10 Contextual c-TS

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  Initialize v
  for $a \in \mathcal{A}$ **do**
    $\hat{\theta}^r \sim \mathcal{N}(\tilde{\theta}^r, v^2 A_a^{-1})$
    $\mu_r \leftarrow \boldsymbol{x}^\top \hat{\theta}^r$
    $\mu_c \leftarrow \boldsymbol{x}^\top \tilde{\theta}^c$ ▷ No sampling for cost
  $a^* \leftarrow \arg\max_a \frac{\mu_r}{\mu_c}$
  return a^*

4.3.3 β-TS

The β-TS algorithm is based on Budgeted Thompson Sampling (BTS) proposed by Xia et al. [17]. In BTS, the reward and cost distributions for each arm are modeled using Beta distributions $\mathcal{B}(a, b)$. The parameters a, b are updated in each round based on observed rewards and costs:

$$a \leftarrow a + r_a, \quad b \leftarrow b + (1 - r_a).$$

In its original formulation, the learner maintained a separate Beta distribution for each arm. However, extending this approach directly to a contextual setting would require maintaining a separate Beta distribution for every possible context $\boldsymbol{x}$. Since the context space in our setting is continuous, this is computationally infeasible and would not make sense since it is very unlikely to observe the same context multiple times. To approximate the effect of past observations without explicitly storing a Beta distribution for every context, we introduce a weighting mechanism. In each round, if the learner has chosen an arm n times, we assume it has observed the same context as in the current round in all previous rounds as well. This assumption allows us to approximate the contextual problem as a standard MAB problem, enabling us to apply the same Beta-sampling principles from the original BTS algorithm. We show the complete arm selection policy in Algorithm 11.

Algorithm 11 β-TS

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  for $a \in \mathcal{A}$ **do**
    $\mu_r \leftarrow M_a^r(\boldsymbol{x})$, $\mu_c \leftarrow M_a^c(\boldsymbol{x})$
    $\hat{\mu}_r \sim \mathcal{B}\left(\frac{\mathbb{I}_{\{a=a^*\}} \cdot \mu_r}{x_t^\top A_a^{-1} x_t}, \frac{\mathbb{I}_{\{a=a^*\}} \cdot (1-\mu_r)}{x_t^\top A_a^{-1} x_t}\right)$
    $\hat{\mu}_c \sim \mathcal{B}\left(\frac{\mathbb{I}_{\{a=a^*\}} \cdot \mu_c}{x_t^\top A_a^{-1} x_t}, \frac{\mathbb{I}_{\{a=a^*\}} \cdot (1-\mu_c)}{x_t^\top A_a^{-1} x_t}\right)$
  $a^* \leftarrow \arg\max_a \frac{\hat{\mu}_r}{\hat{\mu}_c}$
  return a^*

4.3.3.1 Variance Contraction for Faster Convergence

To accelerate convergence in Linear β-TS, we utilize the variance of the linear model's prediction, given by:

$$\sigma^2 = x_t^\top A_a^{-1} x_t.$$

We motivate this approach is motivated by the variance properties of the Beta distribution. Given a Beta distribution $\mathcal{B}(a, b)$, its variance is defined as:

$$\sigma^2 = \frac{ab}{(a+b)^2(a+b+1)}. \tag{4.10}$$

From Equation (4.10), we observe that as a and b increases, the variance decreases. This is called variance contraction and plays a key role in Algorithm 11. As the learner chooses an arm more frequently, more data is collected. This leads to larger values of a and b. Consequently, the Beta distribution becomes increasingly concentrated, implicating lower variance. This is called variance contraction and illustrated in Figure 4.2.

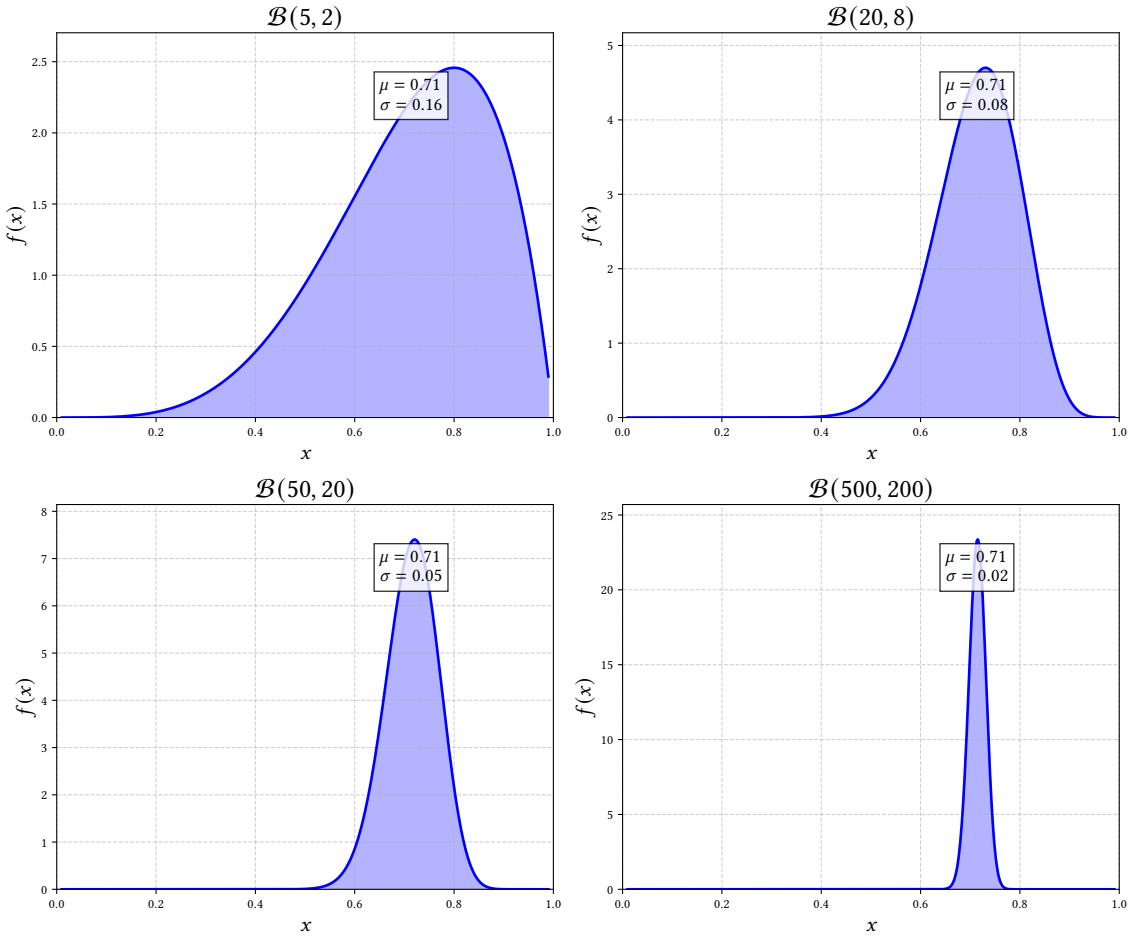

Figure 4.2: Beta distributions with increasing parameter values, demonstrating variance contraction.

Scaling by $\frac{1}{\sigma^2}$ In Linear β-TS, the parameters a and b are scaled by the inverse variance:

$$a' = \frac{a}{\sigma^2}, \quad b' = \frac{b}{\sigma^2}.$$

This scaling mechanism ensures that updates are more aggressive in early rounds when the variance σ^2 is high, and become more conservative in later rounds as variance decreases below 1. This adaptive learning rate should accelerate convergence while maintaining robust uncertainty estimation.

4.3.4 GP TS

The Gaussian Process Thompson Sampling (GP TS) algorithm by Chowdhury et al. [3] follows a smiliar fundamental principle as Linear Thompson Sampling (LinTS): sampling rewards and costs from a Gaussian distribution. However, instead of relying on a linear model, GP TS employs a Gaussian Process (GP) to estimate the reward and cost functions based on observations. The scaling parameter β_t is a key aspect of GP TS and also present in i-GP UCB [3]. This parameter scales the GP's prediction variance when sampling from the normal distribution. By dynamically adjusting the level of exploration, β_t ensures that sampled values reflect both model uncertainty and observed data. Algorithm 12 shows the complete selection policy of GP TS.

Algorithm 12 GP TS

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  for $a \in \mathcal{A}$ **do**
    $\mu_r, \sigma_r \leftarrow M_a^r(\boldsymbol{x})$, $\mu_c, \sigma_c \leftarrow M_a^c(\boldsymbol{x})$
    $\hat{\mu}_r \sim \mathcal{N}\left(\mu_r, \beta_t \sigma_r^2\right)$
    $\hat{\mu}_c \sim \mathcal{N}\left(\mu_c, \beta_t \sigma_c^2\right)$
  $a^* \leftarrow \arg\max_a \frac{\hat{\mu}_r}{\hat{\mu}_c}$
  return a^*

Computing β_t The choice of β_t impacts the trade-off between exploration and exploitation. We consider two approaches:

1. **GP TS Chowdhury [3]** We can compute β_t similar to i-GP UCB, which is shown in equation (4.9). This formulation incorporates the mutual information gain γ, making exploration adaptive by adjusting confidence bounds based on the informativeness of new observations.

2. **GP TS Srinivas [13]** Since Chowdhury et al. used the same term for β_t in the GP TS and i-GP UCB we incorporated the β-term from GP UCB into GP TS. This algorithm incorporates both the decision space size $|D|$ and the number of rounds t.

4.3.5 GP β-TS

The GP β-TS algorithm extends the Linear β-TS approach (see Algorithm 11) by replacing the linear model with a Gaussian Process (GP). Similar to the original β-TS formulation, it samples rewards from a Beta distribution, $\mathcal{B}(a, b)$, incorporating uncertainty into the decision-making process. However, the key difference lies in how the parameters a and b are computed.

Key Modifications from Linear β-TS An improvement in GP-β-TS is the introduction of a scaling factor s, which ensures that the sampled values correctly reflect the uncertainty estimates derived from the Gaussian Process. This is done by utilizing:

- The MIG γ (4.1)
- The number of times the learner chose an arm

Algorithm 13 describes the full selection policy for GP β-TS variants.

Algorithm 13 GP β-TS

function $\pi(N, \boldsymbol{x}, \mathcal{M}, \mathcal{A}, t)$
  for $a \in \mathcal{A}$ **do**
    $\mu_r, \sigma_r \leftarrow M_a^r(\boldsymbol{x})$, $\mu_c, \sigma_c \leftarrow M_a^c(\boldsymbol{x})$
    $\gamma_{t-1}^{r,c} \leftarrow \frac{1}{2} \log\det\left(1 + \frac{K_{r,c}}{\eta^2}\right)$
    $s_{r,c} \leftarrow \max(1.0, \min(\mathbb{I}_{\{a=a^*\}}, \frac{1}{\gamma_{t-1}^{r,c}}))$
    $\hat{\mu}_{r,c} \sim \mathcal{B}\left(s_{r,c} \cdot \mathbb{I}_{\{a=a^*\}} \cdot \mu_{r,c}; s_{r,c} \cdot \mathbb{I}_{\{a=a^*\}} \cdot (1 - \mu_{r,c})\right)$
  $a^* \leftarrow \arg\max_a \frac{\hat{\mu}_r}{\hat{\mu}_c}$
  return a^*

Effect of the Scaling Factor s We use the scaling factor s to shape the behavior of GP-β-TS. It incorporates the MIG γ while accounting for the number of times the learner chose an arm. This ensures that the sampled values variance accurately reflects the uncertainty of the Gaussian Process predictions. Ultimately, this mechanism facilitates faster variance contraction and enhances convergence over time.

5 Experimental Setup

This chapter describes the experimental setup used to evaluate the proposed algorithms under synthetic and real-world conditions. The goal is to show their ability to incorporate context based costs and rewards and compare their performance based on regret minimization and adaptivity to different cost-reward relationships.

5.1 Synthetic Data Experiment

To evaluate the proposed algorithms under controlled conditions, we use synthetic data. In each round t, we draw the context vector $\boldsymbol{x_t}$ from a uniform distribution over the unit hypercube in d dimensions and consider $d = 5$ for this study:

$$\boldsymbol{x} \sim \mathcal{U}([0, 1)^5).$$

5.1.1 Cost and Reward Generation

We use costs and rewards sampled from three different environments to test the bandits.

1. **Bernoulli Sampling with Uniform Weights**: Initially, we sample the true weight vector of reward and cost functions uniformly from the interval $[0, 1]$. If the sum of a weight vector exceeds 1, we normalize it to ensure bounded influence. The learner interprets the predicted values for costs and rewards as probabilities in a Bernoulli trial. The outcome, i.e. the observed values p, take the value 1 with probability p and 0 with probability $1 - p$.

2. **Bernoulli Sampling with Beta-Distributed Weights**: Instead of using a uniform distribution, we draw the weight vector of reward and cost functions from a Beta distribution:

   $$\mathcal{B}(0.5, 0.5)$$

   This choice introduces a polarization effect, favoring values close to 0 or 1, as illustrated in Figure 5.1. Our motivation for this follows the work of Heyden et al. which showed that some upper confidence bound bandits have sever issues in this setting with fixed reward and cost means drawn from similar distributions [7].

   Similar to the uniform case, weight vectors exceeding a sum of 1 undergo normalization to maintain consistency. The learner can interpret the resulting values either as continuous values or as probabilities in a Bernoulli trial.

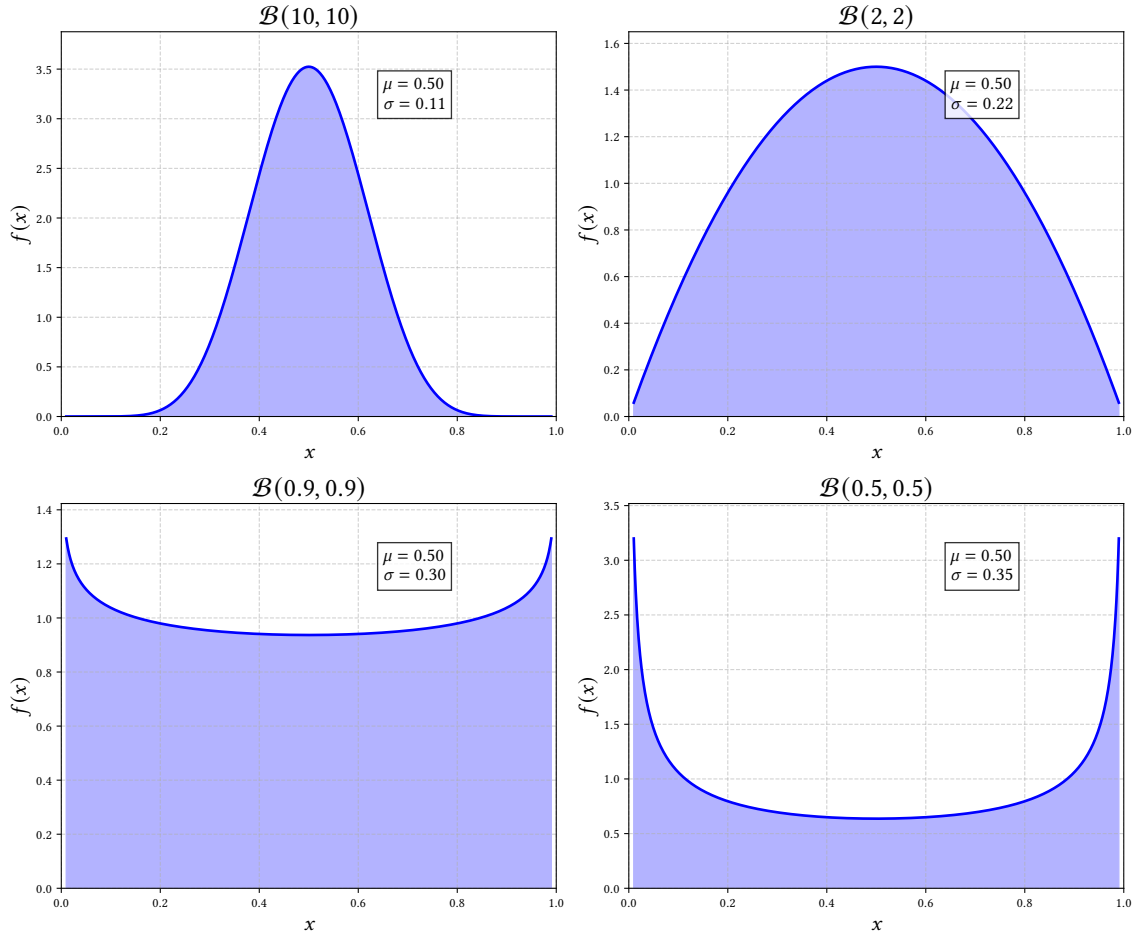

Figure 5.1: Comparison of Beta distributions $\mathcal{B}(\alpha, \beta)$ with varying parameters. The Beta distribution models polarized probabilities (favoring values near 0 or 1), while the uniform distribution represents neutral uncertainty and is equivalent to $\mathcal{B}(1, 1)$.

3. **Continuous Interpretation of 1 & 2**: Instead of interpreting p as the probability in a Bernoulli trial, the learner interprets it as a continuous value for either costs or rewards. This allows for a more gradual and smooth representation of the cost-reward relationship with less uncertainty.

5.1.2 Non-Linear Reward Function

To introduce non-linearity into the cost and reward structure, we employ the following function:

$$r_a(\boldsymbol{x}) = 0.5 + 0.4 \cdot \sin\left(5f(\boldsymbol{x})\right) - 0.1 \cdot \left(f(\boldsymbol{x})\right)^2$$

The parameter $\boldsymbol{x}$ represents the context vector and f is the function that represents the cost-reward-context relationship. We chose this function because it introduces a controlled level of nonlinearity while maintaining a degree of linear approximability. This ensures that linear bandit models do not completely fail, but still face limitations compared to nonlinear approaches. The function includes a sinusoidal term to introduce periodic fluctuations that linear models cannot fully capture, as well as a quadratic term to further increase nonlinearity. This design allows for a meaningful comparison between linear and nonlinear bandit models, highlighting the advantages of nonlinear models. We selected the function empirically by balancing learnability for linear models with sufficient complexity to justify the use of nonlinear techniques. Figure 5.2 shows the graph of $r_a(\boldsymbol{x})$.

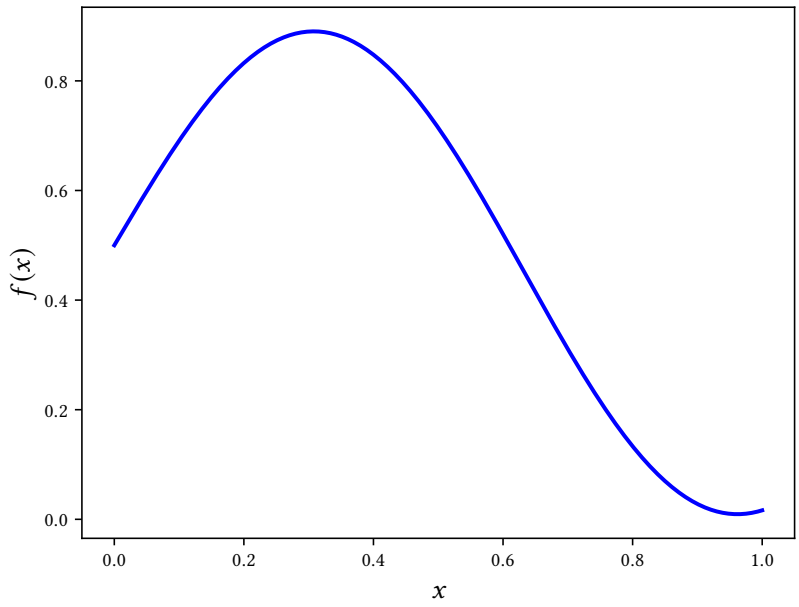

Figure 5.2: Visualization of the synthetic non-linear reward-cost-context relationship.

While the synthetic experiments provide controlled conditions, real-world datasets introduce additional complexity due to noise and imperfect feature representations. To study this, we integrate data from a real-world advertisement dataset.

5.2 Real-World Data Integration

To conduct our experiment with real world data, we estimate a distribution using data from the Facebook Ad Campaign Dataset [9]. The dataset contains three advertising campaigns (arms) and 761 user groups, each represented by a set of contextual features: `age`, `gender`, `interest1`, `interest2`, and `interest3`.

5.2.1 Context Distribution Estimation

We approximate the underlying context distribution by using a Gaussian Kernel Density Estimation (KDE) with a bandwidth of 0.15, selected to balance over- and under-smoothing based on preliminary experiments. We fit the KDE to the preprocessed dataset, following these transformations:

- **Age Binning**: Age values are discretized into 5-year intervals (e.g., 30–34 mapped to 32.0).
- **Feature Scaling**: Numerical features are normalized to the range $[0, 1]$ using Min-Max normalization.

To ensure a realistic representation of the dataset, we weight the Gaussian KDE by using the normalized values of the `impressions` column. The resulting density estimate is defined as

$$\hat{f}(\boldsymbol{x}) = \frac{1}{n} \sum_{i=1}^{n} w_i \cdot \mathcal{N}(\boldsymbol{x} | \boldsymbol{x}_i, \sigma^2),$$

where w_i represents the normalized impression weights, and the kernel bandwidth is $\sigma = 0.15$. We clipped sampled contexts to $[0, 1]^5$ to ensure validity. In figure 5.3 we present the resulting Kernel Density Estimation:

5.2.2 Reward and Cost Computation

We define the reward as the click-through rate (CTR), calculated as

$$\text{CTR} = \frac{\text{clicks}}{\text{impressions}}.$$

Similar, we set the associated costs to the Cost Per Click (CPC), which we computed like

$$\text{CPC} = \frac{\text{spent}}{\text{clicks}}.$$

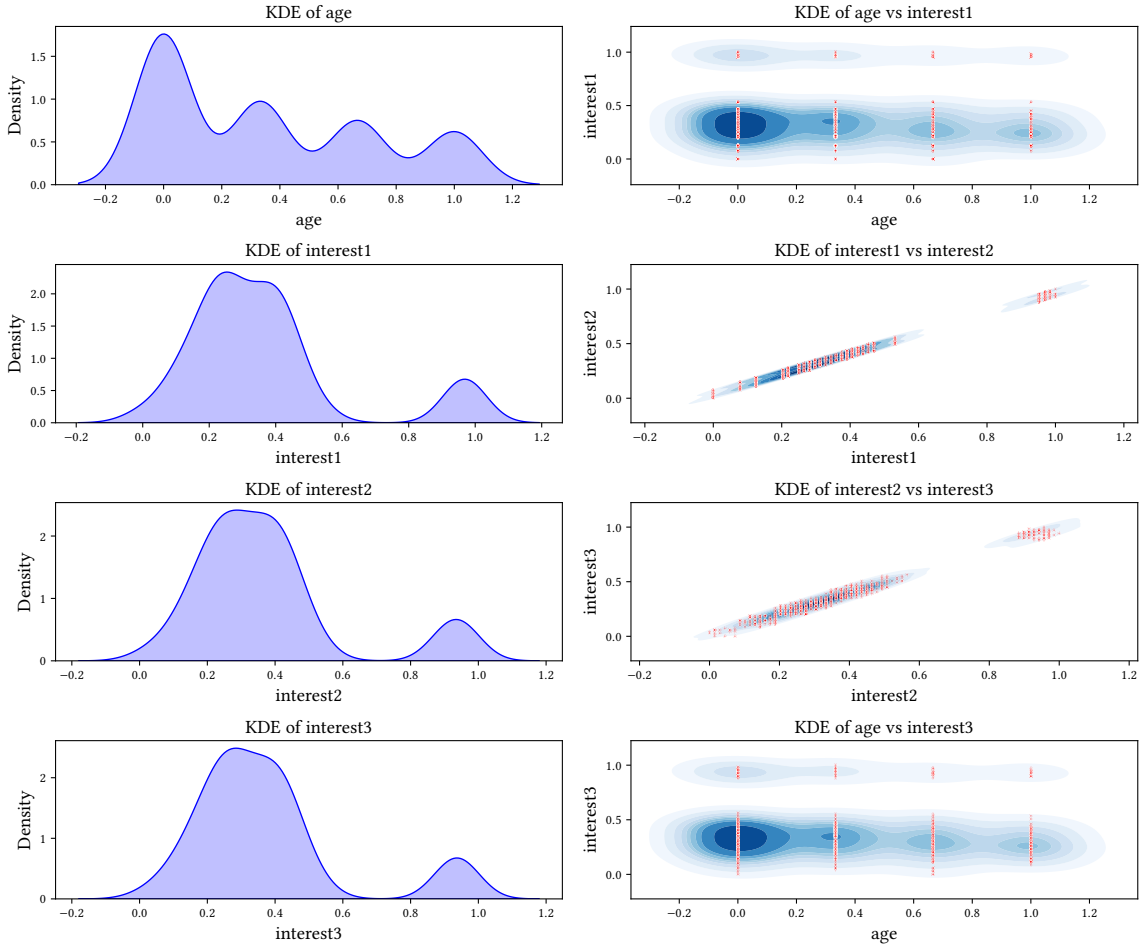

Figure 5.3: Kernel Density Estimation of the Kaggle Dataset [9]. The red points represent the original dataset, while the blue shaded regions illustrate the density estimate and sampled values.

Since not every user group appears across all ad campaigns, predictive models capture the relationship between context features and CTR or CPC. For each campaign, a Gaussian Process Regressor maps context vectors to expected CTRs and CPCs. The kernel function is composed of three terms:

$$k(x, x') = C(1.0) \cdot \text{RBF}(l) + \text{WhiteKernel}(\sigma^2)$$

where:

- $C(1.0, (10^{-3}, 10^3))$ represents a constant kernel with an initial value of 1.0 and bounds between 10^{-3} and 10^3.
- **RBF**$(l, (10^{-2}, 10^2))$ is the Radial Basis Function (RBF) kernel, where the length scale l is initialized to 1.0 and constrained between 10^{-2} and 10^2.
- **WhiteKernel**$(\sigma^2, (10^{-10}, 10^1))$ models independent Gaussian noise with an initial noise level of 10^{-5} and bounds ranging from 10^{-10} to 10^1.

We train the Gaussian Process using the Scikit-Learn implementation with the following configuration:

- Kernel: $k(x, x')$ as defined above.
- Optimizer restarts: $n_{\text{restarts}} = 10$
- Fixed random seed 42 for reproducibility.

This setup allows the GP to capture both smooth variations in the data (via the RBF kernel) and small independent fluctuations (via the WhiteKernel), while maintaining an overall scaling factor through the Constant Kernel.

5.3 Evaluation of Bandit Algorithms

We conducted the evaluation of bandit algorithms using the parameter configurations outlined in Tables 6.3, 6.2, and 6.1. If an algorithm is instance-dependent, we chose its hyperparameters based on the values provided in the corresponding publications. Otherwise, we marked it as *independent* in the *Baseline* column. In cases where no explicit values were available, a conservative estimate followed from prior experience.

Each algorithm ran 50 times under a fixed budget of $B = 1000$. To ensure reproducibility, we set the random seed for each trial equal to the trial's index.

To allow for a fair comparison across trials with different numbers of rounds, we interpolated the results at 100 equidistant points. The final performance curves result from computing the mean at each interpolation point, ensuring a smooth representation of the average performance. We display the cumulative regret as a line plot with the normalized budget on the x-axis and the cumulative regret on the y-axis. Error bars represent the 95% confidence intervals over 50 trials. Additionally, a violin plot illustrates the distribution of summed regret, including the 25%, 50%, and 75% percentiles to highlight key statistical properties. Unlike boxplots, which only summarize quartiles and outliers, violin plots

depict the full probability density. This makes them particularly useful for identifying features such as multimodality, skewness, or variations in distribution shape that a boxplot might obscure.

6 Results and Discussion

In this chapter, we present and discuss the results of our experiments. Each bandit algorithm underwent evaluation according to the experimental setup described in Chapter 5. We find it important to mention that these results do not represent a definitive ranking of the algorithms, as we did not perform a hyperparameter optimization. This comparison aims to analyze the behavior of different policies across various settings, with a focus on their convergence behavior, stability, and sensitivity.

6.1 Comparison of Greedy Policies

We evaluate the performance of Greedy bandit's introduced in Chapter 2.4.1. Table 6.1 summarizes the parameters used in the experiments.

Table 6.1: Parameters to evaluate Greedy policies

Algorithm	Parameter	Reference
Linear b-Greedy	$\alpha = 4$	[16]
Linear ϵ-Greedy	$\epsilon = 0.1$	[8]
Linear ϵ-First	$b_e = 0.1 \cdot B$	[14]
GP First	$b_e = 0.1 \cdot B$	[14]
Neural b-Greedy	$\alpha = 4$	[16]

6.1.1 Evaluation with Synthetic Data

We analyze the impact of different exploration parameters ϵ by evaluating Linear ϵ-First, GP First, Contextual b-Greedy, and Static b-Greedy.

6.1.1.1 Linear Reward and Cost Function

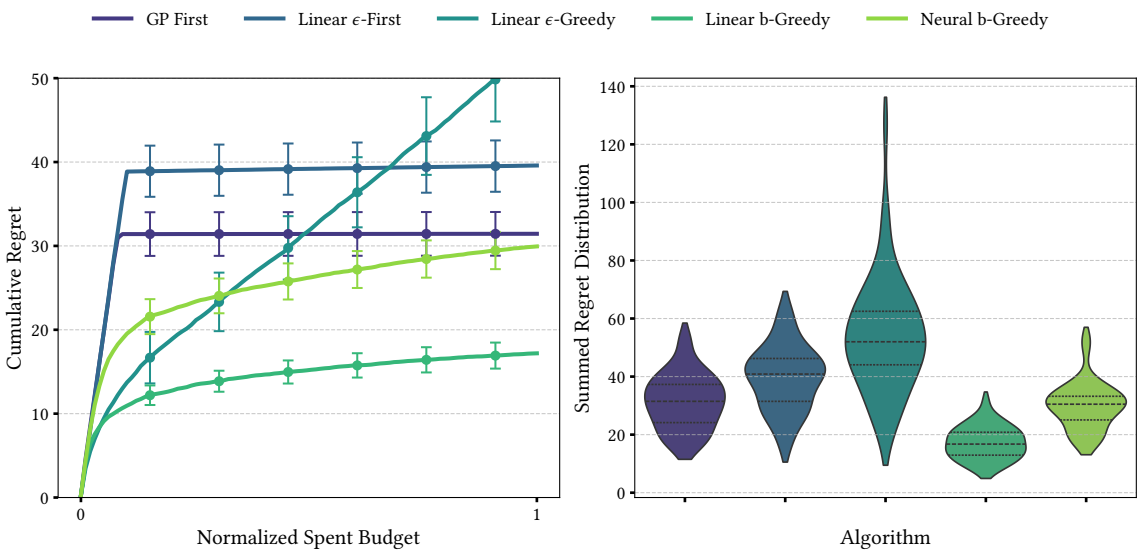

Figure 6.1: Cumulative regret of Greedy bandits with linear rewards and continuous costs.

First, we evaluate the algorithms with linear reward and cost functions and continuos outputs. Figure 6.1 displays the results.

Linear ϵ-Greedy shows an almost linear regret increase over time. This is due to Linear ϵ-Greedy's static exploration parameter. Although it initially outperforms Neural b-Greedy and GP First, its regret surpasses them in the long term.

GP First effectively models the reward-cost-context relationship, stabilizing its regret once the exploration phase concludes. Since it dynamically adjusts its exploration budget, it outperforms Linear ϵ-First, which uses a fixed exploration budget. This stability becomes evident in the violin plot, where GP First demonstrates lower variance compared to Linear ϵ-Greedy and Linear ϵ-First.

Linear b-Greedy performs best overall, achieving the lowest cumulative regret while maintaining narrow 95% confidence intervals. This suggests that its adaptive exploration strategy efficiently balances exploration and exploitation.

Neural b-Greedy is competitive but exhibits higher variance than Linear b-Greedy. Since the neural network requires more training data to generalize well, it initially struggles more than its linear counterpart. Over time, it converges with slightly less stability and higher regret than the linear model.

6.1.1.2 Non-Linear Reward and Cost function

Now, both the reward and cost function are non-linear.

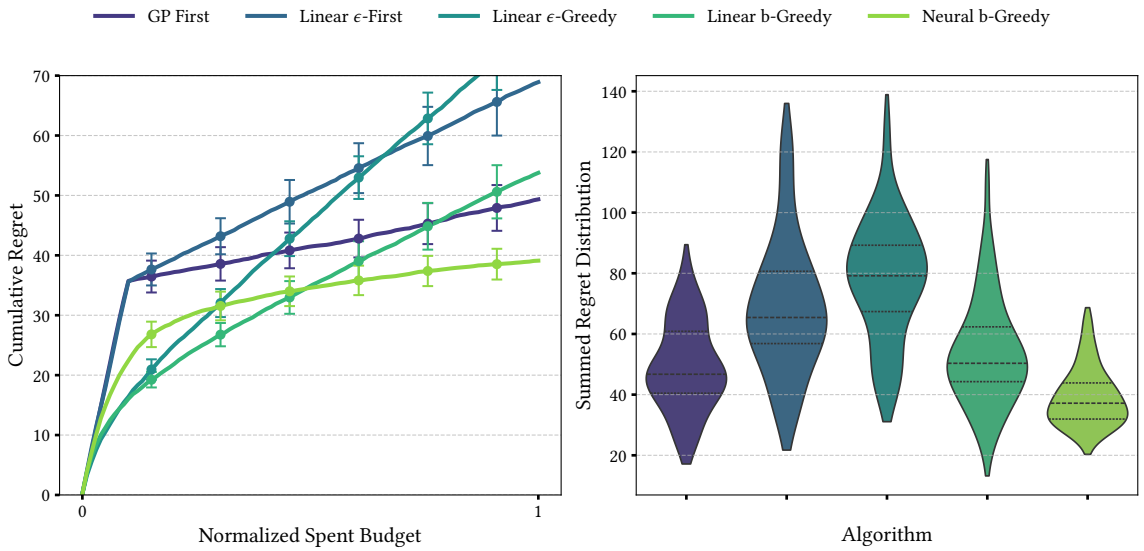

Figure 6.2: Cumulative regret of greedy bandits with non-linear, continuous rewards and costs.

The linear bandits struggle in this setting and cannot compete effectively. Compared to the results of the previous trial, the regret of GP First continues to grow linearly even after the exploration phase. This suggests that the algorithm reduces its exploration budget too quickly, relative to the available budget. In turn, this leads to an insufficient amount of data for the model to accurately learn the cost and reward function. Inconsistency of GP First's behavior could also explain this. This means in some trials GP First learns the function and the regret stabilizes, while in others it does not. This leads to an inclined regret curve since the results are averaged.

Neural b-Greedy exhibits a similar logarithmic regret curve as in the linear setting, albeit with a higher overall regret. This suggests that Neural b-Greedy adapts more effectively to the non-linear structure of the problem, likely due to its capacity to model complex function approximations.

6.1.1.3 Effect of Beta-Distributed Weights

As we can see from figure 6.3, Beta-distributed weights do not appear to have any noticeable effect on the greedy bandits. Greedy algorithms do not quantify uncertainty through the computation of rewards and costs as UCB or Thompson sampling bandits do, which explains this behavior.

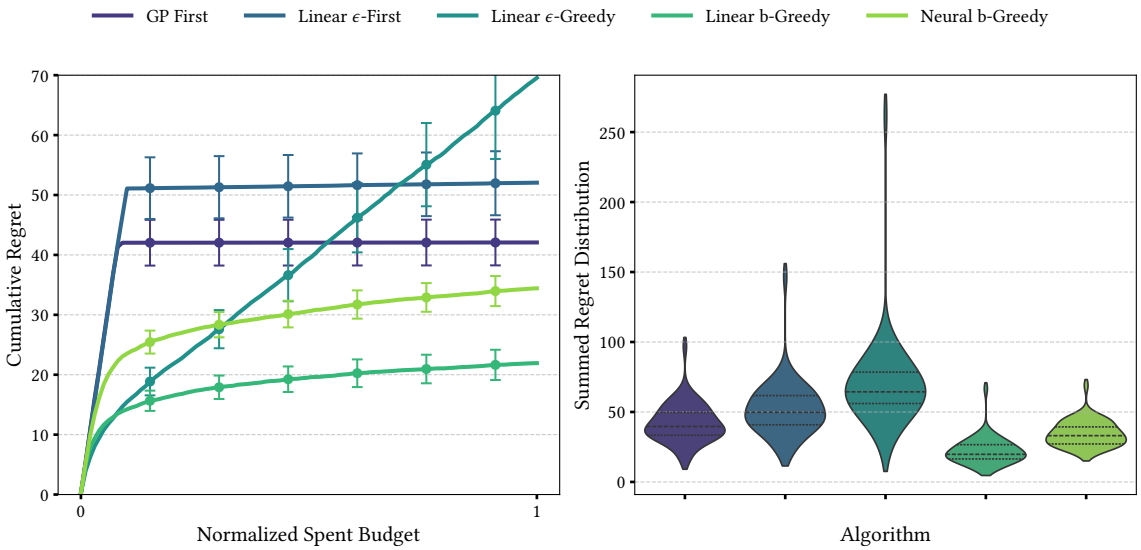

Figure 6.3: Cumulative regret of Greedy bandits with linear rewards and costs, $\mathcal{B}(0.5, 0.5)$ sampled weights.

6.1.1.4 Evaluation in a Bernoulli Setting

Next, we examine a scenario where the learner interprets the reward and cost functions output value as the probability of a Bernoulli-distributed random variable. Figure 6.4 shows the results of this trial.

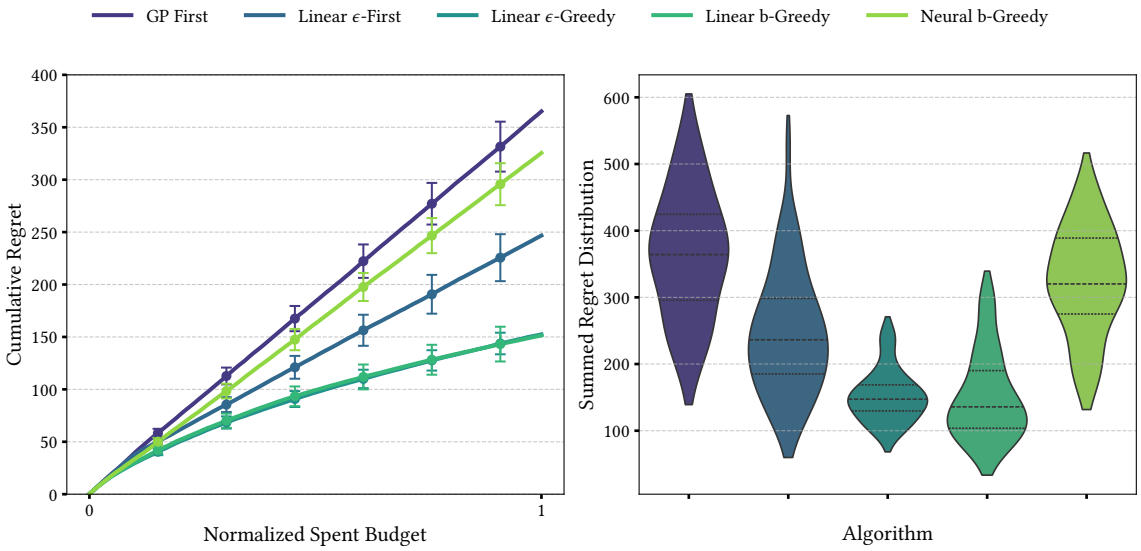

Figure 6.4: Cumulative regret of Greedy bandits with linear, Bernoulli rewards and costs.

The high regret of the Linear ϵ-First variants suggests that their exploration budgets are insufficient to adequately learn the cost and reward functions in this high-variance

setting. The GP First algorithm amplifies this effect, as the learner dynamically reduces the exploration budget, leading to linear regret in both cases.

The performance gap between Linear b-Greedy and Linear ϵ-Greedy in the line plot is significantly smaller than in the continuous setting. This suggests that the hyperparameter choice of b-Greedy is not optimal for this setting and highlights the big disadvantage of hyperparameter based algorithms. The violin plot further reflects this, since the summed regret distribution of b-Greedy is wider than the one of Linear ϵ-Greedy.

Overall, we see that the Linear model performed better than the GP, suggesting this is an issue of the model as well.

6.1.1.5 Combining Beta and Bernoulli Distributions

In figure 6.5 we look at the bandits with linear reward and cost functions and weights that follow a Beta distribution. Additionally, the learner interprets the output value as the probability of a Bernoulli-distributed random variable.

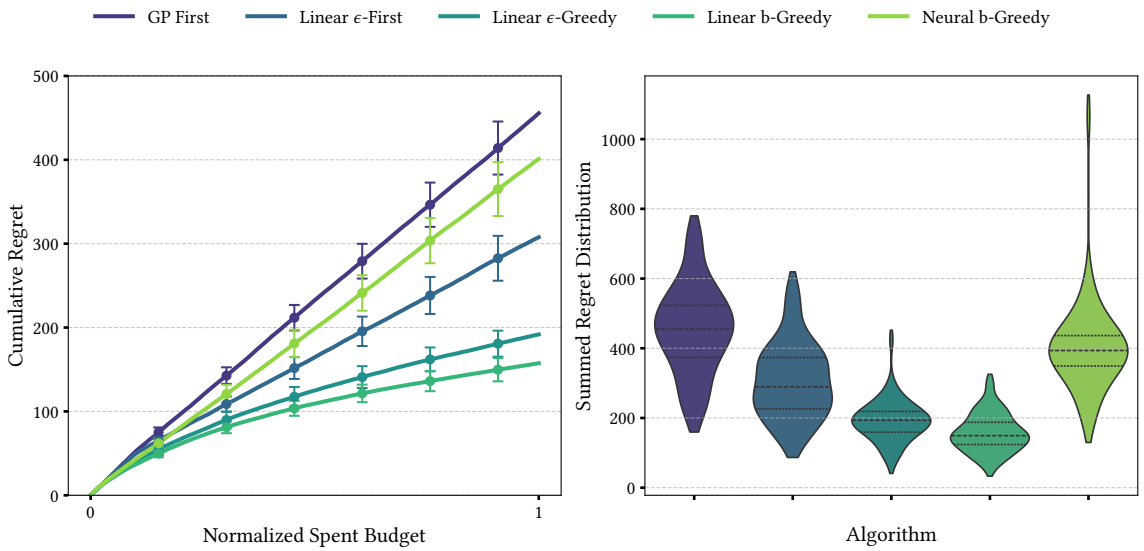

Figure 6.5: Cumulative regret of greedy bandits with linear, Bernoulli rewards & costs, $\mathcal{B}(0.5, 0.5)$ sampled weights.

Despite the increased variance, the Linear b-Greedy algorithm still achieves the lowest cumulative regret, demonstrating its robustness. The Bernoulli setting pronounces b-Greedy's advantage over Linear ϵ-Greedy less due to the inherent randomness in Bernoulli-distributed rewards and costs.

6.1.2 Evaluation with Facebook Advertisement Data

Finally, we evaluate the performance on Facebook advertisement campaign data.

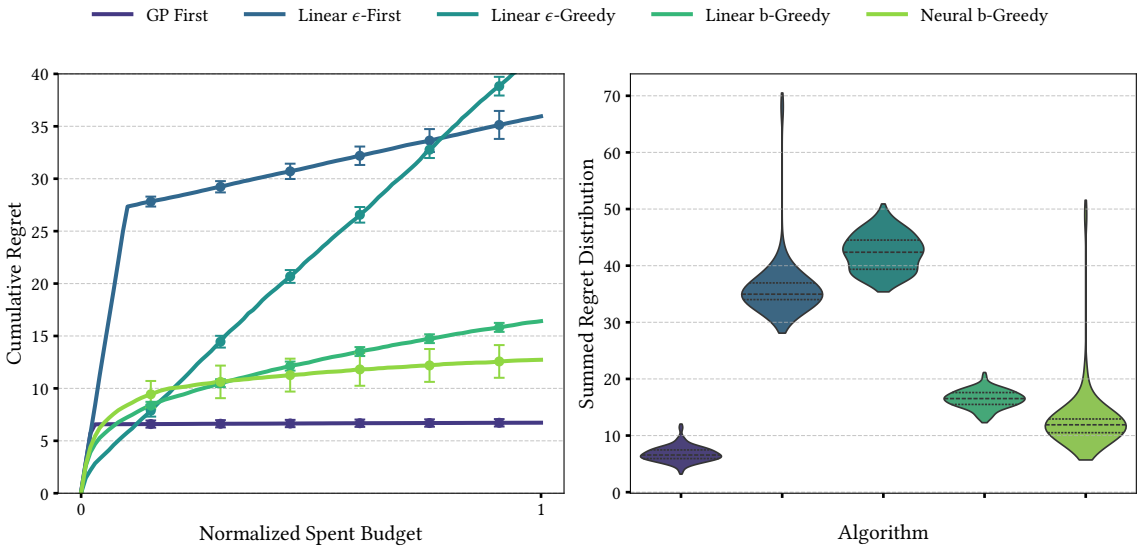

Figure 6.6: Cumulative regret of greedy bandits on Facebook advertisement data with continuous rewards and costs.

The results in Figure 6.6 show that the GP First algorithm performs well in this environment. In contrast, its linear counterpart fails to learn the underlying relationship, even with a potentially much larger exploration budget. As a result, regret continues to increase even after the end of its exploration phase. This highlights a significant drawback of static exploration strategies: they can lead to persistently high regret in environments where their assumptions do not hold.

Comparing the Linear b-Greedy and Neural b-Greedy algorithms, we observe that non-linear models generally achieve better performance in capturing cost and reward functions in this setting. The Neural b-Greedy algorithm exhibits superior asymptotic behavior, despite both algorithms using the same exploration hyperparameters. Given the low variance of GP First, we hypothesize that Gaussian Process-based models effectively handle this type of contextual bandit problem. Although one major drawback of these models is their high computational cost.

While the Linear b-Greedy bandit performs slightly worse than its neural counterpart, its variance in summed regret is lower. The violin plot and narrow 95% confidence intervals indicate this further, suggesting that the linear model leads to more stable results.

6.2 Comparison of Upper Confidence Bound Policies

In this section, we compare different UCB policies according to the setup we described in chapter 5. Table 6.2 summarizes the parameters used in the experiments.

Table 6.2: Parameters to evaluate UCB policies

Type	Parameter	Reference
Budget-CB	$\rho = 1$	[16]
	$\delta = 1/B$	
	$L = 5$	
	$S = 1$	
	$C = 1e - 3$	
LinUCB Tor	independent	-
LinUCB	$\alpha = 0.7$	-
c-LinUCB Tor	independent	-
c-LinUCB Xia	$\alpha = 0.25$	[16]
m-LinUCB Xia	$\alpha = 0.25$	[16]
Lin ω-UCB	$\rho = 0.25$	[7]
Neural ω-UCB	$\rho = 0.25$	[7]
GP UCB	$\delta = 0.1$	[13]
i-GP UCB	$\delta = 0.1$	[3]

6.2.1 Evaluation with Synthetic Data

We begin to compare the UCB algorithms with synthetically generated data to test them in a controlled environment.

6.2.1.1 Linear Reward and Cost Function

Figure 6.7 shows that c-LinUCB Xia, m-LinUCB Xia, GP ω-UCB and Lin ω-UCB achieve the lowest cumulative regret and exhibit strong asymptotic behavior. This indicates narrow confidence intervals, and low variance, as also evident from the violin plot. It suggests that these policies efficiently explore across multiple runs.

Neural ω-UCB, and c-LinUCB Tor algorithms follow closely, showing slightly higher cumulative regret but still outperform most other algorithms. The results suggest that the confidence intervals proposed by Xia et al. [16] result in more stable regret reduction compared to the one from Lattimore et al. However, both exhibit relatively low variance.

The budget-CB bandit demonstrates reasonable asymptotic behavior, though the aforementioned algorithms outperform it. We think this is due to the choice of the hyperparameter δ, which we set adaptively as $\frac{1}{t}$. This choice is potentially suboptimal in this setting, leading to higher long-term regret. Setting this hyperparameter with a fixed and tuned value could increase performance significantly.

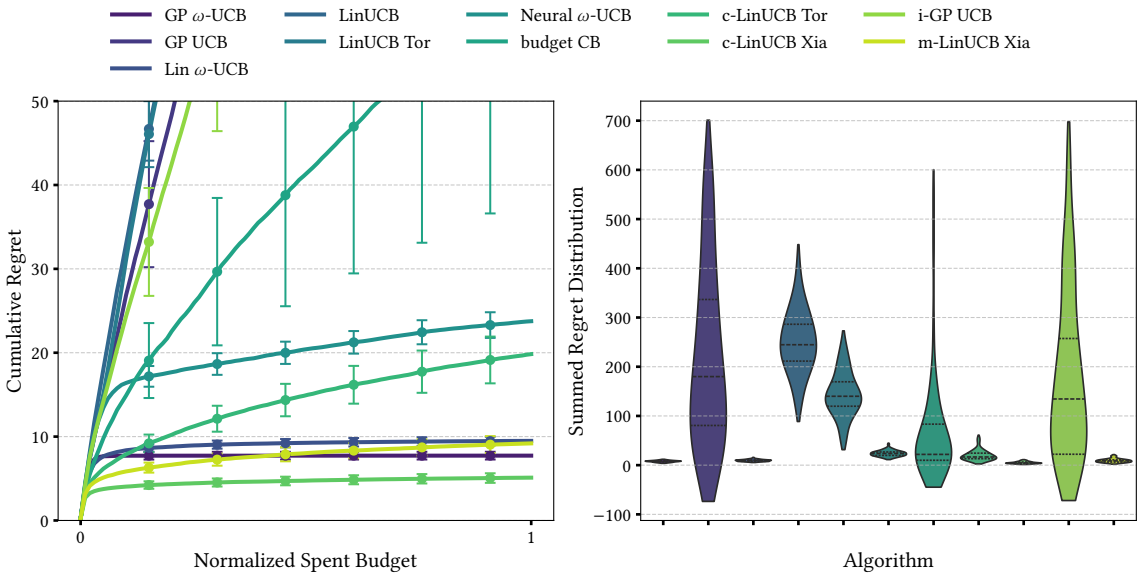

Figure 6.7: Cumulative regret of UCB bandits with linear rewards and continuous costs.

LinUCB takes many rounds to converge, whereas LinUCB Tor achieves faster convergence and performs better overall. The i-GP UCB, and GP UCB policies perform worse and have very high variance across the trials. The violin plot clearly shows this high variance across trials as well. This indicates that the algorithm failed to narrow down their confidence intervals in a manner that fits the data, i.e. too fast or too slow. Notably, the violin plot indicates that GP UCB and i-GP UCB yield nearly identical regret distributions, suggesting that their performance does not significantly differ.

6.2.1.2 Non-Linear Reward and Cost Function

Our first observation in Figure 6.8 is the regret distribution of GP ω-UCB. Unlike the linear setting, its performance is now the best overall, whereas in the linear setting it is just the second best. This indicates strong generalization capabilities when dealing with non-linearity, which we think is a strength of the underlying model and the algorithm combined. Gaussian Processes excel at learning complex functions, but they require efficient exploration to do so effectively.

Similarly, the Neural ω-UCB algorithm exhibits favorable asymptotic behavior, preserving the performance observed in the linear case. Again, we observe that the approach utilizing a Neural Network converges slower than similar policies with different models.

In contrast, the GP UCB and i-GP UCB algorithms continue to struggle, failing to effectively adapt to the non-linear environment. This underlines our hypotheses about the rapid narrowing of confidence intervals in these algorithms.

The summed regret distribution of some algorithms grew less relative to the distribution in the linear setting. This is the case for e.g. c-LinUCB Tor and Neural ω-UCB. The wider confidence intervals of these algorithms likely encourage more exploration. This implicates better performance in environments with more uncertainty.

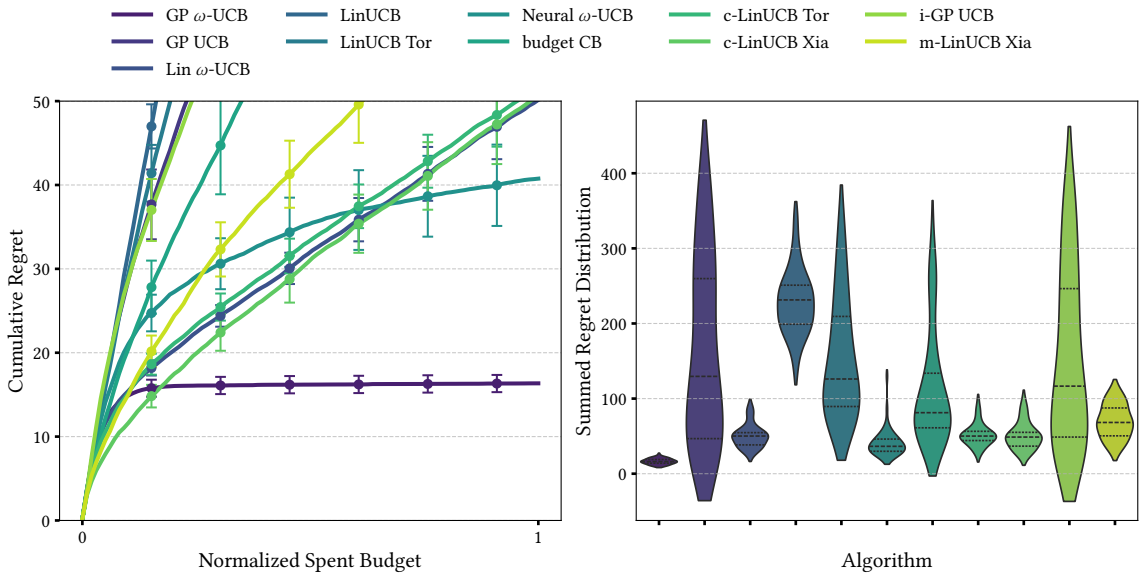

Figure 6.8: Cumulative regret of UCB bandits with non-linear, continuous rewards and costs.

6.2.1.3 Effect of Beta-Distributed Weights

As we can see from figure 6.9, Beta-distributed weights have a greater effect on bandits that compute an upper and lower confidence bound. Other variants remain largely unaffected by this change.

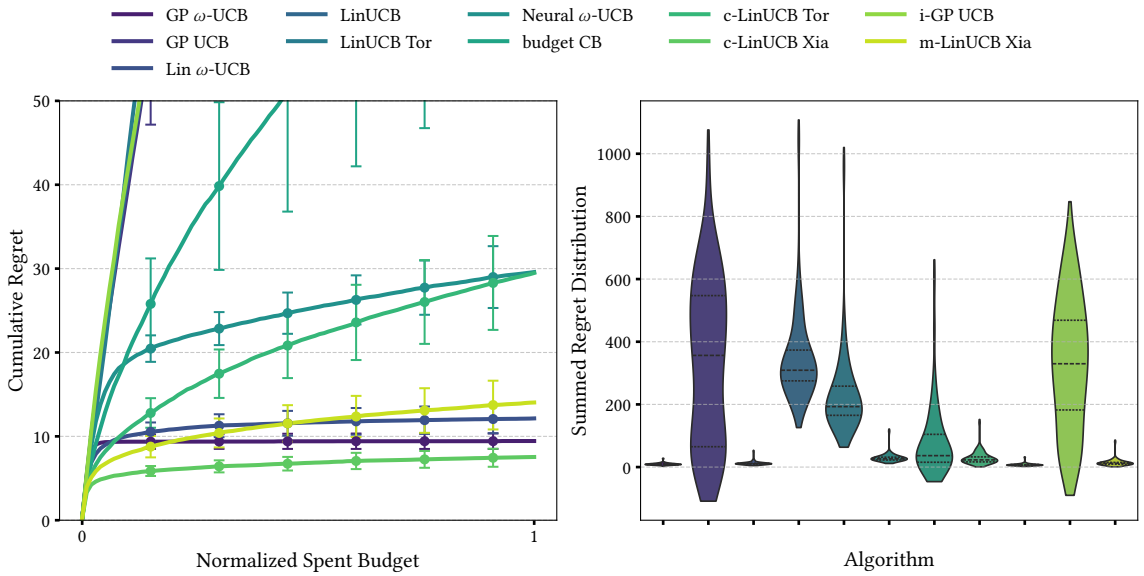

Figure 6.9: Cumulative regret for UCB bandits with linear rewards and costs, $\mathcal{B}(0.5, 0.5)$ sampled weights.

As intended, ω-UCB based bandits perform better in this environment than their competitors due to their design. The line plot suggests that ω-UCB based bandits perform better with a greater budget. However, Beta-distributed weights do not have an as strong effect as initially anticipated. A possible explanation is that the weight vector contains large and small numerical values which cancel each other out, thereby dampening the expected effect. Further analysis could investigate whether a different weight distribution leads to stronger deviations in regret performance.

6.2.1.4 Evaluation in a Bernoulli Setting

Figure 6.10 illustrates the regret distribution of UCB bandits with Bernoulli rewards and costs. The increased variance across all algorithms is a key observation. Compared to the continuous rewards in Figure 6.7, the summed regret distributions show a considerably broader spread. This underlines that the high variance of Bernoulli-distributed feedback introduces greater uncertainty in decision-making.

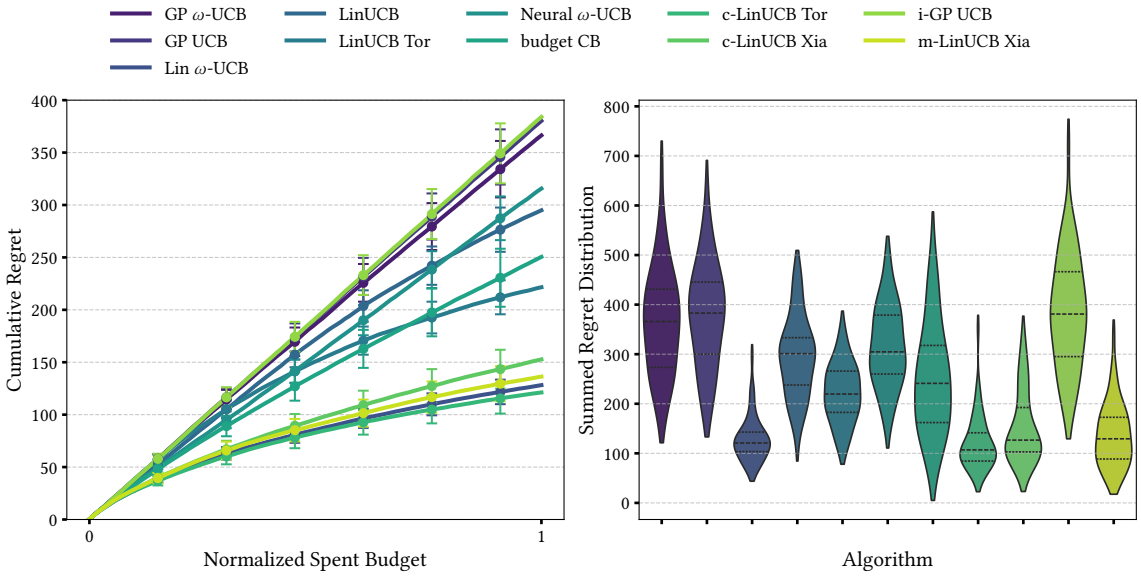

Figure 6.10: Cumulative regret of UCB bandits with linear, Bernoulli rewards and costs.

From a performance perspective, we observe shifts in the performance ranking. The c-LinUCB Tor bandit now outperforms c-LinUCB Xia. The wider confidence intervals likely drive this improvement, as they allow for more conservative decision-making under high uncertainty. Since Bernoulli rewards inherently introduce high variance, a more cautious exploration strategy prevents excessive exploitation of misleading early observations.

In contrast to the continuous reward setting, GP ω-UCB exhibits significantly higher regret, suggesting that it struggles with the increased variance introduced by Bernoulli costs. Since this algorithm performs well with a linear model but struggles with the Gaussian Process, it again suggests that the issue lies in the model rather than the algorithm itself. The Gaussian Process likely struggles to learn the relationship due to the discrete nature of Bernoulli rewards.

6.2.1.5 Combining Beta and Bernoulli Distributions

Figure 6.11 illustrates the cumulative regret of UCB bandits with Bernoulli rewards and costs and $\mathcal{B}(0.5, 0.5)$ distributed function weights.

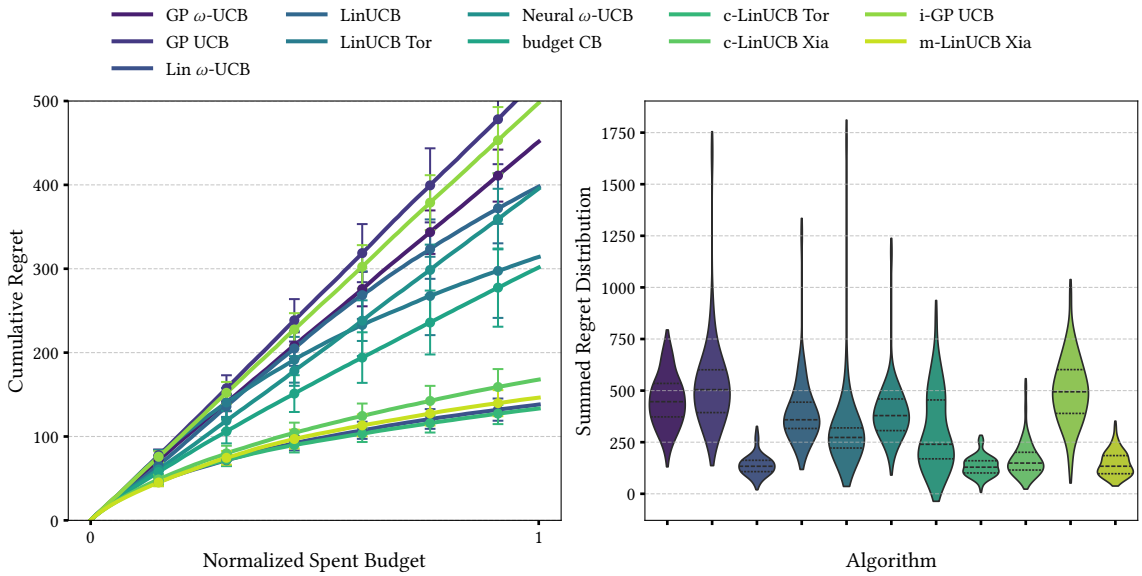

Figure 6.11: Cumulative regret of UCB bandits with linear, Bernoulli rewards and costs, $\mathcal{B}(0.5, 0.5)$ sampled weights.

A key observation is that all algorithms exhibit increased regret, as the reward-to-cost ratio rises due to smaller cost values. Aside from the overall regret increase, c-LinUCB Tor and c-LinUCB Xia continue to perform well, despite not explicitly modeling uncertainty in the cost computation. Incorporating an uncertainty estimate in the reward computation appears sufficient for efficient exploration, whereas applying the same approach to costs results in excessive and uncontrolled exploration.

6.2.2 Evaluation with Facebook Advertisement Data

After evaluating performance in synthetic environments, we now analyze the real-world applicability of these algorithms using Facebook advertisement data. Figure 6.12 shows the results of this trial.

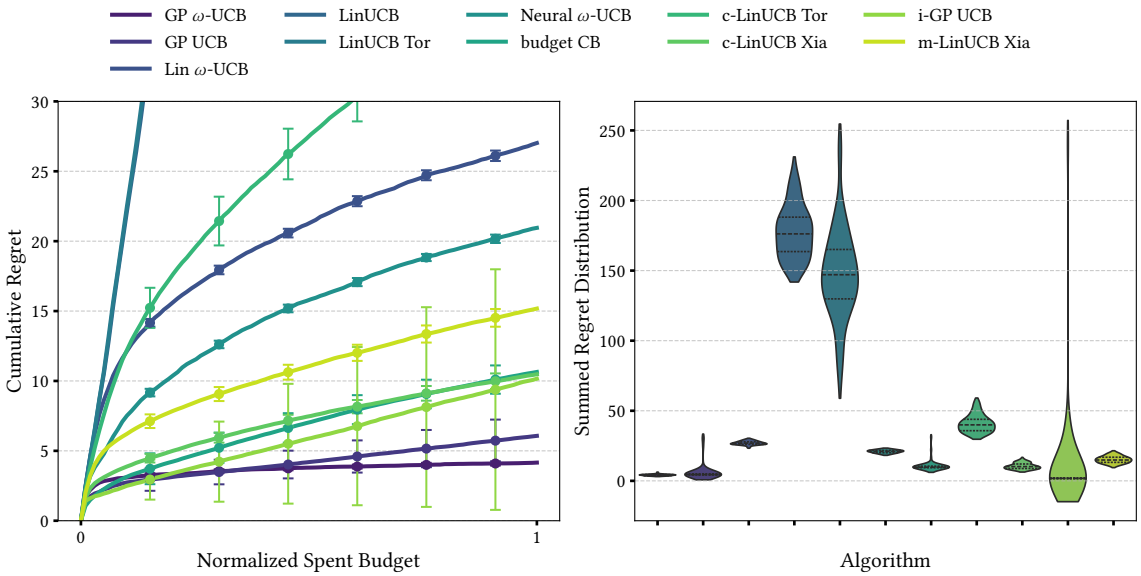

Figure 6.12: Cumulative regret of UCB bandits on Facebook advertisement data with continuous rewards and costs.

In this setting, both the GP ω-UCB and GP UCB algorithms perform well. Surprisingly, GP UCB performs well in this setting, despite its poor performance in all other settings. This supports the hypotheses that the confidence intervals of these algorithms shrink too quickly. The seemingly lower variance of outputs in this setting allows for more stable performance, as indicated by the violin plots of all other algorithms as well.

The Neural ω-UCB algorithm displays behavior quite similar to that of Lin ω-UCB, suggesting that the linear model still captured the relationship to a certain degree. However, Neural ω-UCB still manages to converge faster in this setting.

Budget-CB shows better performance than in previous trials, as well as considerably less variance in results.

6.3 Comparison of Thompson Sampling Policies

This section presents the results of Thompson Sampling policies in different environments. Table 6.3 summarizes the parameters used for the experiment.

Table 6.3: Parameters to evaluate Thompson Sampling policies

Algorithm	Parameter	Reference
Contextual TS	$\nu = 0.1$	[1]
β-TS	independent	-
Contextual c-TS	$\nu = 0.1$	[1]
GP TS Chowdhury	$\delta = 0.1$	[3]
GP TS Srinivas	independent	-
GP β-TS	independent	-

6.3.1 Evaluation with Synthetic Data

We begin by evaluating the Thompson Sampling algorithms with synthetically generated data as described in chapter 5.

6.3.1.1 Linear Reward and Cost Function

In this section, we operate the bandits against linear reward and cost functions and the learner interprets the output as continuous values. Figure 6.13 shows the results for this trial.

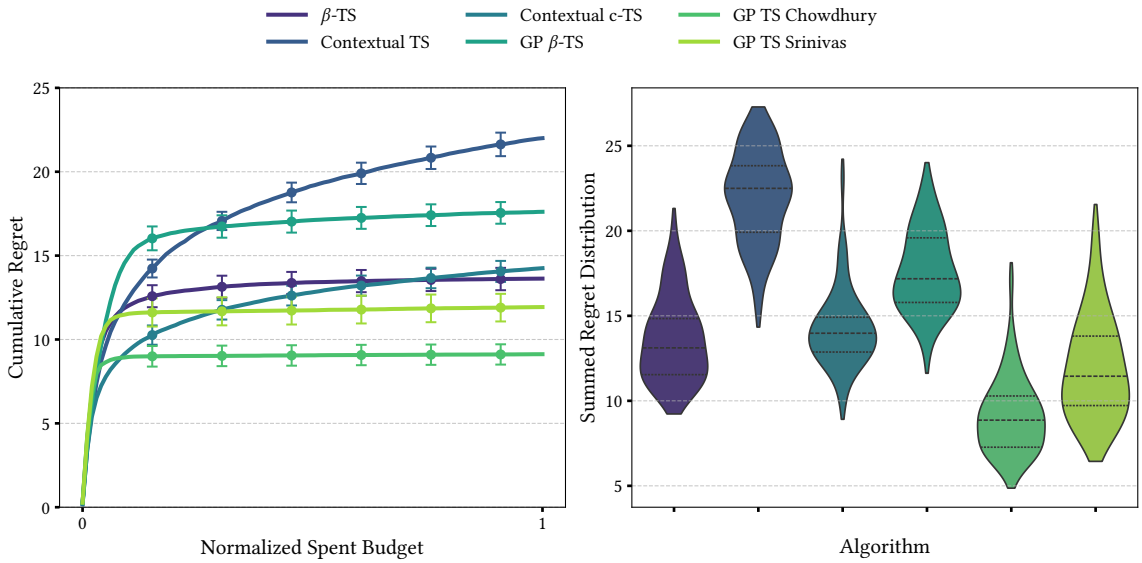

Figure 6.13: Cumulative regret for Thompson Sampling bandits with linear rewards and continuous costs.

All TS algorithms demonstrate strong asymptotic behavior. The GP TS Chowdhury bandit achieves the lowest cumulative regret, while Contextual TS shows the highest regret. Variance across the different algorithms remains consistent, indicating that differences in regret result mainly from variations in the early exploration strategies.

Contrary to the UCB-based algorithms, variants that directly use the predicted cost value without incorporating uncertainty do not achieve the best performance. These algorithms display slightly worse asymptotic behavior and convergence properties than their counterparts, which explicitly account for uncertainty.

6.3.1.2 Non-Linear Reward and Cost Function

Now, we evaluate the bandits with non-linear reward and cost functions, which still yield continuous values. Figure 6.14 presents the results.

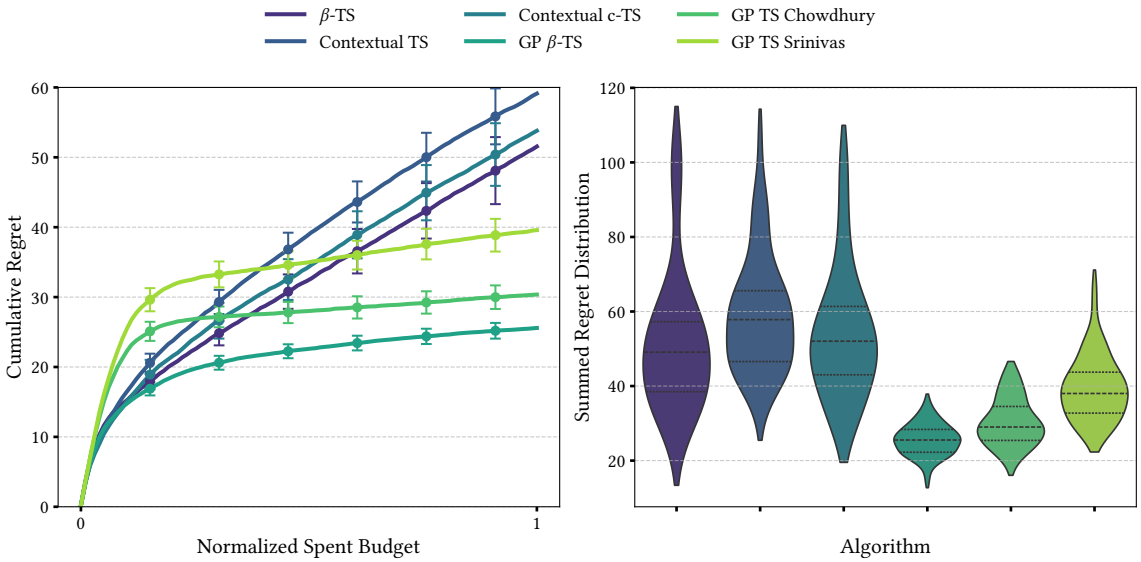

Figure 6.14: Cumulative regret of Thompson Sampling bandits with non-linear, continuous rewards and costs.

As anticipated, the linear approaches struggle because a linear model cannot accurately approximate the non-linear function over the entire interval. We observe a significant shift in the performance of Gaussian Process-based approaches. In the linear trial, GP TS Chowdhury performed the best, while GP β-TS exhibited the worst performance, with nearly double the regret. In the non-linear setting, we observe a reversed performance: GP β-TS now outperforms the others. This reversal occurs because, in non-linear settings, an overly aggressive confidence interval shrinkage leads to premature exploitation. Since GP β-TS maintains broader uncertainty estimates for longer, it can explore more effectively and avoid getting stuck in suboptimal policies. In contrast, GP TS Chowdhury, which performed well under linearity due to its faster convergence, now struggles as it fails to adapt to the complex reward landscape. This bandit takes this trade-off for it's better performance in a low uncertainty setting.

6.3.1.3 Effect of Beta-Distributed Weights

Sampling the function weights from a $\mathcal{B}(0.5, 0.5)$ distribution has a differential impact across the algorithms. While the cumulative regret increases slightly for all algorithms, Contextual TS and β-TS remain relatively unaffected compared to the other approaches.

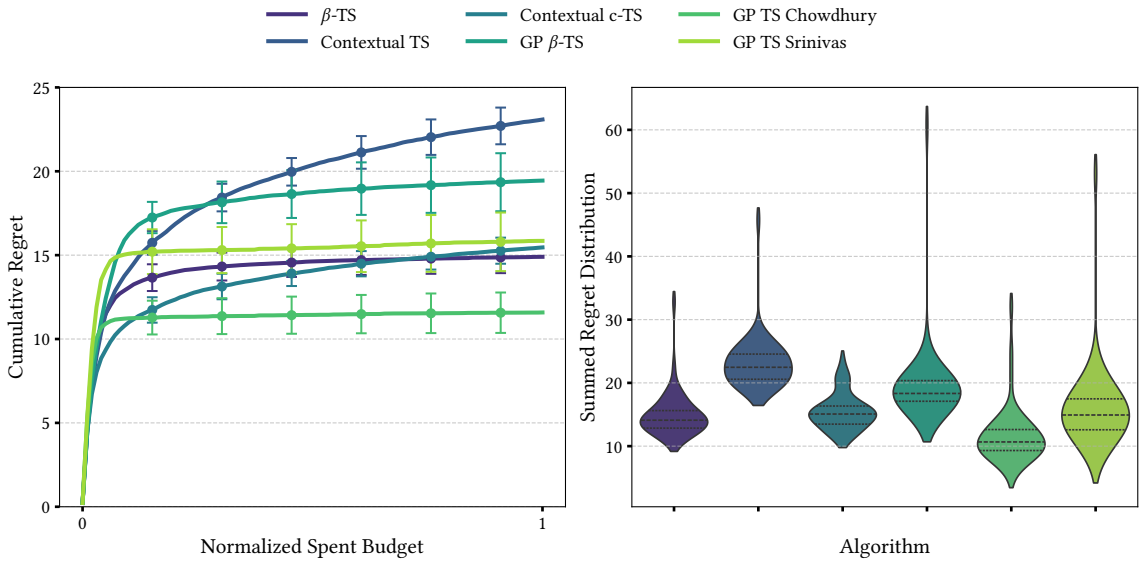

Figure 6.15: Cumulative regret for Thompson Sampling bandits with linear rewards and costs, $\mathcal{B}(0.5, 0.5)$ sampled weights.

This effect becomes particularly noticeable when comparing GP TS Srinivas with Contextual TS and β-TS. In the linear setting (Figure 6.13), GP TS Srinivas initially outperformed both algorithms. In the Beta-distributed setting we see in Figure 6.15, its performance deteriorates, leading to higher cumulative regret compared to both Contextual TS and β-TS. It suggests that certain bandit algorithms demonstrate robustness to variations in the reward and cost distributions, whereas others remain more sensitive to such changes.

6.3.1.4 Evaluation in a Bernoulli Setting

Figure 6.16 displays the cumulative regret of Thompson Sampling bandits when both rewards and costs follow a Bernoulli distribution. As anticipated, the regret variance increases considerably compared to the continuous setting. This effect likely results from the discrete nature of the observations.

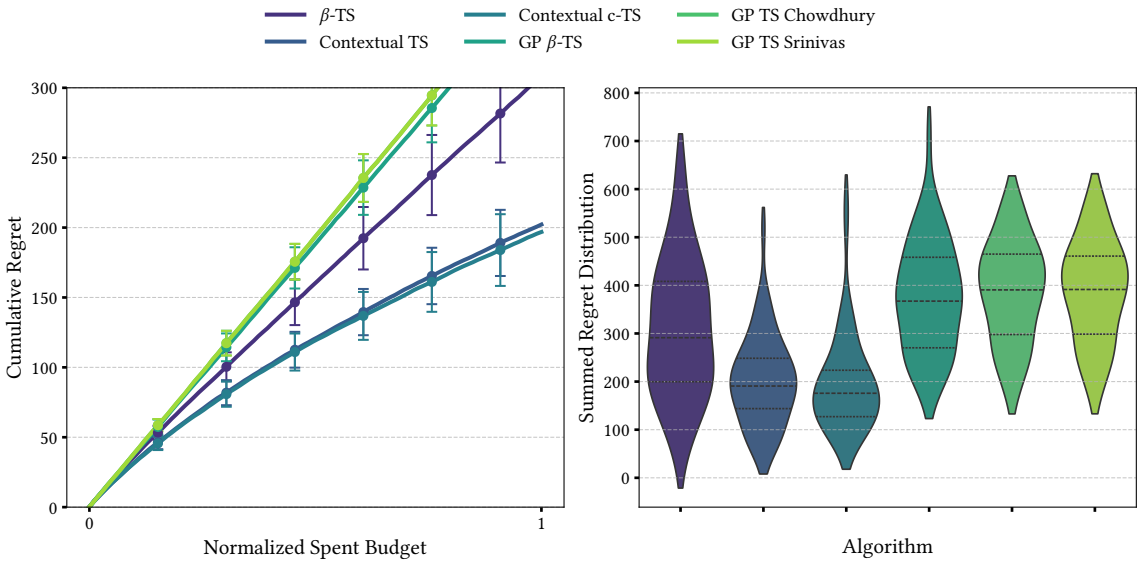

Figure 6.16: Cumulative regret of Thompson Sampling bandits with linear rewards and Bernoulli costs.

We see a similar trend as with the UCB-based algorithms: the asymptotic behavior of most approaches deteriorates in comparison to the continuous setting, resulting in higher overall regret. However, certain algorithms appear to cope with the increased uncertainty more effectively. In particular, Contextual TS and Contextual c-TS outperform the others in this setting. Since Bernoulli rewards and costs introduce high variance in early observations, algorithms that explore more extensively reduce the risk of being misled by initial suboptimal samples. This explains why Contextual TS and Contextual c-TS outperform others, as their broader exploration avoids locking into misleading early estimates too soon.

Selecting different hyperparameters that promote even more exploration could further amplify this effect. The primary challenge in this setting lies in the fact that algorithms require significantly more data to accurately approximate the mean reward and cost. When two arms have very similar rewards, distinguishing between them requires numerous samples. This is why we think most algorithms failed to gather sufficient data within the given budget, their performance remained suboptimal.

Again, GP based algorithms perform considerably worse in this setting, which reproduces our observation for the UCB and Greedy bandits that utilize a GP.

6.3.1.5 Combining Beta and Bernoulli Distributions

Similar to the greedy variants, sampling the weights of the linear function from a $\mathcal{B}(0.5, 0.5)$ distribution exacerbates the effects observed in Figure 6.16.

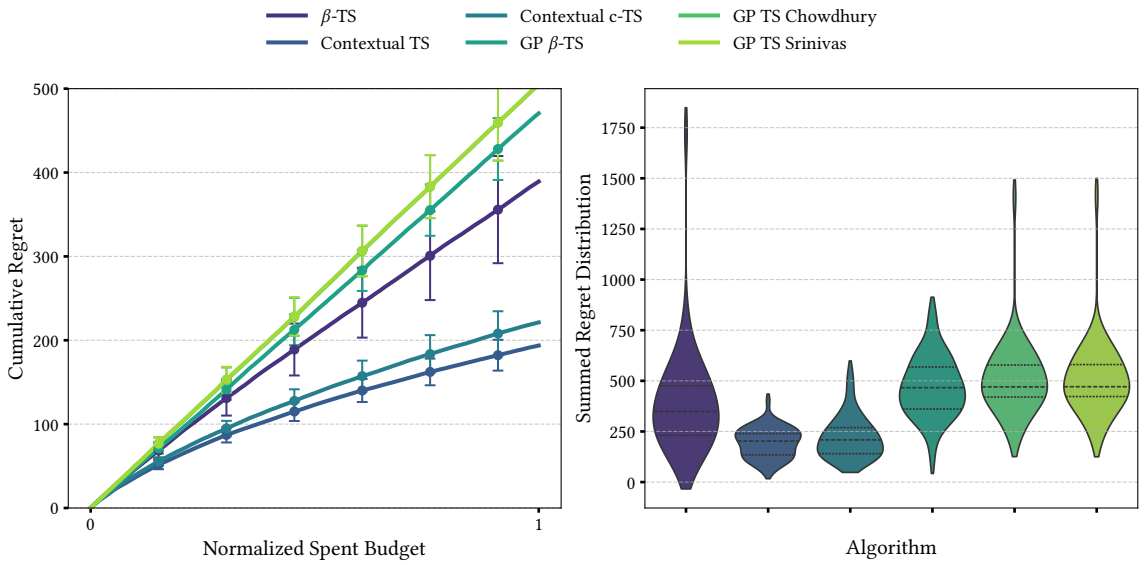

Figure 6.17: Cumulative regret of Thompson Sampling bandits with linear, Bernoulli rewards and costs, $\mathcal{B}(0.5, 0.5)$ sampled weights

As we see, the overall regret increases compared to the previous setting. This is similar to what we observed from the other algorithms in this setting. Some algorithms seem to struggle more with the increased variance, while others maintain relatively stable performance. The effect from the previous Bernoulli trial is further enhanced by the beta distributed weights.

6.3.2 Evaluation with Facebook Advertisement Data

On Facebook advertisement campaign data, the two GP TS variants—GP TS Chowdhury and GP TS Srinivas—outperform all other algorithms by a significant margin. However, the variance for GP TS Srinivas is slightly higher compared to GP TS Chowdhury. The key distinction between them is the use of mutual information gain in GP TS Chowdhury, which appears to enhance performance in this setting.

Additionally, we observe that all algorithms utilizing a linear model perform considerably worse. Among these, β-TS stands out as the best, demonstrating superior adaptation to the real-world data. The GP β-TS algorithm exhibits similar asymptotic behavior to other GP TS variants, suggesting that an improved variance contraction mechanism—potentially through a refined scaling parameter—could lead to even better performance.

Notably, the variance of Gaussian Process-based algorithms remains remarkably stable across different experimental settings with continuous values. This underscores their robustness in real-world applications. However, while GP based algorithms demonstrate robust variance control and low regret, they require careful tuning of hyperparameters

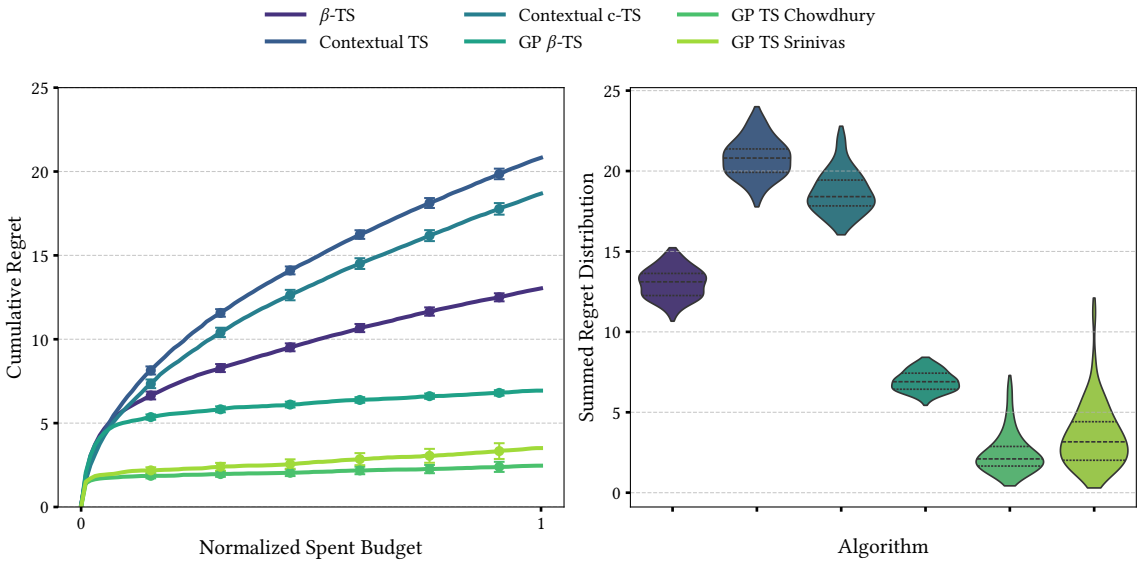

Figure 6.18: Cumulative regret of Thompson Sampling bandits on Facebook advertisement data with continuous rewards and costs.

and sufficient computational resources to fully leverage their advantages in real-world settings.

6.4 Evaluation against an Adversary

In this chapter, we evaluate the performance of the proposed bandit algorithms in an adversarial setting, where the reward and cost functions periodically change. Specifically, the adversary alters the reward and cost functions every 200 rounds. We set this interval to 200 rounds to provide sufficient time for the models to approximate the underlying functions before any changes occur.

Since these algorithms target stationary environments, we expect a notable increase in regret. By tracking regret progression over time we reveal whether some algorithms demonstrate resilience to these changes, despite lacking explicit adaptation for adversarial settings.

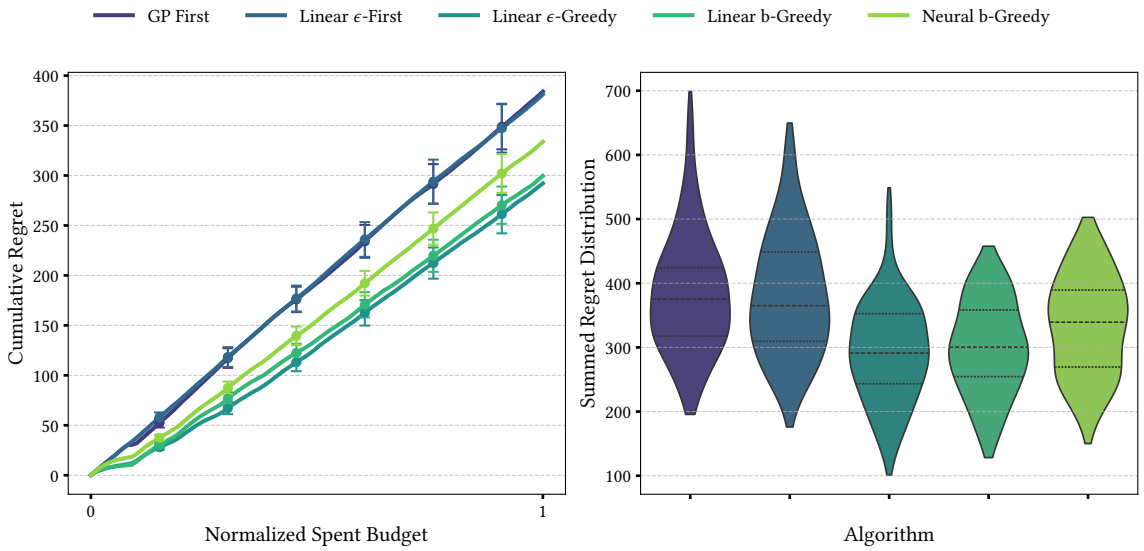

Figure 6.19: Cumulative regret and summed regret distribution of Greedy bandits on synthetic data with continuous rewards and costs in an adversarial setting.

Across Figures 6.19, 6.20, and 6.21, we observe distinct "elbows" in the cumulative regret curves whenever the adversary alters the reward and cost functions. This pattern indicates that all algorithms experience a temporary surge in regret following each function shift, necessitating additional exploration to re-learn the optimal strategy.

Interestingly, some bandits that did not performer well in the linear continuous setting now exhibit improved performance. The Linear ϵ-Greedy algorithm in Figure 6.19 serves as a notable example. Previously, its static exploration strategy led to subpar performance in the linear setting, as it continued exploring even after the learner found an optimal strategy. In the adversarial setting, this persistent exploration enables the algorithm to adapt more effectively to changes in later rounds.

Lin ω-UCB, which performed well in the linear continuous setting, remains one of the top-performing algorithms in the adversarial environment (Figure 6.20). This choice of the variance scaling parameter $\eta = 1$, which corresponds to the maximum variance setting, likely drives this success. The high variance forces the algorithm to continuously question

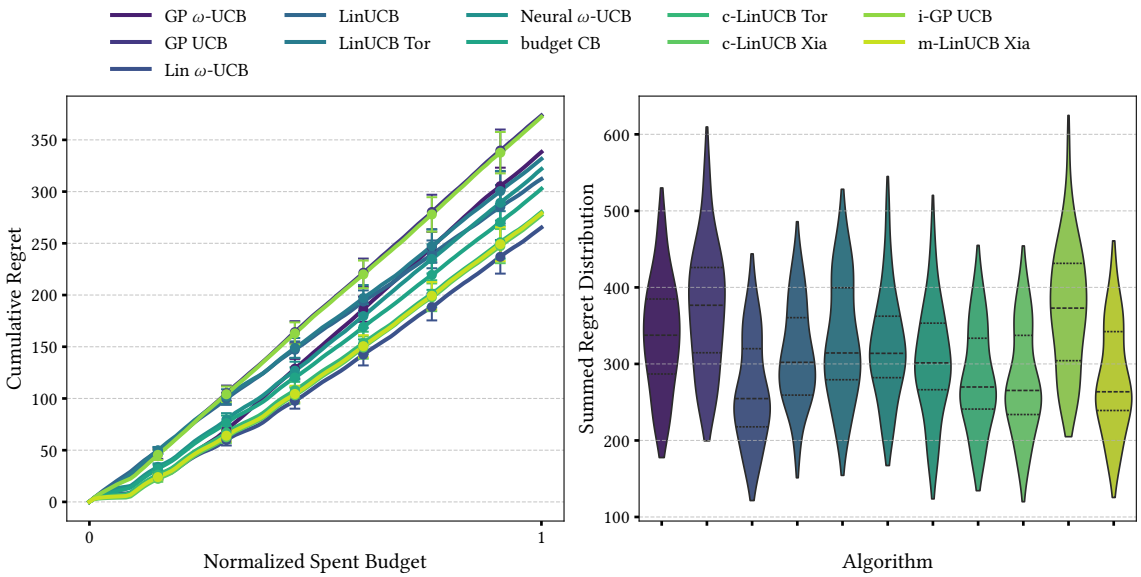

Figure 6.20: Cumulative regret and summed regret distribution of UCB bandits on synthetic data with continuous rewards and costs in an adversarial setting.

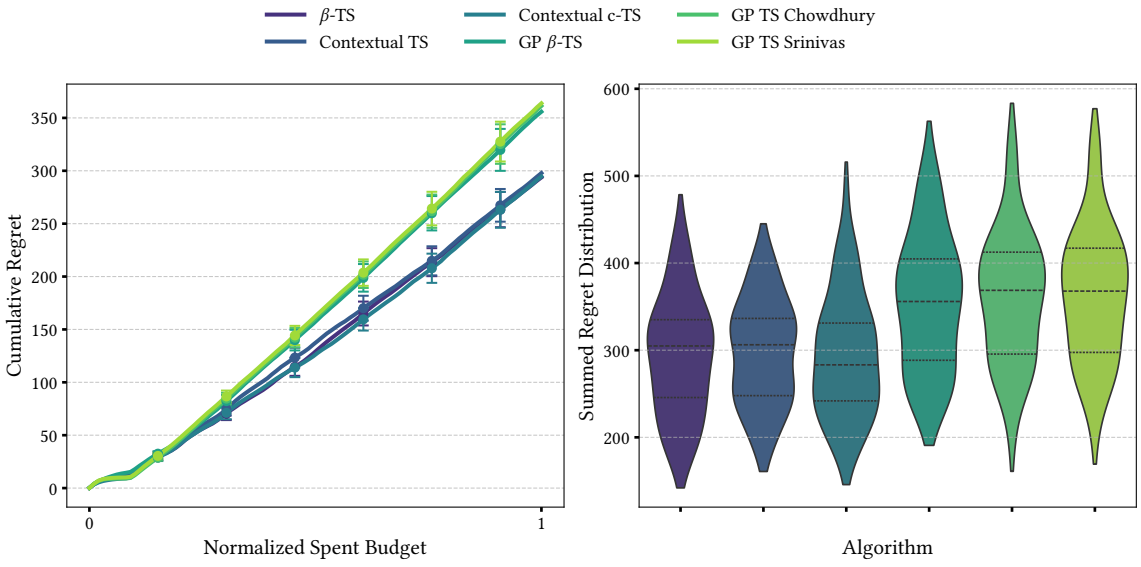

Figure 6.21: Cumulative regret and summed regret distribution of Thompson Sampling bandits on synthetic data with continuous rewards and costs in an adversarial setting.

its estimates, preventing it from prematurely committing to a suboptimal algorithm. As a result, Lin ω-UCB remains more adaptable to abrupt environmental changes.

Thompson Sampling bandits appear almost reversed in their performance compared to the linear setting (Figure 6.21). Approaches that performed best in the linear environment now exhibit worse performance. Their low variance in prior distributions likely accelerates convergence in stationary environments but hinders adaptation in dynamic settings. Once these algorithms converge to a seemingly optimal strategy, the learner would have to retrain the model to adapt to changes. These findings reveal a fundamental limitation of standard bandit algorithms in adversarial settings. Future research could explore integrating change detection mechanisms to enable bandits to adapt more rapidly to sudden shifts in the environment. Fouché et al. published a promising approach in this field. They integrated the ADWIN change detection algorithm into a Thompson Sampling-based bandit model to result in an adaptive TS bandit [6].

6.5 Bagging Bandits with EXP4

In this section, we explore the bandits with expert advice framework in a budgeted contextual setting, with a particular focus on EXP4. Our EXP4 algorithm weights experts according to the difference between their predicted reward and the actual observed reward. Experts that provide more accurate predictions receive exponentially larger weights, increasing their selection probability in future rounds. This process allows EXP4 to dynamically shift its strategy towards the best-performing expert over time.

We find it important to note that EXP4 rather represents a framework than a complete bandit algorithm. To select an arm, we integrate several bandit strategies from Chapter 4 into the EXP4 framework. The idea is that EXP4 can dynamically choose the bandit that best fits the current environment. In our experiment, EXP4 utilizes four experts: GP b-Greedy (a variant of the linear b-Greedy with a Gaussian Process model), c-LinUCB Xia, GP TS Srinivas, and GP TS Chowdhury.

In our setup, we only consider the reward to weight the experts. This results from the similarity between the reward and cost functions. Because of this similarity, an accurate reward prediction represents both models effectively. The reward difference determines the expert's weight, following the classical EXP4 weighting scheme. The work of Lattimore et al. further explains how the weighting in EXP4 works [8].

We can then test the EXP4 algorithm across various environments to evaluate its performance. This approach is particularly useful when the exact setting in which the bandits operate remains unknown. In such cases, we cannot simply choose the best bandit beforehand. EXP4 offers flexibility to adapt to the environment while learning the cost-reward-context relationship. Figures 6.22 and 6.23 display our results in a linear and non-linear environment. For comparison, we select well performing bandits from previous experiments.

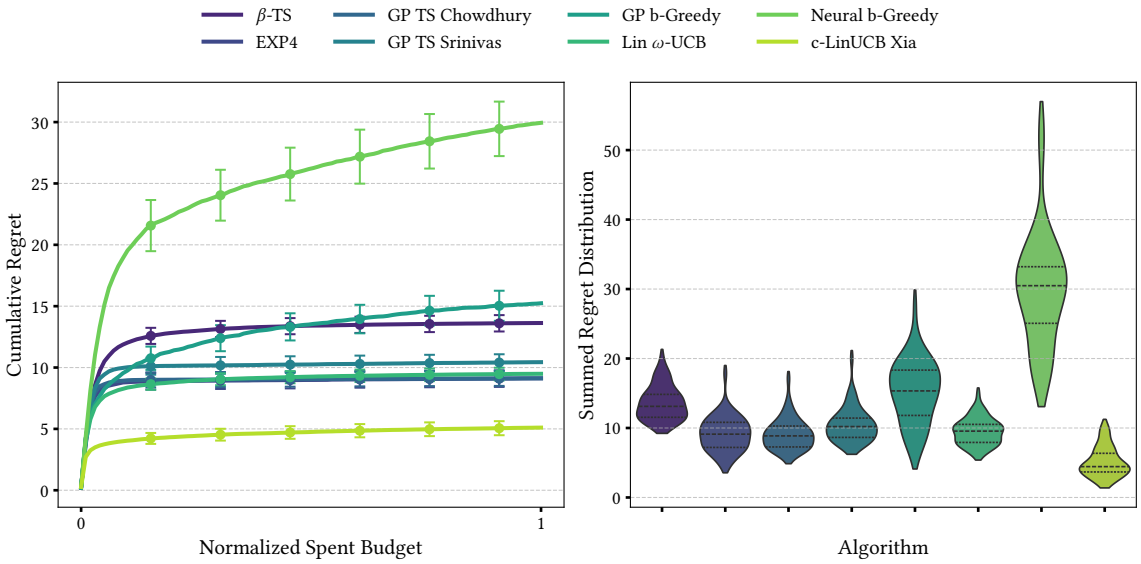

Figure 6.22: Cumulative regret and summed regret distribution of EXP4 on synthetic data with linear, continuous rewards and costs.

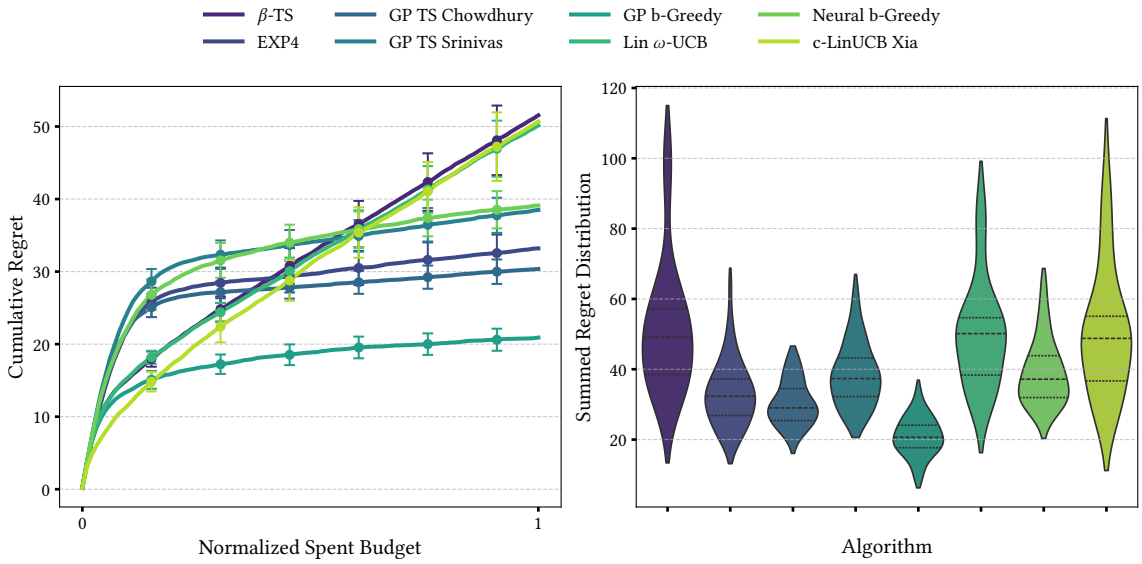

Figure 6.23: Cumulative regret and summed regret distribution of EXP4 on synthetic data with non-linear continuous rewards and costs.

Figure 6.22 shows that the EXP4 bandit achieves relatively low regret, but still performs worse than the best-performing bandit. We expect this outcome because EXP4 requires time to identify the most effective expert (bandit) from its set of experts. Nevertheless, its performance ranks among the top three, with results nearly identical to Lin ω-UCB, showing a slight advantage in later rounds. The key strength of this approach is its ability to perform well across multiple settings simultaneously, as demonstrated in Figures 6.22 and 6.23. In later rounds, EXP4 tends to favor GP-based bandits, as their predictions become more accurate as the linear models with an increasing amount of Data. However, this characteristic can also introduce disadvantages. In our experiment, EXP4 consistently weighted GP TS Srinivas the highest overall, even though c-LinUCB Xia performed better in the linear setting. This occurs because GP-based models generate increasingly accurate predictions as the dataset grows. At a certain point, their predictions surpass those of the linear model, resulting in a preference for the GP-based algorithm in our weighting scheme. We believe this avoidable by employing a more sophisticated metric for the weighting process, allowing for a better balance between the strengths of both linear and non-linear models.

6.6 Summary

In this Chapter, we empirically evaluated various bandit algorithms across multiple experimental settings to analyze their behavior under different reward-cost structures, including adversarial environments and the EXP4 framework.

Among the greedy algorithms, Linear b-Greedy achieved the lowest cumulative regret. However, the most surprising result is the effectiveness of the GP First algorithm, a modification of Linear ϵ-First that dynamically adjusts the exploration budget based on model uncertainty. This approach demonstrated particularly strong performance on the Facebook advertisement dataset.

GP ω-UCB, Lin ω-UCB, and Neural ω-UCB demonstrated strong stability across all environments, maintaining consistent regret trends. While they did not achieve the lowest regret in all cases, they consistently exhibited favorable variance and asymptotic behavior. Additionally, m-LinUCB Xia and the c-LinUCB variants yielded strong results. In contrast, GP UCB, i-GP UCB, LinUCB, and LinUCB Tor performed significantly worse, failing to adapt effectively to most settings except for the advertisement dataset.

GP TS algorithms generally displayed excellent asymptotic behavior in continuous settings. However, in the Bernoulli reward-cost setting, their performance closely resembled that of GP UCB and GP Greedy variants. These results suggest that the models have difficulty handling binary rewards and costs. They may also need a higher degree of exploration to perform optimally. The newly proposed GP β-TS algorithm performed particularly well in the Bernoulli setting, while ranking mid-tier in other scenarios. Its slower variance contraction likely enables extended exploration in environments with high uncertainty, such as those with Bernoulli-distributed rewards and costs. This underlines our suggestion.

On the Facebook advertisement campaign dataset, GP β-TS also achieved strong results, but ultimately GP TS variants perform better. GP β-TS's variance contraction mechanism appears overly conservative, limiting its ability to fully adapt to certain environments. Overall, GP TS algorithms demonstrated the best performance across all settings.

In adversarial environments where reward and cost functions change periodically, algorithms like Linear ϵ-Greedy demonstrated improved performance due to their ongoing exploration, helping them adapt quickly to shifts.Although generally strong, GP TS variants exhibited lower adaptability in these dynamic settings. Finally, the EXP4 framework, which adapts by choosing the best performing expert, showed strong results across a range of environments, although it is never the best in any setting due to exploration of suboptimal experts. Additionally, we suggested that our weighting scheme preferred algorithms with the more accurate model, since our weighting bases on the accuracy of the model.

Overall, the GP TS algorithms exhibited the most consistent performance across a variety of settings, highlighting their robustness and adaptability, especially in real-world scenarios. The suboptimal performance of neural network-based approaches likely results from insufficient hyperparameter tuning, either in the neural network architecture itself or in the bandit algorithm's exploration strategy.

7 Conclusion and Outlook

7.1 Conclusion

In this thesis, we explored the field of contextual budgeted multi-armed bandits (BCMAB) and proposed novel algorithms to enhance decision-making under budget constraints and context information. Budget-conscious decisions are essential for driving economic efficiency, and incorporating contextual information to refine these decisions is critical to maximize reward. By doing so, organizations can achieve more informed, cost-effective outcomes.

This thesis provides a comprehensive introduction into the field of Multi-Armed Bandits, laying the foundation for the novel algorithms introduced in this thesis. We developed novel algorithms based on existing ideas from both budgeted and contextual bandits, aimed at improving the balance between exploration and exploitation by leveraging context information while maintaining cost-efficiency.

We evaluated the proposed algorithms across synthetic, real-world and adversarial environments, analyzing their empirical performance in terms of cumulative regret and behavior if faced with different levels of uncertainty. Our results suggest that some approaches are better adaptable to budgeted settings than others. Notably, Gaussian Process (GP)-based algorithms demonstrated strong generalization capabilities, especially in environments with non-linear reward-cost relationships. However, they struggled in settings with Bernoulli-distributed rewards and costs. The reason for this, we assume, is that GPs are designed to model continuous outputs, which makes them prone to overfitting in environments where observed values are restricted to binary values (0 or 1).

Furthermore, we tested the proposed algorithms on Facebook advertisement data to assess their applicability in real-world settings. On this data, GP TS variants outperformed the others, followed closely by GP ω-UCB and GP UCB. The performance of GP UCB was surprising, given that it had not performed well in other settings. This was likely due to the algorithm decreasing the confidence interval widths too fast.

In the adversarial setting, algorithms such as Lin ω-UCB demonstrated robust performance relative to other proposed algorithms. However, the general performance of the bandits was poor, since they are not designed for this environment. Integrating change detection mechanisms could significantly enhance performance in adversarial or non-stationary environments, allowing algorithms to dynamically adapt to shifting reward and cost distributions. This adaptation is challenging, as the bandit must modify its exploration behavior. A change detection mechanism requires a continuous stream of data from all arms to make informed decisions, which can only be achieved if each arm is played frequently enough.

Motivated by the varying performances of the proposed algorithms across different settings, we constructed a bandit ensemble using the EXP4 framework. Our ensemble exhibited stable results across various settings, since it's weighting mechanism assigns the largest weight to the bandit that makes the most accurate predictions. These findings highlight the effectiveness of ensemble bandits in situations where the environment's specifications are not known beforehand.

7.2 Outlook

While our empirical results already provide valuable insights into the empirical behavior of the proposed algorithms, we identify several opportunities for further research.

Hyperparameter Optimization. One important direction of research is hyperparameter optimization, which aims to identify the best configurations for minimizing cumulative regret, the primary objective of this work. However, hyperparameters are typically fine-tuned for specific environments, which often leads to suboptimal performance in others. It also remains unclear what impact hyperparameter optimization has on theoretical performance guarantees. To overcome this limitation, further research is needed to establish regret bounds that are independent of hyperparameters. While algorithms that do not rely on hyperparameter optimization may experience higher regret compared to optimized algorithms, they are more broadly applicable, as they do not require tailoring to specific environments. This is especially important for application domains where the characteristics of the environment are not well understood. Deriving such algorithms is complex, as it requires metrics capable of quantifying the state of the environment well enough to function across different types of environments.

Improving Budget-Based Exploration. Another promising avenue of research is the further exploration of budget-based strategies to enhance the balance between exploration and exploitation. Currently, most algorithms manage exploration by considering factors such as the number of rounds played or the uncertainty based on model parameters, but they often neglect the remaining budget. The remaining budget can serve as a more precise indicator of how much exploration should be done at a given point, based on the actual resource usage of the bandit rather than the number of rounds played. We demonstrated a similar approach with our GP First bandit, where the algorithm dynamically adjusts the exploration budget based on model parameters, avoiding unnecessary exploration and thereby improving performance. While the performance of GP First could be replicated by optimizing the hyperparameters of the classical ϵ-First algorithm within the same environments, incorporating the budget directly into the decision-making process allows for a more "self-adjusting" approach, eliminating the need for computationally expensive hyperparameter tuning. However, it remains unclear whether this approach will lead to improved regret bounds, which warrants further investigation.

Theoretical Analysis. Beyond empirical validation, future work could focus on establishing theoretical regret bounds for the proposed algorithms. While this goes beyond the scope

of this thesis, deriving guarantees and providing formal proofs would strengthen the theoretical foundation of our proposed algorithms and lay the groundwork for further development. Such analysis is also crucial for making fair comparisons between different bandit algorithms and for gaining a deeper understanding of the strengths and weaknesses of the proposed approaches.

Bibliography

[1] Shipra Agrawal and Navin Goyal. *Thompson Sampling for Contextual Bandits with Linear Payoffs*. 2014. arXiv: 1209.3352 [cs.LG]. URL: https://arxiv.org/abs/1209.3352.

[2] Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. "Finite-time Analysis of the Multiarmed Bandit Problem". en. In: *Machine Learning* 47.2 (2002), pp. 235–256. ISSN: 1573-0565. DOI: 10.1023/A:1013689704352.

[3] Sayak Ray Chowdhury and Aditya Gopalan. *On Kernelized Multi-armed Bandits*. 2017. arXiv: 1704.00445 [cs.LG]. URL: https://arxiv.or%20g/abs/1704.00445.

[4] Sam Cohan. *Building a Multi-Armed Bandit System from the Ground Up: A Recommendations and Ranking Case Study*. 2022. URL: https://medium.com/udemy-engineering/building-a-multi-armed-bandit-system-from-the-ground-up-a-recommendations-and-ranking-case-study-b598f1f880e1 (visited on 03/09/2025).

[5] Emile Contal, Vianney Perchet, and Nicolas Vayatis. "Gaussian Process Optimization with Mutual Information". In: *ArXiv* abs/1311.4825 (2013). URL: https://api.semanticscholar.org/CorpusID:9474720.

[6] Edouard Fouché, Junpei Komiyama, and Klemens Böhm. "Scaling Multi-Armed Bandit Algorithms". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. KDD '19. Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 1449–1459. ISBN: 9781450362016. DOI: 10.1145/3292500.3330862.

[7] Marco Heyden et al. "Budgeted Multi-Armed Bandits with Asymmetric Confidence Intervals". In: *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '24. New York, NY, USA: Association for Computing Machinery, Aug. 2024, pp. 1073–1084. ISBN: 9798400704901. DOI: 10.1145/3637528.3671833.

[8] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. URL: https://www.cambridge.org/core/books/bandit-algorithms/8E39FD004E6CE036680F90DD0C6F09FC.

[9] Madis Lemsalu. *Facebook Ad Campaign Dataset*. https://www.kaggle.com/datasets/madislemsalu/facebook-ad-campaign. (Visited on 03/11/2025).

[10] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". In: *Proceedings of the 19th international conference on World wide web*. arXiv:1003.0146 [cs]. 2010, pp. 661–670. DOI: 10.1145/1772690.1772758.

[11] Lihong Li et al. "A Contextual-Bandit Approach to Personalized News Article Recommendation". en. In: *Proceedings of the 19th international conference on World wide web*. arXiv:1003.0146 [cs]. Apr. 2010, pp. 661–670. DOI: 10.1145/1772690.1772758.

[12] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. Available for download in electronic format. The MIT Press, 2006. ISBN: 0-262-18253-X. URL: https://www.gaussianprocess.org/gpml/ (visited on 03/09/2025).

[13] Niranjan Srinivas et al. "Gaussian Process Optimization in the Bandit Setting: No Regret and Experimental Design". In: *IEEE Transactions on Information Theory* 58.5 (2012), pp. 3250–3265. ISSN: 0018-9448, 1557-9654. DOI: 10.1109/TIT.2011.2182033. URL: http://arxiv.org/abs/0912.3995.

[14] Long Tran-Thanh et al. "epsilon-first policies for budget-limited multi-armed bandits". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI'10. Atlanta, Georgia: AAAI Press, 2010, pp. 1211–1216.

[15] Yingce Xia et al. "Budgeted Bandit Problems with Continuous Random Costs". In: *Asian Conference on Machine Learning*. Ed. by Geoffrey Holmes and Tie-Yan Liu. Vol. 45. Proceedings of Machine Learning Research. Hong Kong: PMLR, 2016, pp. 317–332. URL: https://proceedings.mlr.press/v45/Xia15.html.

[16] Yingce Xia et al. "Finite budget analysis of multi-armed bandit problems". In: *Neurocomputing*. Special Issue on Machine Learning 258 (2017), pp. 13–29. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2016.12.079.

[17] Yingce Xia et al. "Thompson sampling for budgeted multi-armed bandits". In: *Proceedings of the 24th International Conference on Artificial Intelligence*. IJCAI'15. Buenos Aires, Argentina: AAAI Press, 2015, pp. 3960–3966. ISBN: 9781577357384. URL: https://dl.acm.org/doi/10.5555/2832747.2832801.

[18] Yisong Yue, Sue Ann Hong, and Carlos Guestrin. *Hierarchical Exploration for Accelerating Contextual Bandits*. arXiv:1206.6454 [cs, stat]. 2012. DOI: 10.48550/arXiv.1206.6454.

[19] Weitong Zhang et al. *Neural Thompson Sampling*. arXiv:2010.00827 [cs, stat]. Dec. 2021. DOI: 10.48550/arXiv.2010.00827.

[20] Mengying Zhu et al. *Adaptive Portfolio by Solving Multi-armed Bandit via Thompson Sampling*. 2019. arXiv: 1911.05309 [cs.LG]. URL: https://arxiv.org/abs/1911.05309.