A Novel Switch-Type Policy Network for Resource Allocation Problems

Jerrod Wigmore[1], Brooke Shrader[2], and Eytan Modiano[1]

[1]Massachusetts Institute of Technology
[2]MIT Lincoln Laboratory

***Abstract*—Deep Reinforcement Learning (DRL) has become a powerful tool for developing control policies in queueing networks, but the common use of Multi-layer Perceptron (MLP) neural networks in these applications has significant drawbacks. MLP architectures, while versatile, often suffer from poor sample efficiency and a tendency to overfit training environments, leading to suboptimal performance on new, unseen networks. In response to these issues, we introduce a switch-type neural network (STN) architecture designed to improve the efficiency and generalization of DRL policies in queueing networks. The STN leverages structural patterns from traditional non-learning policies, ensuring consistent action choices across similar states. This design not only streamlines the learning process but also fosters better generalization by reducing the tendency to overfit. Our work presents three key contributions: first, the development of the STN as a more effective alternative to MLPs; second, empirical evidence showing that STNs achieve superior sample efficiency in various training scenarios; and third, experimental results demonstrating that STNs match MLP performance in familiar environments and significantly outperform them in new settings. By embedding domain-specific knowledge, the STN enhances the Proximal Policy Optimization (PPO) algorithm's effectiveness without compromising performance, suggesting its suitability for a wide range of queueing network control problems.**

I. Introduction

In the realm of queueing network controls, Deep Reinforcement Learning (DRL) has emerged as a promising approach to learn high-performing network control policies [1]–[5]. As is standard in the DRL community, these works employ the Multi-layer Perceptron (MLP) neural network architecture for its flexibility and ability to serve as a universal function approximator [6]. However, this flexibility comes with significant drawbacks, particularly in terms of sample efficiency and generalization capabilities.

This material is based upon work supported by the Department of the Air Force under Air Force Contract No. FA8702-15-D-0001, and by the National Science Foundation grant number CNS-2148183.

This material is based upon work supported by the Department of the Air Force under Air Force Contract No. FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Department of the Air Force.

Training MLP policy networks in DRL frameworks often requires a substantial number of interactions with the training environment(s) before learning a policy that performs well in the training environment. Furthermore, the policies learned through this architecture tend to overfit to the training environments, resulting in poor performance when applied to new, unseen environments. This overfitting issue poses a major challenge, as real-world stochastic networks are dynamic and varied, necessitating robust and generalizable control policies.

To address these challenges, we propose a novel policy network architecture tailored specifically for a large class of queueing network control problems: the switch-type neural network (STN). Our approach is inspired by structural patterns observed in many non-learning policies within the network control literature, which often exhibit "switch-type" behavior [7]. Specifically, these policies maintain consistency in their action selection across similar states, ensuring that if action i is chosen for state $(s_1, ..., s_i, ..., s_K)$, then action i will be chosen for state $(s_1, ..., s_i+1, ..., s_K)$, where the state element s_i may represent the queue size at a node or channel conditions.

By incorporating this switch-type structure into the neural network design, we aim to enhance both the sample efficiency during training and the generalization performance post training. Sample efficiency is improved because we restrict the search space to switch-type policies during training, rather than the class of all policies that can be represented by an MLP-based policy network. Generalization performance is improved because we limit the ability of the neural network to memorize the training data, and instead learn a policy with consistent structure across the state-space.

Our contributions are threefold: (1) we introduce a novel switch-type policy architecture for queueing network controls as an alternative to the MLP policy network architecture, (2) we provide extensive empirical validation demonstrating improved sample efficiency in both single-environment and multi-environment training scenarios, and (3) we conduct thorough experimental validation showing that trained switch-type policy networks match the performance of the MLP policy architecture when tested on the same environments seen during training, and significantly outperform the MLP architecture when tested on unseen environments.

ISBN 978-3-903176-73-7 © 2025 IFIP

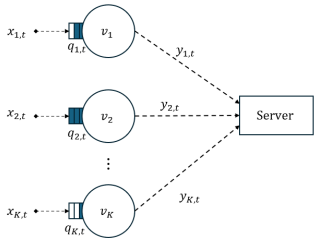


Fig. 1: Example of a single-hop scheduling environment. Packets arrive to each queue according to their independent arrival distributions. The service rate between each queue and the server varies according to an independent i.i.d. process. The server selects one of the K queues to serve in each time-step.

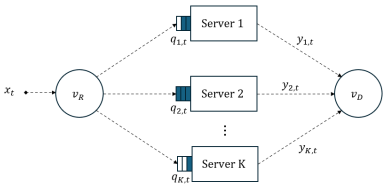


Fig. 2: Example of a multi-path routing environment. A single packet enters the network at the routing node labeled v_R. The policy determines which of the K queues the packet is then routed to. Each server maintains its own separate queue. After a packet is processed by any server, it is sent to node v_D where it immediately leaves the network.

II. Background

We start by providing an overview of the resource allocation problems of interest. We then formally define switch-type policies and provide numerous examples in the literature of network control policies that are switch-type. We then provide background on DRL and the MLP architecture.

A. Resource Allocation Problems

We are going to focus on discrete time resource allocation where a network control agent chooses to interact with one of K components (queues, servers, users, etc) in each time-step $t \in (0, 1, 2, ...)$. The state of component k, $\mathbf{s}_{k,t}$, evolves according to one of two Markov transition distributions $P(\mathbf{S}_{k,t+1}|\mathbf{s}_{k,t}, a_t = k)$ or $P(\mathbf{S}_{k,t+1}, \mathbf{s}_{k,t}, a_t \neq k)$ where $\mathbf{S}_{k,t+1}$ is a random variable representing the next state and a_t is the "action" or decision made by the central network controller in time t. The objective is to minimize the average expected cost:

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}[c(\mathbf{S}_t)] = \lim_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \sum_{k=1}^{K} \mathbb{E}[c(\mathbf{S}_{k,t})] \quad (1)$$

where $\mathbf{S}_t = (\mathbf{S}_{1,t}, ..., \mathbf{S}_{K,t})$ represents the network state at time t, and $c(\mathbf{s}_{k,t})$ is a cost function. This problem can be modeled as a Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, c, \rho_0)$ where the state-space $\mathcal{S}$ is the set of all possible network states, $\mathcal{A}$ is a finite set of actions that can be taken in each state, $P(\mathbf{S}_{t+1}|\mathbf{s}_t, a_t) = \prod_{k=1}^{K} P(\mathbf{S}_{k,t+1}|\mathbf{s}_{k,t}, a_t)$ is the Markov transition probability function, $c(\mathbf{s})$ the cost function, and $\rho_0(\mathbf{s})$ is the initial state-distribution. The objective is to design or learn a Markov policy $\pi : \mathcal{S} \mapsto \Delta(\mathcal{A})$ where $\Delta(\mathcal{A})$ is a probability simplex over $\mathcal{A}$ as to minimize equation (1). For deterministic policies we use $\pi(\mathbf{s}) = i$ to denote that the policy π chooses action $a = i$ in state $\mathbf{s}$, and for stochastic policies we use $\pi(i|\mathbf{s}) = p_i$ to denote that the policy π chooses action $a = i$ with probability p_i in state $\mathbf{s}$. For a stochastic policy $\pi(\cdot|\mathbf{s})$ denotes the distribution over all actions for policy π in state $\mathbf{s}$.

We provide two examples of such resource allocation problems below.

1) Single-hop Scheduling: In the single-hop scheduling problem, there are K parallel queues and a single server. Each queue has a dedicated link to the server. Packets arrive at queue $k \in [K]$ (where $[K] = \{1, 2, ..., K\}$) at time t according to a discrete independent and identically distributed (i.i.d.) arrival distribution $P(X_k)$, with X_k representing the number of arrivals for queue k. Upon arrival at queue k, packets are buffered, and the length of queue k at the beginning of time t is denoted as $q_{k,t}$.

The capacity between queue k and the server also follows a discrete i.i.d. distribution $P(Y_k)$, where Y_k represents the number of packets the server can process from queue k. The arrival and capacity distributions are assumed to be independent of each other and the network states.The state of the queueing network at time t can be represented as a set of queue-length and capacity tuples $\mathbf{s}_t = \{(q_{k,t}, y_{k,t})\}_{k\in[K]}$, with $\mathbf{s}_{k,t} = (q_{k,t}, y_{k,t})$ denoting the state of queue k at time t. At each time step, the agent observes the current network state $\mathbf{s}_t$ and chooses a queue to serve, denoted by an integer $a_t \in [K]$. The state of queue k evolves according to:

$$q_{k,t+1} = \begin{cases} [q_{k,t} - y_{k,t}]^+ + x_{k,t}, & \text{if } a_t = k \\ q_{k,t} + x_{k,t}, & \text{if } a_t \neq k \end{cases}$$

where $[z]^+ = \max(z, 0)$ and $(y_{k,t}, x_{k,t})$ are sampled i.i.d. from $(P(Y_k), P(X_k))$ respectively. This single-hop scheduling environment model is representative of networking scenarios where a shared resource must be allocated among K users, such as bandwidth in a wireless network or processor sharing.

2) Multi-path Routing: In the multipath routing problem, there are K parallel servers, each with its own independent queue. Each server's service capacity follows a distribution $P(Y_k)$, where $y_{k,t} \sim P(Y_k)$ denotes the number of packets the k-th server can process from its queue at time t. Packets exit the network immediately after being processed by their respective server.

At each time-step, a single packet x_t arrives at a routing node. Similar to the single-hop scheduling environments, the network state is represented as $\mathbf{s}_t = \{(q_{k,t}, y_{k,t})\}_{k\in[K]}$. At each time-step, a centralized controller selects a server to send

the arriving packet to, denoted by $a_t \in [K]$. The state of server k's queue evolves according to:

$$q_{k,t+1} = \begin{cases} [q_{k,t} - y_{k,t}]^+ + x_t, & \text{if } a_t = k \\ [q_{k,t} - y_{k,t}]^+, & \text{if } a_t \neq k \end{cases} \tag{2}$$

where $[z]^+ = \max(z, 0)$.

B. Switch-type Policy

For resource allocation problems where the state $\mathbf{s}$ can be factored into K components $\mathbf{s} = (\mathbf{s}_1, ...\mathbf{s}_K)$, we define the switch-type policy as follows:

Definition 2.1: A switch-type policy is a deterministic Markov policy $\pi_{sw} : \mathcal{S} \mapsto \mathcal{A}$ that adheres to the following structure: If $\pi_{sw}(\mathbf{s}) = i$, for $\mathbf{s} = (\mathbf{s}_1, ..., \mathbf{s}_i, ...\mathbf{s}_K)$ where $\mathbf{s}_k \in \mathbb{R}^N$ then for any state $\mathbf{s}'$ satisfying:

$$\mathbf{s}'_k = \begin{cases} \mathbf{s}_k + \mathbf{b}, & k = i \\ \mathbf{s}_k, & k \neq i \end{cases} \tag{3}$$

for $\mathbf{b} \in (0, \infty)^N$, it follows that $\pi_{sw}(\mathbf{s}') = i$.
For instance, consider a single hop scheduling problem with $K = 2$ queues. In a state $\mathbf{s} = (q_1, y_1, q_2, y_2)$, where q_i denotes length of queue i and y_i denotes the capacity of the link between queue i and the server, suppose that the switch type policy selects to serve queue 1, indicated as $\pi_{sw}(\mathbf{s}) = 1$. For the modified state $\mathbf{s}' = (q_1 + 1, y_1, q_2, y_2)$, where the length of queue 1 has increased from $\mathbf{s}$ while all other elements remain the same, the switch type policy would also choose to serve queue 1 i.e. $\pi_{sw}(\mathbf{s}') = 1$. Switch-type policies may also be switch-type in terms of some transformation of the state. For example, for a multipath routing problems, we will focus on policies that are switch-type in terms of the network state given as $\mathbf{s} = (-q_1, y_1, -q_2, y_2)$, meaning that if the network control agent chooses to route the incoming packet to queue 1 in state $\mathbf{s} = (-q_1, y_1, -q_2, y_2)$, it will also route the incoming packet to queue 1 in the state $\mathbf{s}' = (-(q_1 - 1), y_1, -q_2, y_2)$ where the length of queue 1 has been decreased. When the decision regions are visualized, switch-type policies have clear "switching curves" which indicate where the policy switches its decision from one action to another. An example of these switching curves is shown in Figure 3.

Switch-type policies are widely utilized in network control. For instance, MaxWeight algorithms, commonly used for scheduling problems, compute a weight $h(\mathbf{s}_k)$ for each component state $\mathbf{s}_k \in \mathbf{s}$ and select actions as $\pi_{\text{mw}}(\mathbf{s}) = \arg\max_k h(\mathbf{s}_k)$ [8], [9]. When $h(\cdot)$ is non-decreasing in $\mathbf{s}_k$, the policy is switch-type. For example, $h(\mathbf{s}_k) = q_k \times y_k$ in single-hop scheduling makes π_{mw} switch-type. Similarly, the join-the-shortest-queue (JSQ) policy is an instance of the MaxWeight policy for $h(\mathbf{s}_k) = -q_k$ in multi-path routing [10]. MaxWeight policies are often throughput-optimal, stabilizing networks without requiring knowledge of dynamics, though they may be suboptimal for cost minimization [11].

Whittle Index policies are another class of switch-type policies used in resource allocation problems [12], [13]. These heuristic policies solve Restless Multi-Armed Bandit (RMAB) problems by computing indices $I(\mathbf{s}_k)$ for each component and choosing $\pi_{\text{wi}}(\mathbf{s}) = \arg\max_k I(\mathbf{s}_k)$. If the component cost $c(\mathbf{s}_k)$ is non-decreasing, $I(\mathbf{s}_k)$ inherits this property, making the policy switch-type. However, proving indexability and deriving closed-form indices can be challenging [14], [15].

In some cases, the optimal policy itself is switch-type. For single-queue, multi-server scheduling, the delay-optimal policy is switch-type [16]–[18], and this result extends to multi-queue, multi-server MDPs [19]. Empirical evidence supports the same for N-model networks [1]. Under specific conditions, the Longest-Connected-Queue policy is delay-optimal [8]. Optimal switch-type policies also extend to other objectives, such as minimizing Age-of-Information (AoI) [7]. The success and ubiquity of switch-type policies in the literature motivated the learnable switch-type policies proposed in this work.

1) Empirical Demonstration of Switch-type Policy: To illustrate the decision regions of a switch-type policy, we use the Policy Iteration algorithm to approximate an optimal policy for a specific single-hop scheduling problem [20]. This instance features two queues ($K = 2$) with Bernoulli arrivals at $\lambda = 0.4$. Service distributions are $P(Y_k = 0) = 0.5$, $P(Y_k = 1) = 0.3$, and $P(Y_k = 2) = 0.2$ for both queues, representing a symmetric problem where arrival and service distributions are identical.

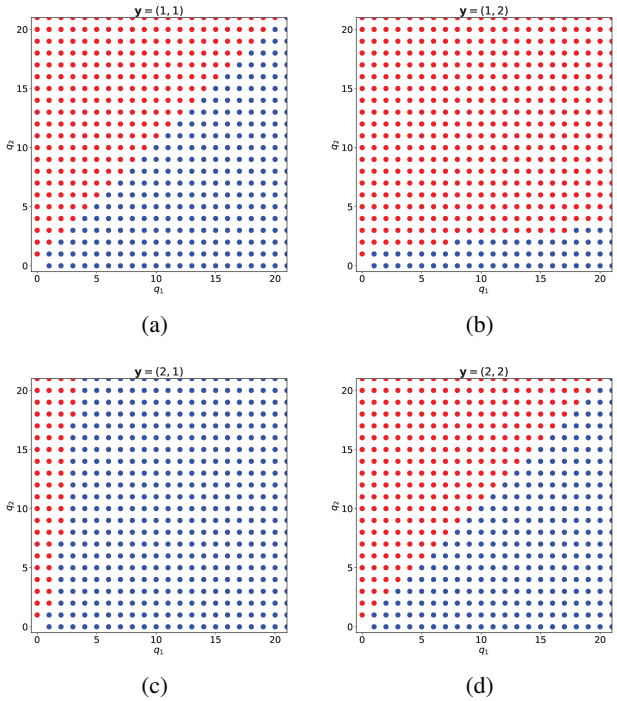
The transition function $P(\mathbf{S}_{t+1}|\mathbf{s}_t, a_t)$ was created using these dynamics. To handle the unbounded state space, we applied the sequence of approximate MDPs approach [21], solving for the optimal policy in increasingly larger finite-state approximations until convergence over $(q_1, q_2) \in [0, 20]$ and $(y_1, y_2) \in [0, 2]$.

The decision regions of the derived policy π_{PI} are shown in Figure 3. Figures 3a-3d depict decision regions for $(y_1, y_2) = (1, 1), (1, 2), (2, 1), (2, 2)$. When link states are equal, the optimal policy serves the longest queue, forming linear switching curves. For unequal service rates, the curves become nonlinear but still satisfy the conditions of a switch-type policy. This is evident from the scatter plots: if $\pi_{PI}(\mathbf{s}) = 1$ for a point, all points to the right are similarly classified, validating switch-type consistency in (q_1, q_2) and (y_1, y_2).

C. Deep Reinforcement Learning

Traditional approaches for queueing network control, such as MaxWeight and Whittle index policies, have significant limitations, including suboptimal cost minimization, restricted applicability, and computational infeasibility for large-scale networks. To overcome these issues, we employ model-free Deep Reinforcement Learning (DRL) [22], which learns policies directly from environment interactions, making it suitable for complex, large-scale problems.

Specifically, we use the Proximal Policy Optimization (PPO) algorithm [23], an actor-critic method with separate policy (π_θ) and value (V_ϕ) networks. PPO alternates between sampling data and optimizing a surrogate objective function.



(a) (b) (c) (d)

Fig. 3: Decision regions for the policy π_{PI}. Blue denotes $\pi_{PI}(\mathbf{s}) = 1$ and red denotes $\pi_{PI}(\mathbf{s}) = 2$. The x-axis corresponds to q_1 and the y-axis corresponds to q_2. Each plot corresponds to a different set of service states $\mathbf{y} = (y_1, y_2)$. Thus the set of plots represents the decision region over the truncated state-space of $(q_1, q_2) \in [0, 20] \cap (y_1, y_2) \in [1, 2]$

Its key innovation is a clipped objective, stabilizing training by preventing excessive policy updates:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathop{\mathbb{E}}_{\rho_{\pi_\theta}, \pi_\theta}\Big[\min\big(u_t(\theta)\hat{A}^{\pi_\theta}(\mathbf{s}_t, a_t), \text{clip}(u_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}^{\pi_\theta}(\mathbf{s}_t, a_t)\big)\Big]$$

where $u_t(\theta)$ is the probability ratio of new and old policies, $\hat{A}^{\pi_\theta}$ is the advantage estimate, and ϵ is a hyperparameter controlling the clipping range.

D. Multi-layer Perceptron Policy Networks

Multi-layer Perceptrons (MLP) are the standard neural network architecture used for policy networks for environments with vector states and discrete actions. Mathematically, an MLP can be represented as a sequential stack of linear transformations with non-linear activation functions. A single fully-connected layer of an MLP network can be written as

$$\mathbf{z}^{(l)} = \sigma(\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}) \tag{4}$$

where $\mathbf{W}^{(l)}$ is the weight matrix of the lth layer, $\mathbf{z}^{(l-1)}$ is the output of the previous layer, $\mathbf{b}^{(l)}$ is the bias vector of the lth layer, and $\sigma(\cdot)$ is a non-linear activation function. We use $\theta = \{(\mathbf{W}^{(l)}, \mathbf{b}^{(l)})\}_{l\in[L]}$ to refer to the sets of all parameters associated with the neural network. Due to their architecture and the universal approximation theorem, MLPs have the capability to approximate any continuous function to an arbitrary degree of accuracy, given sufficient neurons and layers [6]. This makes them well-suited for representing complex, non-linear relationships between the environment state and the desired action or value. As written above, fully connected layers are continuous functions from $\mathbb{R}^{d_{l-1}} \mapsto \mathbb{R}^{d_l}$ where d_l is the output dimension of the lth layer. Let $\mathbf{z} \in \mathbb{R}^K$ denote the output of an MLP. For DRL, we would like to convert $\mathbf{z}$ to a probability distribution. For a discrete distribution over K potential actions, a softmax layer is applied to the output of the MLP:

$$\text{SoftMax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j\in[K]}\exp(z_j)} \tag{5}$$

which converts any vector $\mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K$ into a normalized probability distribution over K elements.

III. Switch-Type Neural Network Design

This section introduces our proposed neural network architecture to be used as policy networks in DRL applications. This architecture, termed a "switch-type policy network", ensures the resulting policy is a stochastic switch-type policy which is described below.

Definition 3.1: A stochastic switch-type policy is a stochastic Markov policy $\pi_{ssw} : \mathcal{S} \mapsto \Delta(\mathcal{A})$ that adheres to the following structure: If $\pi_{ssw}(i|\mathbf{s}) = p_i$, for $\mathbf{s} = (\mathbf{s}_1, ..., \mathbf{s}_i, ...\mathbf{s}_K)$ where $\mathbf{s}_k \in \mathbb{R}^N$ then for any state $\mathbf{s}'$ satisfying:

$$\mathbf{s}'_k = \begin{cases} \mathbf{s}_k + \mathbf{b}, & k = i \\ \mathbf{s}_k, & k \neq i \end{cases} \tag{6}$$

for $\mathbf{b} \in (0, \infty)^N$, it follows that $\pi_{ssw}(i|\mathbf{s}') \geq p_i$.
In other words, if we increase any element of $\mathbf{s}_i$, the probability of choosing action $a = i$ increases under a stochastic switch-type policy. Stochastic policies are favored over deterministic policies for on-policy DRL algorithms, as stochastic policies ensure exploration of the state and action spaces which is crucial for learning an optimal policy.

To represent stochastic switch-type policies using neural networks, we utilize stacked monotonic hidden layers which are described in the next subsection. We formally define monotonicity with respect to multivariate functions first:

Definition 3.2: Let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a continuous differentiable multivariate function. The function $f(\mathbf{x})$ is said to be monotonically non-decreasing on $\mathbf{x}$, if $\forall i, x_i \leq x'_i, \forall j \neq i, x_j = x'_j \Rightarrow f(\mathbf{x}) \leq f(\mathbf{x}')$ holds for any two points $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^n$.

Our policy network architecture is based on the following lemma:

Lemma 1: For an MDP with state-space $\mathcal{S} = \mathcal{S}_1 \times ... \times \mathcal{S}_K$, let the function $\mathbf{f} : \mathcal{S} \mapsto \mathbb{R}^K$ be decomposable such that $\mathbf{f}(\mathbf{s}) = (f(\mathbf{s}_1), ..., f(\mathbf{s}_K))$ for some function f. If for all $k \in [K]$, $f : \mathcal{S}_k \mapsto \mathbb{R}$ is monotonically non-decreasing in all elements of $\mathcal{S}_k$, then the policy:

$$\pi_{\mathbf{f}}(\cdot|\mathbf{s}) = \text{SoftMax} f(\mathbf{s}) \tag{7}$$

is a stochastic switch-type policy.
Similarly, if softmax is replaced with an argmax, we obtain a deterministic switch-type policy.

A. Switch-Type Policy Architecture

Our proposed architecture leverages two key components:

1) **Monotonic Hidden Layers**: These hidden layers guarantee the resulting model is a monotonic function. This is achieved through a combination of exponentiated weights and a ReLU-N activation function.
2) **Input Vector Transformation:** This component enables faster model evaluation during forward passes.

We delve into each component in the following subsections:

1) Monotonic Hidden Layers: Our switch-type policy network is comprised of fully-connected hidden layers that enforce monotonicity. These monotonic hidden layers utilize an exponentiated weight unit and Relu-N activation function, and this combination was first proposed for universal monotonic function approximation in [24]. Exponentiated weight units perform the following linear operation

$$\exp(\mathbf{W}^{(l)})\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ is the weight matrix and bias vector of the lth layer, and $\mathbf{z}^{(l-1)}$ is the output of the previous layer. Since $\exp(\mathbf{W}^{(l)})$ is guaranteed to be positive, this linear operation is monotonically non-decreasing in $\mathbf{z}^{(l)}$. The output of this linear operation is then passed through a ReLU-N activation function $\sigma_{RN} : \mathbb{R}^d \mapsto [0, N]^d$, defined as [25]:

$$\sigma_{RN}(\mathbf{z})_i = \begin{cases} N, & \text{if } z_i > N, z_i \to N^+ \\ x, & \text{if } 0 < z_i < N, z_i \to N^-, z_i \to 0^+ \\ 0, & \text{if } z_i < 0, z_i \to 0^- \end{cases} \quad (8)$$

This combination guarantees the output of the lth layer:

$$\mathbf{z}^{(l)} = \sigma_{RN}(\exp(\mathbf{W}^{(l)}))\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}) \quad (9)$$

is monotonically non-decreasing with respect to the previous layer's output $\mathbf{z}^{(l-1)}$. The ReLU-N activation function is used instead of the commonly used ReLU activation function, as a ReLU network with all positive weights can only model a convex function [26]. Meanwhile, a fully-connected ReLU-N network with at least four hidden layers is a universal function approximator for purely monotonic functions [24]. Much like an MLP architecture, these monotonic hidden layers can be stacked to create a monotonic deep neural network. For an L layer network, the final output layer is comprised of an exponentiated weight unit with a weight vector $\mathbf{w}_L$ and scalar bias b_L without a final activation function, e.g.,

$$\mathbf{z}^{(L)} = \exp(\mathbf{W}^{(L)})\mathbf{z}^{(L-1)} + \mathbf{b}^{(L)} \quad (10)$$

This ensures the model formed by the stacked layers

$$f_\theta(\mathbf{z}^{(0)}) = \exp(\mathbf{W}^{(L)})\sigma_{RN}(\exp(\mathbf{W}^{(l-1)})^\intercal \cdots)) + b^{(L)}))$$

where $\mathbf{z}^{(0)} \in \mathbb{R}^n$ is the input vector to the model, is a multivariate monotonic function $f_\theta : \mathbb{R}^n \mapsto \mathbb{R}$.

2) Input Transformation: The monotonic neural network f_θ takes the place of the monotonic function f in Lemma 1. Unlike standard policy networks, f_θ is not applied to the entire state vector $\mathbf{s}$, but is instead applied to each component state $\mathbf{s}_k \in \mathbf{s}$ individually. This operation can be performed in parallel by a simple reshaping operation that takes observation $\mathbf{s} \in \mathbb{R}^{K \times n}$ and converts it to a matrix $\mathbf{S} \in \mathbb{R}^{(K,n)}$ where the kth row is $\mathbf{s}_k$. Passing $\mathbf{S}$ through the monotonic neural network then produces a vector $\mathbf{z}^L \in \mathbb{R}^K$ where each component z_k^L is monotonically non-decreasing in $\mathbf{s}_k$.

3) Output Layer: We add a softmax layer on top of the final monotonic hidden layer to produce the switch-type policy network. This resulting policy architecture ensures that for any weights $\theta = \{(\mathbf{W}^{(l)}, \mathbf{b}^{(l)})\}_{l \in [L]}$, the resulting policy π_θ is a stochastic switch type policy.

IV. Single Environment Training Comparison

We start by comparing STN and MLP policy networks in terms of sample efficiency during training on single-hop scheduling and multihop routing environments.

A. Training Environments

First we provide a brief overview of the environment sampling procedures for the single-hop scheduling and multi-path routing environments; detailed descriptions are available in the accompanying technical report [27]. For single-hop scheduling, five environments with $K = 4$ queues were generated. Arrivals to each queue and service capacities for each link followed Poisson distributions, with rate parameters sampled uniformly from $(0, 3)$. Stability was ensured by running the MaxWeight policy on each environment. For multi-path routing, five environments with $K = 8$ queues were generated using a similar procedure. Only the service rates were sampled uniformly from $(0, 1)$, as the arrival process was deterministic. Stability was verified using the Shortest Queue policy.

B. Training and Evaluation Procedure

Separate STN ($\pi_{\theta,STN}$) and MLP ($\pi_{\theta,MLP}$) policies were trained on each of the 10 environments using the PPO algorithm. The training hyperparameters and an overview on how they were chosen is available in the accompanying technical report [27]. Each policy was trained for $T_{\text{train}} = 1,000,000$ environment steps. Following training, policies were evaluated on their respective environments over three trajectories, each lasting $T = 50,000$ steps. The empirical time-average cost for a trained policy π_θ on environment $\mathcal{E}$ was computed as $J(\pi_\theta, \mathcal{E}) = \frac{1}{T}\sum_{t=0}^{T-1} c(\mathbf{s}_t)$. Results presented in subsequent sections are based on these evaluations.

C. Single Environment Training Results

We present results from the single environment training experiments. Since the optimal average cost $\min_\pi J(\pi, \mathcal{E})$ varies across environments, performance is compared using the normalized average cost:

$$J_0(\pi_\theta, \mathcal{E}) = \frac{J(\pi_\theta, \mathcal{E})}{J(\pi_0, \mathcal{E})}, \quad (11)$$

where π_0 is the MaxWeight policy for single-hop scheduling or the Shortest Queue policy for multi-path routing environments.

Sample efficiency during training is evaluated using a $T_{MA} = 5,000$ step moving average of the cost. The natural logarithm of this moving average is plotted in Figure 4 for both problem classes, showing that STN policies require fewer samples to outperform π_0 compared to MLPs. Only the first $500,000$ training steps are shown in Figure 4 as its evident that performance converged far before the total training horizon T_{train} for all policies. These results demonstrate that an STN policy network requires fewer samples from the environment to produce a policy that outperforms π_0 compared to the MLP policy architecture. Additionally, even before training, the STN policy architecture often results in a policy that outperforms the initial policy of the MLP policy network. This difference in initial policy performance is very apparent for multi-path routing environments as seen by the plots in the right column of Figure 4. In fact, the initial STN policy only slightly improves during training for the multi-path environments.

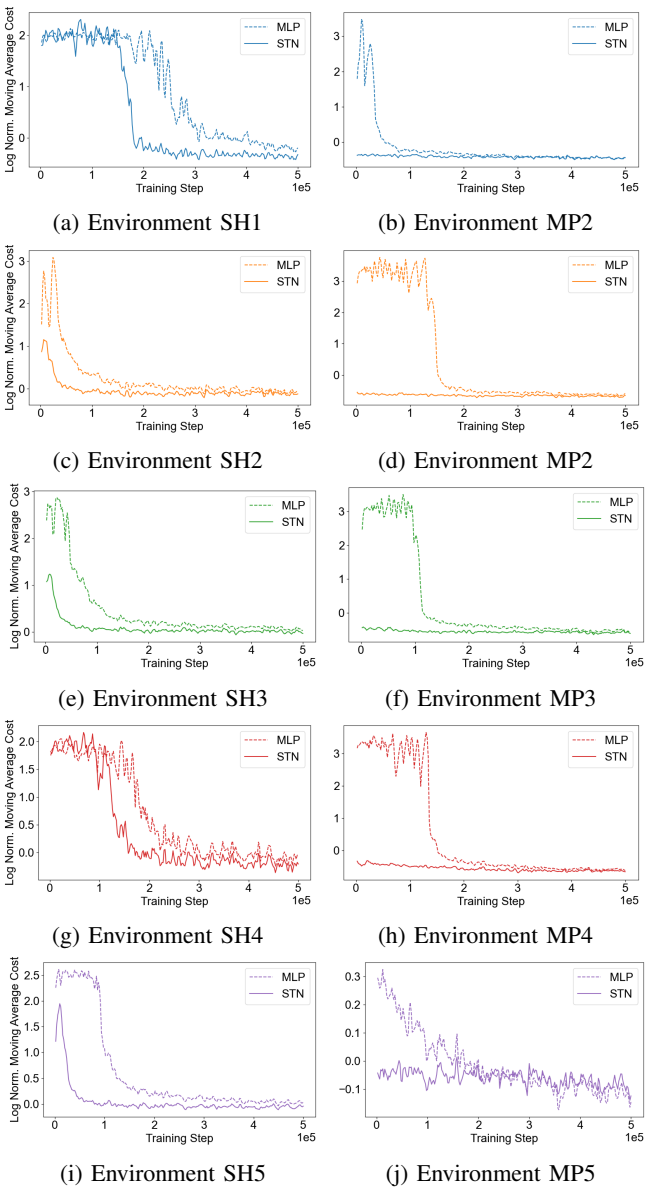
We also evaluate the final average costs of trained policies on their respective environments, shown in Figure 5. Results indicate that while MLPs optimize over a broader policy class, they converge to similar performance as STNs, suggesting that the optimal policies for these resource allocation problems are switch-type.

V. Generalization

Next, we study the ability of the STN and MLP policy networks to generalize to unseen environments. The following experimental procedure was repeated for both the single-hop scheduling and multipath routing problem classes. First, an environment set $\boldsymbol{\mathcal{E}}$ containing 100 different environments was generated using the parameter sampling procedures detailed in section IV-A. This environment set $\boldsymbol{\mathcal{E}}$ was then split into a training set $\boldsymbol{\mathcal{E}}_{train}$ which contained five environments, and $\boldsymbol{\mathcal{E}}_{test}$ which contained the remaining ninety-five environments from $\boldsymbol{\mathcal{E}}$. A single policy network was trained for each policy network architecture on all environments in $\boldsymbol{\mathcal{E}}_{train}$, and this single policy network was then tested on all environments $\boldsymbol{\mathcal{E}}$. The performance at test time on environments in $\boldsymbol{\mathcal{E}}_{train}$ characterize how well the policy network learned to perform in environments it saw during training, while the performance on environments in $\boldsymbol{\mathcal{E}}_{test}$ characterize a policy network's ability to generalize to unseen environments.

A. Training and Evaluation Procedure

In the previous single-environment training procedure, we trained a single policy on each environment in the training set. For the current study, we trained a single policy π_θ on all environments in the training set $\boldsymbol{\mathcal{E}}$ simultaneously. The training process is only slightly modified from the single-environment procedure: trajectories of length $T_{eps} = 2,000$ are generated using π_θ in each environment in parallel, resulting in an update step with a batch size of 10,000 for each update. The total number of training steps collected during the training phase is 10,000,000, meaning that 2,000,000 training steps



(a) Environment SH1 (b) Environment MP2

(c) Environment SH2 (d) Environment MP2

(e) Environment SH3 (f) Environment MP3

(g) Environment SH4 (h) Environment MP4

(i) Environment SH5 (j) Environment MP5

Fig. 4: Moving average cost of the training policy versus training step. The left column corresponds to the single-hop scheduling training environments and the right column corresponds to the multi-path routing training environments.

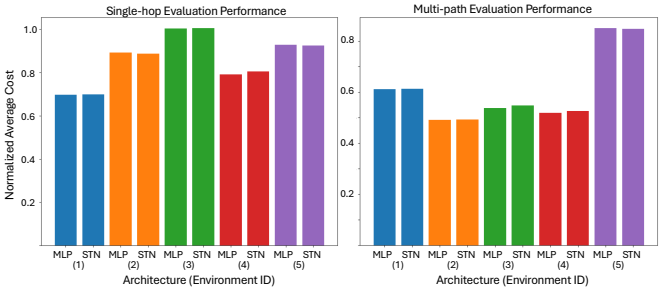


Fig. 5: Average cost of the trained policies in the same environments they were trained in.

are collected from each training environment over the entire training process. After training, we evaluated the trained policy on all environments in $\boldsymbol{\mathcal{E}}$.

B. Observation Encoding

In the single-environment study, environment parameters were not required for policy learning. However, for generalization, the agent must observe both the network state $\mathbf{s}_t$ and the environment parameters. Without this, the environment would be partially observable, as the agent could not identify which environment it operates in.

For this study, the agent observes $(q_{k,t}, y_{k,t})$ for each queue k, along with λ_k and μ_k for single-hop environments, and μ_k for multi-path environments. Due to the monotonic nature of STN networks, the sign of observation vector elements impacts performance. Training across five environments revealed that the encoding $\mathbf{o}_{k,t} = (q_{k,t}, y_{k,t}, \lambda_k, -\mu_k)$ performed best for single-hop environments, while $\mathbf{o}_{k,t} = (-q_k, y_k, \mu_k)$ was optimal for multi-path environments. These encodings were used for both policy and value networks in this study.

C. Generalization Performance

We evaluate both policy network architectures on all environments in $\boldsymbol{\mathcal{E}}$ and report normalized average costs, focusing on training and testing sets. The average normalized cost over environment set $\boldsymbol{\mathcal{E}}$ is denoted as $J_0(\pi_\theta, \boldsymbol{\mathcal{E}})$. For the MLP network, some environments yielded very large average cost, indicating the policy failed to stabilize the queuing network and costs grew unbounded. To account for this, we also report an outlier-rejected average, $J_0(\pi_{\theta,MLP}, \tilde{\boldsymbol{\mathcal{E}}})$, excluding environments with $J(\pi_{\theta,MLP}, \mathcal{E}) > 5$. These metrics are summarized in Tables I and II, with histograms of costs shown in Figure 6.

Results show that the STN network consistently outperforms the MLP network across all metrics and environments. The performance gap is most significant in single-hop environments, where STN achieves up to $8\times$ and $38.7\times$ better performance in training and testing, respectively. Even with outlier rejection, STN outperforms MLP in all cases, with MLP failing to stabilize 20% of the single-hop environments. In multi-path environments, the performance gap is smaller but still favors STN across all metrics. On average, STN also surpasses the MaxWeight and Shortest Queue baseline policies (π_0).

D. Multi-Environment Training Sample Efficiency

We also compare the training sample efficiency of each architecture. As in the single environment training procedure, we track the $T_{MA} = 5,000$ step moving average cost during training for each environment $\mathcal{E}^{(j)} \in \boldsymbol{\mathcal{E}}_{train}$, and normalize this metric by $J(\pi_0, \mathcal{E}^{(j)})$. We then take an average of this normalized moving average cost across all training environments. We plot these training curves in Figure 7 using the logarithm of the mean normalized cost on the y-axis, as the magnitude of this metric varied widely during training especially for the MLP policy. As seen for the case when

$J_0(\pi_\theta, \boldsymbol{\mathcal{E}})$	MLP		STN	
$\boldsymbol{\mathcal{E}}_{train}$	6.90	(13.39)	0.860	(0.049)
$\tilde{\boldsymbol{\mathcal{E}}}_{train}$	0.907	(0.521)	0.860	(0.049)
$\boldsymbol{\mathcal{E}}_{test}$	33.7	(94.1)	0.870	(0.128)
$\tilde{\boldsymbol{\mathcal{E}}}_{test}$	1.20	(0.521)	0.870	(0.128)

TABLE I: Mean and standard deviation (in parenthesis) of the normalized average cost during evaluation for both architectures across each set of single-hop environments. For the outlier rejected metrics for the MLP policy network, 1 context was omitted from $\tilde{\boldsymbol{\mathcal{E}}}_{train}$ and 19 contexts were omitted from $\tilde{\boldsymbol{\mathcal{E}}}_{test}$.

$\hat{J}(\pi_\theta, \boldsymbol{\mathcal{E}})$	MLP		STN	
$\boldsymbol{\mathcal{E}}_{train}$	0.571	(0.162)	0.550	(0.159)
$\boldsymbol{\mathcal{E}}_{test}$	1.03	(0.956)	0.625	(0.140)
$\tilde{\boldsymbol{\mathcal{E}}}_{test}$	0.938	(0.274)	0.625	(0.140)

TABLE II: Mean and standard deviation (in parenthesis) of the normalized average cost during evaluation for both architectures across each set of multi-path environments. For the outlier rejected metrics for the MLP policy network, 1 context was omitted from $\tilde{\boldsymbol{\mathcal{E}}}_{test}$.

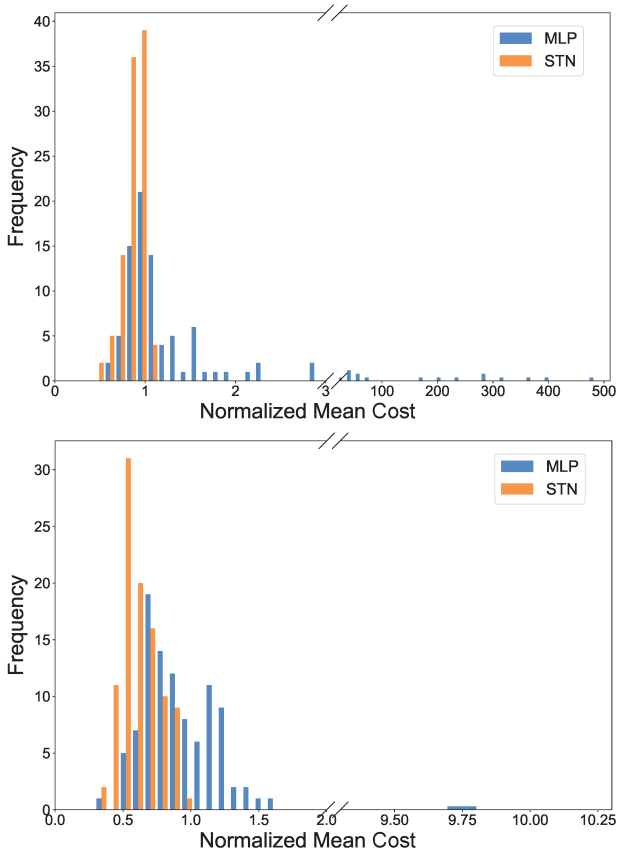


Fig. 6: Histogram of $\hat{J}_0(\pi_\theta, \mathcal{E}^{(j)})$ for the test environments $\mathcal{E}^{(j)} \in \boldsymbol{\mathcal{E}}_{test}$ for the single-hop scheduling environments (top) and multi-path routing environments (bottom) . Note the "break" on the x-axis halfway through the plot, where the scaling changes, where the right half of the plot shows the outliers for each test set.

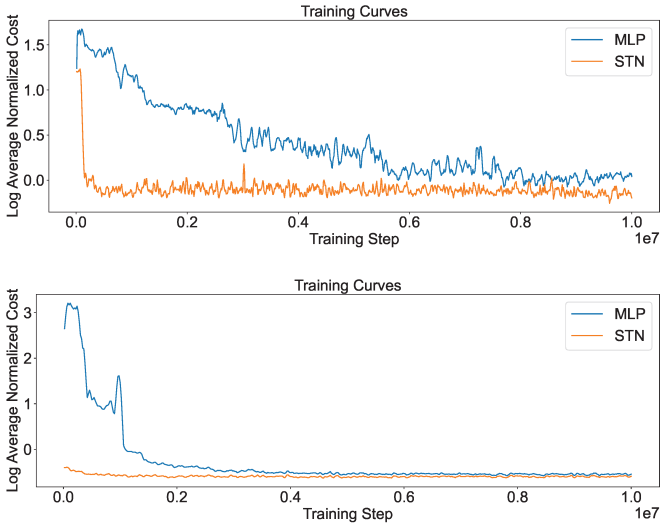


Fig. 7: Moving average cost of the training policy averaged across all five training environments vs the training step. Top plot corresponds with the single-hop training environments and the bottom corresponds with the multi-path training environments

a single policy network is trained on a single environment, the STN policy network is significantly more sample efficient than the MLP policy network when training on multiple environments simultaneously.

VI. Conclusion

In this work, we introduced the switch-type policy network (STN), a novel neural network architecture inspired by the long-history of switch-type resource allocation control policies. The STN enhances the sample efficiency of the PPO algorithm without compromising performance, demonstrating its suitability for the studied problem classes. Unlike MLPs, which often overfit to training environments, the STN generalizes effectively to unseen environments while matching MLP performance in familiar settings. These results suggest that optimal policies are switch-type with respect to both queue sizes and arrival/service rates. Future work will explore the broader optimality of the switch-type policy class across diverse environments.

References

[1] J. G. Dai and M. Gluzman, "Queueing Network Controls via Deep Reinforcement Learning," *Stochastic Systems*, vol. 12, no. 1, pp. 30–67, mar 2022, publisher: INFORMS.

[2] U. Ayesta, "Reinforcement learning in queues," *Queueing Systems*, vol. 100, no. 3, pp. 497–499, apr 2022.

[3] M. Raeis, A. Tizghadam, and A. Leon-Garcia, "Queue-Learning: A Reinforcement Learning Approach for Providing Quality of Service," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 1, pp. 461–468, may 2021, number: 1.

[4] H. Fawaz, D. Zeghlache, T. A. Pham, J. Leguay, and P. Medagliani, "Deep Reinforcement Learning for Smart Queue Management," *Electronic Communications of the EASST*, vol. 80, sep 2021.

[5] J. Wigmore, B. Shrader, and E. Modiano, "Intervention-Assisted Policy Gradient Methods for Online Stochastic Queuing Network Optimization: Technical Report," apr 2024, arXiv:2404.04106 [cs].

[6] A. Pinkus, "Approximation theory of the MLP model in neural networks," *Acta Numerica*, vol. 8, pp. 143–195, jan 1999.

[7] Y.-P. Hsu, E. Modiano, and L. Duan, "Scheduling Algorithms for Minimizing Age of Information in Wireless Broadcast Networks with Random Arrivals: The No-Buffer Case," sep 2018, arXiv:1712.07419 [cs, math].

[8] L. Tassiulas and A. Ephremides, "Dynamic server allocation to parallel queues with randomly varying connectivity," *IEEE Transactions on Information Theory*, vol. 39, no. 2, pp. 466–478, mar 1993, conference Name: IEEE Transactions on Information Theory.

[9] S. Meyn, "Stability and Asymptotic Optimality of Generalized MaxWeight Policies," *SIAM Journal on Control and Optimization*, vol. 47, no. 6, pp. 3259–3294, jan 2009, publisher: Society for Industrial and Applied Mathematics.

[10] H.-C. Lin and C. Raghavendra, "An analysis of the join the shortest queue (JSQ) policy," in *[1992] Proceedings of the 12th International Conference on Distributed Computing Systems*, jun 1992, pp. 362–366.

[11] M. Neely, "Stochastic Network Optimization with Application to Communication and Queueing Systems," *Synthesis Lectures on Communication Networks*, vol. 3, no. 1, pp. 1–211, 2010.

[12] P. Whittle, "Restless bandits: activity allocation in a changing world," *Journal of Applied Probability*, vol. 25, no. A, pp. 287–298, jan 1988.

[13] M. O'Keeffe, P. S. Ansell, K. D. Glazebrook, and J. Nio-Mora, "Whittle's index policy for a multi-class queueing system with convex holding costs," *Mathematical Methods of Operations Research (ZOR)*, vol. 57, no. 1, pp. 21–39, apr 2003.

[14] K. Liu and Q. Zhao, "Indexability of Restless Bandit Problems and Optimality of Whittle Index for Dynamic Multichannel Access," *IEEE Transactions on Information Theory*, vol. 56, no. 11, pp. 5547–5567, nov 2010, conference Name: IEEE Transactions on Information Theory.

[15] U. Ayesta, M. K. Gupta, and I. M. Verloop, "A unifying computations of Whittle's Index for Markovian bandits," *arXiv: Optimization and Control*, jun 2019.

[16] W. Lin and P. Kumar, "Optimal control of a queueing system with two heterogeneous servers," *IEEE Transactions on Automatic Control*, vol. 29, no. 8, pp. 696–703, aug 1984, conference Name: IEEE Transactions on Automatic Control.

[17] G. Koole, "A simple proof of the optimality of a threshold policy in a two-server queueing system," *Systems & Control Letters*, vol. 26, no. 5, pp. 301–303, dec 1995.

[18] I. Viniotis and A. Ephremides, "Extension of the optimality of the threshold policy in heterogeneous multiserver queueing systems," *IEEE Transactions on Automatic Control*, vol. 33, no. 1, pp. 104–109, jan 1988, conference Name: IEEE Transactions on Automatic Control.

[19] V. Rykov, "Monotone Control of Queueing Systems with Heterogeneous Servers," *Queueing Systems*, vol. 37, no. 4, pp. 391–403, mar 2001.

[20] D. P. Bertsekas, *Dynamic Programming and Optimal Control, Vol 1*.

[21] L. I. Sennott, "Average Cost Optimal Stationary Policies in Infinite State Markov Decision Processes with Unbounded Costs," *Operations Research*, vol. 37, no. 4, pp. 626–633, 1989, publisher: INFORMS.

[22] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, second edition ed., ser. Adaptive computation and machine learning series. Cambridge, Massachusetts: The MIT Press, 2018.

[23] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," aug 2017, arXiv:1707.06347 [cs].

[24] H. Kim and J.-S. Lee, "Scalable Monotonic Neural Networks," oct 2023.

[25] S. S. Liew, M. Khalil-Hani, and R. Bakhteri, "Bounded activation functions for enhanced training stability of deep neural networks on visual pattern recognition problems," *Neurocomputing*, vol. 216, pp. 718–734, dec 2016.

[26] X. Liu, X. Han, N. Zhang, and Q. Liu, "Certified Monotonic Neural Networks," dec 2022, arXiv:2011.10219 [cs].

[27] J. Wigmore, B. Shrader, and E. Modiano, "A Novel Switch-Type Policy Network for Resource Allocation Problems: Technical Report," Jan. 2025.