National College of Ireland

Cost and Carbon Aware Reinforcement Learning Autoscaling for Multi-Cloud Kubernetes Spot Instances

MSc Research Project
Cloud Computing

Vikrant Anil Ghode
Student ID: x23324180

School of Computing
National College of Ireland

Supervisor: Prof. Shaguna Gupta

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Vikrant Anil Ghode
Student ID:	x23324180
Programme:	Cloud Computing
Year:	2025
Module:	MSc Research Project
Supervisor:	Prof. Shaguna Gupta
Submission Due Date:	11/12/2025
Project Title:	Cost and Carbon Aware Reinforcement Learning Autoscaling for Multi-Cloud Kubernetes Spot Instances
Word Count:	XXX
Page Count:	25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Vikrant Anil Ghode
Date:	28th January 2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Cost and Carbon Aware Reinforcement Learning Autoscaling for Multi-Cloud Kubernetes Spot Instances

Vikrant Anil Ghode
x23324180

Abstract

In a few public clouds, cloud-native microservices typically operate on managed Kubernetes. Elasticating the resources assists them in utilizing better, nevertheless, the cost of computation and emissions caused by the operations remains significant. Free but cancelable (such as Spot or Preemptible instances) are much cheaper yet may be terminated at any time and will have different effects on emissions depending on that area, so delivery is more difficult. This paper presents an autoscaler which employs reinforcement learning to balance these three variables: cost, lifecycle carbon, and service reliability, through switching among clouds, regions, purchasing choices, and timing.

The autoscaler monitors real-time demand, far tail-latency indicators, revocation risk, and predictions of carbon intensity and then decides what and how many machines to execute. It is benchmarked with genetic algorithms and threshold on typical workloads of microservices. The point is that a multi-objective RL policy may reduce costs and emission levels and retain additional SLO infractions within a low cap that provides an open and reusable strategy that users can apply to FinOps and greener cloud computing.

1 Introduction

Resource management that is carbon conscious has become a significant method of minimizing the environmental impact of cloud computing. Nevertheless, current carbon-conscious schedulers are mainly concerned with batch workloads or delay-tolerant workloads and relaxed execution constraints. These strategies do not work with latency-sensitive microservices, which are required to maintain continuous austere tail-latency service-level objectives (SLOs) and cannot support unstable capacity or deferred execution.

Simultaneously, with the popularity of Spot instances, there is a considerable amount of cost-saving and operational risk associated with unexpected revocations. Current autoscalers are usually optimised to either use or cost only and are not explicitly optimised to deal with the combination of carbon intensity, interruption hazard, and latency guarantees. This is driven by the requirement to collectively maximize carbon efficiency, cost, and reliability on production microservices and suggests a carbon-conscious autoscaling system tailored specifically to those parameters.

1.1 Background and Motivation

This research is motivated by three pressures.

Cost pressure. Autoscaling is very quick to add extra capacity when the load spikes; unless you manage it in terms of price, the costs increase.

Environmental pressure. The place and time of running things differs from the quantity of carbon utilized and thus workloads must be placed in mind of carbon.

Methodological opportunity. Learning based controllers, particularly reinforcement learning, can deal with dynamic situations and produce more signals by combining various ones than static rules.

This paper provides a new plan that manages three things at once, how long it takes for microservices that need quick responses to work, how much it costs to run them, and how dependable they are. The design is cloud independent and has dependencies on the resources that can be revoked.

1.2 Problem Statement

Latency sensitive microservices are becoming more widely deployed on Kubernetes by various public clouds. Cloud providers have shorter term revocable capacity (e.g. Spot/Preemptible) which can be much more cost effective, at the cost of irregular pre-emptions, but placement and timing can have a significant impact on emissions, as the carbon intensity of the grid differs by location and time of day. The default autoscalers are optimised in terms of utilisation and simple performance indicators without modelling price, revocation risk, and carbon. Carbon aware scheduling can be done in Kubernetes. It is usually working on one cluster, on one batch, or if we are allowed to relax the service level objectives (Hallberg; 2024), (Piontek et al.; 2023).

Temporal (and emerging spatial) carbon-aware shifting has demonstrable benefits in reducing emissions to flexible workloads at datacentre scale (Radovanovic et al. 2021/2023), but at this scale, always on services with SLOs and interruption risk tightly coupled are hardly ever considered with these approaches. Learning based autoscaling can be shown to be better than threshold heuristics in microservices using parallel work (Bai et al. 2024) but does not explicitly hedge carbon or revocation (generally because it is performance/cost optimisation). At last, clouds/region hedging systems with Spot capacity can ensure model serving with replicas and falling back to ondemand (Mao et al. 2024/2025), but are specialised and not a general, open autoscaler of arbitrary microservices. Thus, the unresolved issue is to develop and empirically test an open and multicloud Kubernetes autoscaler that collaboratively optimises cost and carbon lifecycle and satisfies latency SLOs in realistic pre-emption dynamics.

The controller has to consider signals that vary with time, e.g. workload demand, acceptable slowdown, opportunities for cancellation, cost and carbon intensity. It must also change from one cloud, region or purchase option to another. In production it has to be reliable even with API limits and delay in measurement. By making it a success, three independent bodies of work would be connected, including carbon-aware orchestration, learning-based autoscaling, and reliability conscious Spot adoption, into a single, reproducible framework.

1.3 Research Questions

The work is guided by the following research questions (RQs).

Primary RQ. Can a cost and carbon-conscious reinforcement-learning autoscaler reduce the total infrastructure cost and lifecycle carbon by a large margin when compared to strong baselines, and keep latency-SLO violations within a small and pre-specialized tolerance, when realistic interruption rates are used?

RQ1. How do spatiotemporal carbon signals shape placement and hedging behaviour? Spatio-Temporal carbon guides the policy to operate in cleaner areas/hours where there is SLO headroom, hedge with on-demand floors, prefer low-risk Spot, and operation based on hysteresis to avoid flapping quantified using per-hour CO2, SLO-breach deltas and cross-cloud diversity.

RQ2. What does the cost carbon reliability Pareto frontier look like versus strong baselines? The policy of RL is superior to tuned HPA+CA and GA baselines at the same iso-reliability, and its slope of trade-off between the percentages of gCO2 and cost is superior at iso-reliability following the adjustment of Spot share.

RQ3. How sensitive are results to workload mix and latency budgets? Sensitivity indicates that narrower P95/P99 goals and denser service graphs reduce the time/space window and reduce gains, but that tighter budgets and burst-tolerant queues allow greater cost/CO2 savings, with the elasticity encapsulated by a fixed SLO envelope.

RQ4. Which simtoreal practices stabilise behaviour under API limits, telemetry lag, and network jitter?Domain randomization provides stable sim-to-real behaviour, as does action clipping and change-rate caps, API budgeters (backoff+jitter), stale-data fallbacks, anti-correlated replica spreading, and shadow-guarded rollout with optional on-cluster fine-tuning, demonstrated by a reduced actions/hour, churn, and recovery time.

1.4 Proposed Solution

It is a many goal, layered autoscaler, which favors a reinforcement learning policy. Each time it decides, it determines which cloud service provider, the cloud region, the type of instance, and the purchase plan, and even the size of the node pool. It gives a combination of information: the busyness of the system, the slowest tasks, notifications of resource appropriations, the existing prices, and the amount of carbon in the respective regions.

The reward promotes low cost and low carbon, but also low cost and low carbon rewards breaking the service level agreement. Operators are free to decide the relative importance of the various goals. A simulator with actual past data allows the policy to be trained safely and then deployed; once it is trained it is then deployed into a Kubernetes controller which communicates to cluster autoscalers and control planes.

The RL autoscaler is compared to a genetic algorithm and a threshold baseline on microservice tests of ecommerce and social media applications; studies are also conducted by dropping the carbon part or disregarding the multicloud trading to understand the effect of each part.

1.5 Research Objectives

The research objectives are:

- Develop a tracedriven simulator to evaluate revocation and carbon usage.
- Develop a reinforcement learning controller that can allow you to set tradeoff weights and include safe fallback logic.

- Establish a multi-cloud Kubernetes testbed with realistic microservices and traffic generation.
- Measure performance based on cost, lifecycle carbon, SLO-breach rate and interruption recovery time.
- Release the code, traces, and configurations to support reproducibility and future extension.

1.6 Limitations and Challenges

There are several challenges anticipated. Cross zone or region revocation events are possible, and this makes it less efficient and riskier towards the service level objectives. SSignals of carbon intensity may be lagging or changing; monitoring the progression of data through its entire life may make things uncertain. Trying to take lessons from simulations and apply them to the real world can go wrong because of delays in controlling, noisy data, and waiting because of limitations. The number of live trails that can be conducted may be limited by budget and quotas constraints. It may be necessary to conduct additional tests to be able to generalize outside of the selected benchmarks. These issues are covered in the study by means of ablations, uncertainty capable analyses and emphasis on transparent and reproducible artifacts.

1.7 Report Structure

The rest of this report is structured in the following way. Section 2 reviews existing literature. on carbon conscious scheduling, autoscaling reinforcement, multi-cloud Spot. Multi-objective optimisation and optimisation in cloud control. Section 3 describes the methodology, such as data, the trace-driven simulator, and RL formulation. The system architecture and design specifications are presented in Section 4. Section 5 details the deployment of Kubernetes operator, toolchain and engineering protection. Section 6 indicates the evaluation metrics, datasets and experimental scenarios. Section 7 presents the experimental findings, simulator and real-world. Finally, Section 8 concludes with the contributions, limitations and discusses directions.

2 Literature Review

This review examines literature in the field of carbon-conscious scheduling, autoscaling, multi-cloud orchestration, and RL to resource management, and summarizes available gaps.

2.1 Carbon-Aware Scheduling in Kubernetes and Clouds

Carbon-conscious orchestration Carbon-conscious orchestration has come out of datacentre-scale shifting to Kubernetes-native schedulers. Hallberg demonstrates that workloads can be directed to less-carbon areas/times and reduction of emissions can be achieved by adding real-time and anticipated carbon-intensity indicators to K8s, but latency-sensitive SLAs are difficult to achieve at small scale (Hallberg; 2024). Google has a system named Carbon-Intelligent Computing that proves the ability to support temporal (and emerging

spatial) advancing with day-ahead forecasts without violating the overall service targets at hyperscale (Radovanovic et al.; 2021). Caspian presents a multi-cluster scheduler that is a QoS-aware and takes into account carbon signals as well as performance goals in a federation Bahreini et al. (2024). In addition to platforms, empirical research results in up to with inter-datacentre job shifting of a maximum of an emissions reduction of up to 18% with a moderate latency effect (Ren et al.; 2023). Earlier power-sensitive scheduling conceptualized utility trade-off based on dynamic pricing and fixed carbon variables, and formed the conceptual basis of cost-carbon balancing (Feng and Buyya; 2016). Taken together, all these papers indicate real-world carbon-conscious control, but the majority of them apply to batch/deferrable workloads or individual clusters instead of multi-cloud or latency-sensitive microservices (Hallberg; 2024);(Radovanovic et al.; 2021);(Bahreini et al.; 2024);(Ren et al.; 2023);(Feng and Buyya; 2016)

2.2 Reinforcement Learning for Autoscaling and Resource Management

Under the variation in workload RL always performs better than threshold heuristics in container orchestration. In comparison to fixed policies, Deep Q-Learning in K8s can reduce over-provisioning and handle non-stationary demand more effectively than it can anticipate demand fluctuations (Zhang et al.; 2022). Inter-service dependencies, as well as response-time signals, are used in RL-Auto to enhance the compliance of SLA with microservices (Xu et al.; 2023). Uncertainty about revocations is also managed by RL proactive migration controlled by a particular agent minimizes the impact of interruption in a single-cloud scenario (Chen et al.; 2021). Hierarchical RL helps to boost the convergence rate by breaking down the decisions (instance type, region, scale triggers) down and multi-agent designs decentralise the control between services or node pools to enhance scalability and robustness (Muller et al.; 2022). Previous analyses of DDPG/A3C as a web scaling technique indicate the superiority or RL over thresholds in conditions of high-variance load (Ma et al.; 2023). Nonetheless, the latency/cost alone optimisation is used in most works and does not include carbon signal or multi-cloud revocation hedging (Lorido-Botran and Bhatti; 2021).

2.3 Multi-Cloud Spot Instance Optimisation and Reliability

The field of work on spot markets investigates cost savings and resilience in preemption, but seldom incorporates environmental signals or RL control (Patra et al.; 2024). SpotKube identifies cost-efficient spot configurations in K8s, but is not carbon aware and cross-cloud federated in real time (Chohan et al.; 2019). Volatility is reduced by classic MapReduce studies using checkpointing/restart policies, but at the cost of cloud-specific assumptions, such as warm-standby strategies to enhance MTTR to microservices and classic MapReduce studies employing them as well (Li and Kannan; 2021). Pricing models formulate selection as a MDP to optimize bidding, given risk tolerance, although in single-region application, usually, it is a MDP that has a single region (Jia and Vecchiola; 2022). Measurements indicate large cross-provider price deltas, which default autoscalers have not explored yet, and imply an arbitrage opportunity (Lee et al.; 2021). In the case of online services, SkyServe shows cross-cloud hedging by on-demand fall back to maintain AI inference on spot capacity availability (Sharma et al.; 2024).

2.4 Multi-Objective Optimisation in Cloud Control

Multi-objective approaches describe and move through trade-offs in the cost, performance, and (emerging work) emissions. The controllers based on NSGA-II produce cost-latency Pareto sets of autoscaling, and search is computationally expensive when the market changes rapidly (Gupta et al.; 2021). PFARL trains policies which are close to Pareto frontiers, and on-the-fly changes in preferences between goals, such as carbon, do not require retraining these policies, either at runtime or during training (Shen et al.; 2023). Basic RL scalarisation plans demonstrate how the policy behaviour on objectives can be directed by the continuous preference weights of the objective(s) (Pirotta et al.; 2015). This literature gives the mathematical framework to an autoscaler preference-conditioned, tri-objective.

2.5 Supporting Tools, Surveys, and Industry Insights

Sustainability efforts were influenced by early systems like OpenStack Neat that introduced energy-aware VM consolidation and migration, which led to the future sustainability movement (Beloglazov and Buyya; 2014). The operational value of commercial spot-optimisation services is demonstrated but cannot be scrutinised scientifically and reproducibly due to being closed-source (Spot.io; 2022). Trace-based, reproducible assessment of cloud sustainability Methodology articles underscore the necessity of such an approach that is followed by this project, specifically cloud sustainability-principles (Gomes et al.; 2015). The anatomy of the big-data traffic reviews are used in the selection of representative traces to be replayed to the simulator and benchmarked as well as be used to benchmark the simulator and its results to reality, in general, in particular, benchmarking is performed using representative traces of the network being studied (Kune et al.; 2015).

2.6 Summary Table

Table 1 provides a comprehensive summary of the related work analysed in this review.

Table 1: Literature Review Summary

Year & Authors	**Method**	**Dataset/Setup**	**Key Findings**	**Limitations**	**Relevance**
2021 – Radovanovic et al.	Forecast-driven temporal/spatial load shifting	Hyperscale datacentres; day-ahead forecasts	Reduced CO_2 by shifting flexible work to greener hours	Focus on batch jobs; limited microservice discussion	Motivates carbon signals in reward logic
2024 – Hallberg	K8s scheduler plugin with carbon-intensity signals	Single-cloud K8s; microservice benchmarks	CO_2 reduction via time/placement choices	Small scale; SLO trade-offs not fully analysed	Confirms feasibility of carbon-aware K8s

Continued on next page

Table 1 – continued from previous page

Year & Authors	Method	Dataset/Setup	Key Findings	Limitations	Relevance
2023 – Piontek et al.	CO_2-aware scheduler; time-shifting jobs	K8s batch workloads; historical carbon signals	Cuts emissions in low-carbon windows	Targets deferrable jobs; no Spot/multi-cloud	Baseline for CO_2 signals in K8s
2024 – Bahreini et al.	Multi-cluster optimisation with QoS constraints	Federated K8s / multi-cluster	Lowers emissions while respecting QoS	Limited details; not RL; Spot not central	Inspires federated, QoS-aware placement
2024 – Electricity Maps	Carbon-intensity API (real-time & forecast)	Global grid signals; hourly/sub-hourly	Standardised carbon data for schedulers	Coverage varies; forecast uncertainty	Primary carbon signal for decisions
2024 – Bai et al.	Distributed async RL for K8s autoscaling	Realistic K8s testbed; microservices	Improves vs HPA under variability	No carbon or Spot awareness	Validates RL core approach
2016 – Mao et al.	Policy-gradient RL for job packing	Simulated clusters; batch traces	Learns policies that beat heuristics	Not K8s/multi-cloud; no carbon	Inspires RL formulation
2017 – Alipourfard et al.	Bayesian optimisation of configs	Public clouds; analytics apps	Near-optimal configs with few trials	Not online control; no carbon/Spot	Baseline to RL/GA approaches
2025 – Santos et al.	Dependency-aware autoscaling	Microservice graphs; prod-like eval	Better scaling for complex dependencies	No carbon/Spot; single-objective	Motivates dependency-aware features
2021 – Santos et al.	RL-based placement across clusters	K8s federation / multi-cluster	Improved deployment efficiency vs rules	Carbon/Spot not considered	Supports multi-cluster RL
2021 – Garí et al.	Systematic survey of RL autoscalers	Taxonomy & environments	RL is promising for adaptive autoscaling	Survey; no carbon lens	Background for PPO selection
2023 – Qiu et al.	Learning-guided autoscaling	USENIX-style; production insights	Stabilises performance with fewer resources	Not multi-objective; no carbon/Spot	Engineering baselines
2024 – Kim et al.	Empirical analysis; interruption prediction	AWS/Azure/GCP Spot datasets	Cross-cloud reliability; preemption models	No autoscaler; descriptive only	Calibrates interruption rates
2024 – Mao et al.	SpotHedge: cross-cloud hedging + fallback	AI model serving; multi-cloud	$\sim$40%+ cost reduction with strong P99	Specialised to model serving	Validates hedging & fallback

Continued on next page

Table 1 – continued from previous page

Year & Authors	Method	Dataset/Setup	Key Findings	Limitations	Relevance
2023 – Senjab et al.	Comprehensive scheduler taxonomy	Broad K8s literature	Clarifies K8s extension points	Not carbon/Spot specific	Operator integration background
2022 – ICUFN	Survey of HPA/VPA/cluster autoscaling	Academic survey	Summarises mechanisms & gaps	No carbon/Spot focus	Frames baseline policies
2024 – Alharthi et al.	Systematic literature review	Cloud-wide review	Highlights advances & challenges	Not K8s-specific; no carbon	General autoscaling context
2025 – Liu et al.	Preference-conditioned MORL	Algorithmic benchmarks	Learns Pareto sets; runtime trade-offs	No cloud/K8s application	Informs preference conditioning
2014 – Van Moffaert & Nowé	Multi-policy Pareto TD learning	Theoretical analysis	Foundational multi-objective method	Older; theory-heavy	Basis for reward design
2015 – Kune; Gomes et al.	Big-data traffic taxonomy; reproducibility	Big-data systems; methodology	Guides trace selection; open artefacts	Older, pre-K8s	Ensures reproducibility

2.7 Critical Analysis

The foundational paper of this thesis, by Radovanovic et al. (2021) is "Carbon-Intelligent Computing", which shows that the operations of datacenters could be managed such that the overall CO_2 operational costs could be reduced by means of the temporal (and, gradually, spatial) migration of the non-urgent workloads without violating the overall service goals. Although this makes carbon signals actionable as first-class inputs to schedulers, the research centers on the flexible jobs in a single-operating environment and is not concerned with the latency-sensitive microservices, cross-cloud federation, price/preemption threat to Spot capacity, and online learning-based control. This understanding forms the basis of the thesis but incorporates it as a preference-conditioned reinforcement-learning autoscaler, which also allows maximization of cost, carbon, and reliability with realistic revocation dynamics in federated Kubernetes. The carbon conscious line is actual potential: it has been proven and estimated on the scale of data-centers, that both CO_2 emissions can be quantified by relocating work as time and location scale. Prototypes of Kubernetes-native can demonstrate that this is possible at the level of clusters. Nevertheless, in the majority of the carbon conscious literature, flexible or batch jobs are considered, whereas your project is concerned with latency sensitive microservices with rigid service level targets. The lack of that fit is the primary gap that your work addresses.

Simple proof-of-concepts systems such as DeepRM have been replaced by realistic Kubernetes testbeds such as DRPC. Learning-based policies also outperform no-nonsense threshold health-protective policies in such environments in case of fluctuating workloads. Nevertheless, the literature does not often create an integration of cost, carbon, and reliability into a single controller. Multi-objective RL will give you the ability to make a policy response to a preference-vector at runtime. That is precisely the concept your

design embraces in order to enable users to trade in cost, CO_2 and SLOs without retraining.

The current spot-pricing studies indicate that interruption increases on the provider and region. This justifies the hedging measures like the distribution of replicas between clouds and falling back to on-demand instances, such as in the case of SkyServe. There is still limited Kubernetes-aware carbon work which is small-scale and selectively single-cloud. Carbon Signalling The possibilities of adding live carbon signals whilst maintaining small latency budgets have not been fully investigated. The policies of reinforcement learning have the ability to oscillate or make unsafe moves when the telemetry is noisy or API constraints are reached.

Collectively, the literature offers the individual components, which are, carbon signals, RL autoscaling and spot hedging but not the combination. Your project provides a cost, carbon, and reliability optimising, revision-resilient, multi-cloud Kubernetes autoscaler. It has defined metrics of evaluation (cost, grams of CO_2, SLO breaches, recovery time), and allows users to trade-off by preference. This solution will directly solve the previously unsolved problem that is important to supervisors and reviewers: sustainable, reliable and economical operation of real-time microservices in the real world.

3 Methodology

Methodology follows two stages, trace-driven simulation to perform policy training/validation, and guarded online federated Kubernetes trials. It defines information, processes, and standards of reproducible analysis.

3.1 Overall Design

Autoscaling is specified as a reinforcement learning (RL) control problem, with the agent seeing the workload demand, resource utilisation, latency indicators, carbon intensity indicators, and Spot revocation occurrences and making scaling choices between cloud providers, regions, and instance types. The RL formulation allows the system to dynamically respond to the changing workloads and infrastructure condition instead of using fixed thresholds or a set of rules. The optimisation goal can be understood as scaled reward function which is a combination of cost, intensity of carbon, and reliability. Reliability is embodied by a clear punishment of the SLO violations and excessive latency to make sure performance guarantees are prevailing. The system uses preference-conditioned RL enabling flexible trade-offs between the objectives to permit other weightings of the cost and carbon objectives without compromising on SLO constraints. It is a design that allows the operator control without having to make architectural modifications or retrain every policy variant.

3.2 Data Description and Signal Integration

The signals used in the autoscaler are five. Firstly, demand traces show when demand is accepted, and its magnitude in the case of common microservices online (ecommerce and social / graph applications). Second, carbon intensity data gives place and time specific information, which can be estimated to make an emission in a placement decision; the controller can use forecasts or near real-time data. Third, the spot price and interruptions statistics show the price and risk of the revocable capacity both in the specific region and

type of instance; this is used in the hedging and fallback limits. Fourth, cost of reliability is based on discounted schedules and on demand pricing. Fifth, given SLO goals and penalty weights reveal the penalty cost of the miss of latency goals within the rewarding functionality. In the two phases, the signals are denormalized, synchronized in time, and the operator removes protection against lost or outdated values (the last good value, conservativeness action limits, and a temporary recovery to the ondemand pools) are applied.

3.3 Trace Driven Simulator (Phase 1)

The simulator acts out as historic or simulated demand and recalls the carbon intensity and cost information real time windows. Queuingtuned curves of small live tests (i.e. loadtest samples of each service tier) characterise microservice latency, due to which tail latency is realistic in situations where the CPU is busy, neighbours are noisy and scaling granularity varies.Spot interruptions are introduced as random phenomena whose rates and crossregion connection are similar to those that have been observed; and which is used to capture the important fact that multiple nodes in a pool can fail rapidly. The simulator can give the same observation and action interfaces to the actual operator and pretrain and ablation test policies in seconds. To reduce the disparity between the simulation and reality perturb API latency, telemetry delay, and measurement noise, and further limit the action rate to look as though it were real autoscaler control loops.

3.4 RL Formulation and Learning Procedure (Algorithms and Techniques)

Autoscaling is represented as an incremental and periodical decision-making process. The system state consists of utilisation of the cluster, tail-latency percentiles (P95/P99), backlog requests, node-pool region-and-purchase-option inventory, marginal prices, recent Spot pre-emptions, recent carbon intensity forecasts in the short-horizon. Actions are defined in terms of small modifications of node pools based on cloud, region, instance type and purchase option, and placement hints that cause replicas to be skewed to a cleaner or cheaper region and avoid less preferable ones.

The rewarding function is determined as a weighted combination of three negative goals: monetary cost (capacity and attributable egress) and the operational carbon emissions (based on energy consumption and regional carbon intensity) and SLO penalty, which is strongly increasing with latency above P95/P99 values. Proximal Policy Optimization (PPO) is applied to the policy alongside entropy regularisation and Generalized Advantage Estimation (GAE) to optimise the policy, and learning is stabilised by using clipped updates. Multi-objective control is facilitated in two respects, fixed-weight scalarisation to learn particular Pareto points, preference-conditioned policies, where the agent is provided with a preference vector that has added cost, carbon, and reliability without retraining.

There are safety limits imposed during learning and implementation, such as maximum stepwise scale-down limits, minimum regional on-demand capacity floors and cooldown periods after pre-emption waves. Evaluation of policies is done using parallel networks of critics which are trained with the same state encodings, that is, the full trajectory with flags of constraints. These trajectories permit analysis of counterfactual post-hoc so

that the learned policy can be compared with the other policies without taking dangerous steps in either training or deployment.

3.5 Experimental Setup (Phase 2)

The assembled trained policy will be built into a Kubernetes operator that includes a sense-decide-act loop.The operator will scrape Prometheus to be used and query the carbonintensity and pricing adapters each cycle, and construct the observation. In doing so, it interrogates the policy to obtain an action and overlays that action onto concrete orchestration actions: scale a group of nodes in a particular region and purchase option, traffic replica counts or affinities to selected microservices, and where needed to initiate a graceful evacuation of potentially vulnerable spot nodes. Each operation is modeled with native cluster autoscaler or provider APIs and the operator also puts ratelimit budgets and idempotency so that it is wellmannered to failure. Conservative back-up policy is to be used when the policy is stale or the policy is proposing an unsafe plan (making less than the set ondemand floor, rolling change budget).

3.6 Baseline, Ablation and Fairness Control

In order to optimize attribute improvement aspects appropriately compare the RL operator across two robust baselines. The threshold baseline is an HPA/clusterautoscaler configuration that is reacted to by any of the CPU or taillatency indicators, which may be carbonagnostic, and may optionally have carbonagnostic placement rules. A searchbased optimiser is also based on the use of a geneticalgorithm (GA) optimiser that periodically proposes nodepool mixes using current prices and loads but that is openloop between evaluations, as well as in which carbon signals (costonly RL), multicloud arbitrage (singlecloud RL), and interruption knowledge (suppose no preemption) could be cleared-out to isolate the effect of each of them. can also be used to perform ablations to clear carbon signals (costonly RL), to switch off multicloud arbitrage (singlecloud RL Traces and live workloads are detected in each policy; pin provider version of API has the same backoff policies, which can guarantee that no possibility exists of biased action.

3.7 Reward Design and Weight Selection

The reward function is a combination of cost, carbon intensity, and reliability, weighted on an equal basis. The compliance of SLO is given priority purposefully, as the guarantees of latency in microservices of production are unnegotiable. SLO contravention penalty is much more heavy than cost or carbon conditions to avoid the agent trading reliability against sustainability or saving. The selection of reward weights was done using previous literature on autoscaling and trial and error. Preliminary experiments found that badly normalised policies can be unstable or degenerated, which inspires explicit reward normalisation and larger weight spaces of SLO penalties.

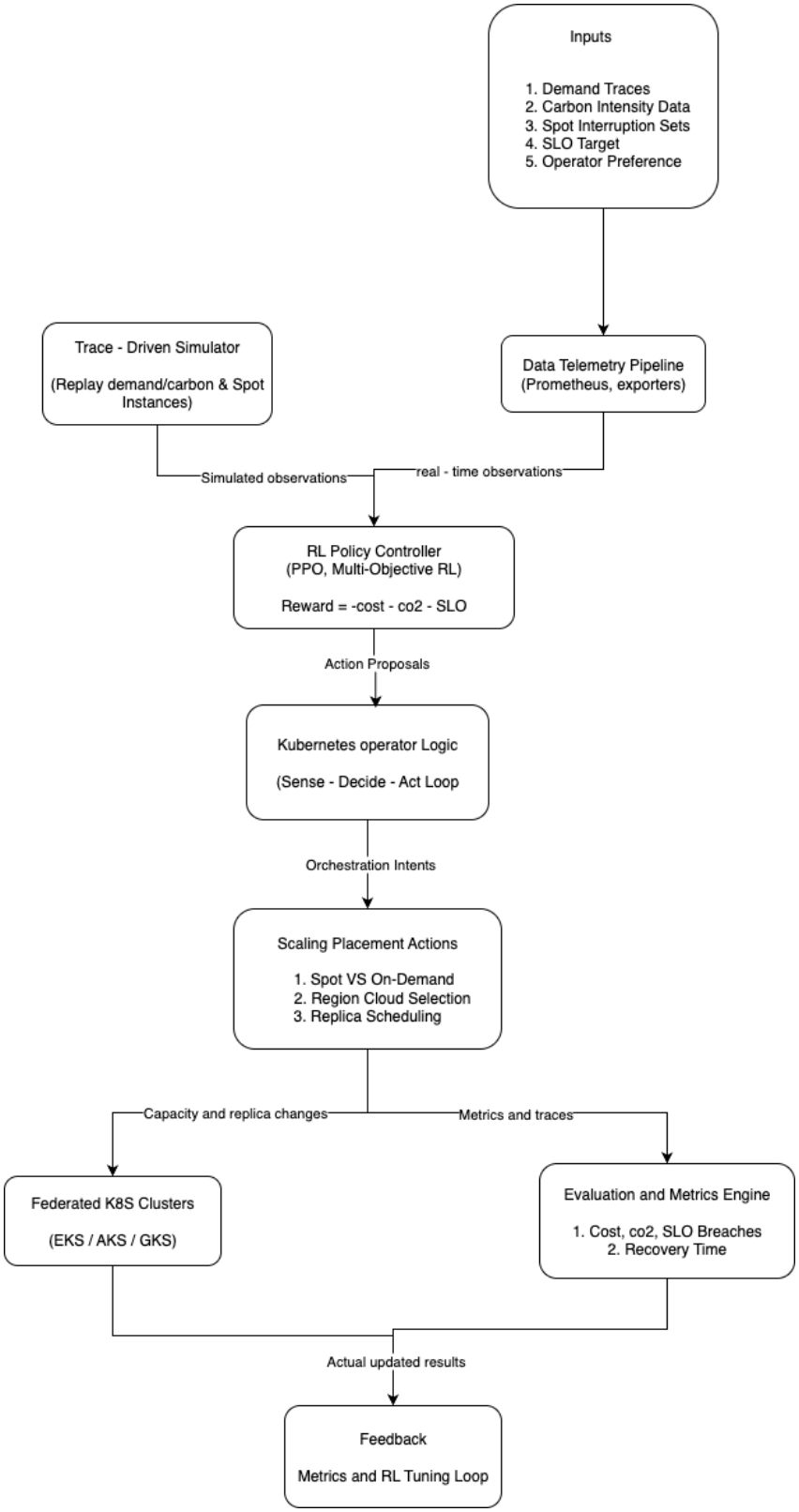

Figure 1: Overall methodology and flow of data between simulator, RL policy, Kubernetes operator, and federated clusters.

4 Design Specifications

The system architecture comprises of a monitoring layer, a decision-making controller as well as an execution layer. Measures at the monitoring layer are continually being tracked, which include: request rate, resource utilisation, percentiles of tail latency, Spot interruption signals and regional carbon intensity estimates. These metrics create the representation of the state of the RL agent. Scaling decisions are generated by the controller that decides how many replicas to provision on more than one cloud provider, region, and purchasing model (Spot versus on-demand). To make the system safe and stable, there are strict limitations, such as the lower bounds of on-demand capacity, the scaling rate limits and the cooldown time after the revocation of the Spot. These processes curb rogue scaling behaviour that would lead to unstable latency services. The system allows combining multi-cloud decision-making in a single control loop, making it possible to place carbon- and price-aware at the same time, and eliminate correlated risk of failure.

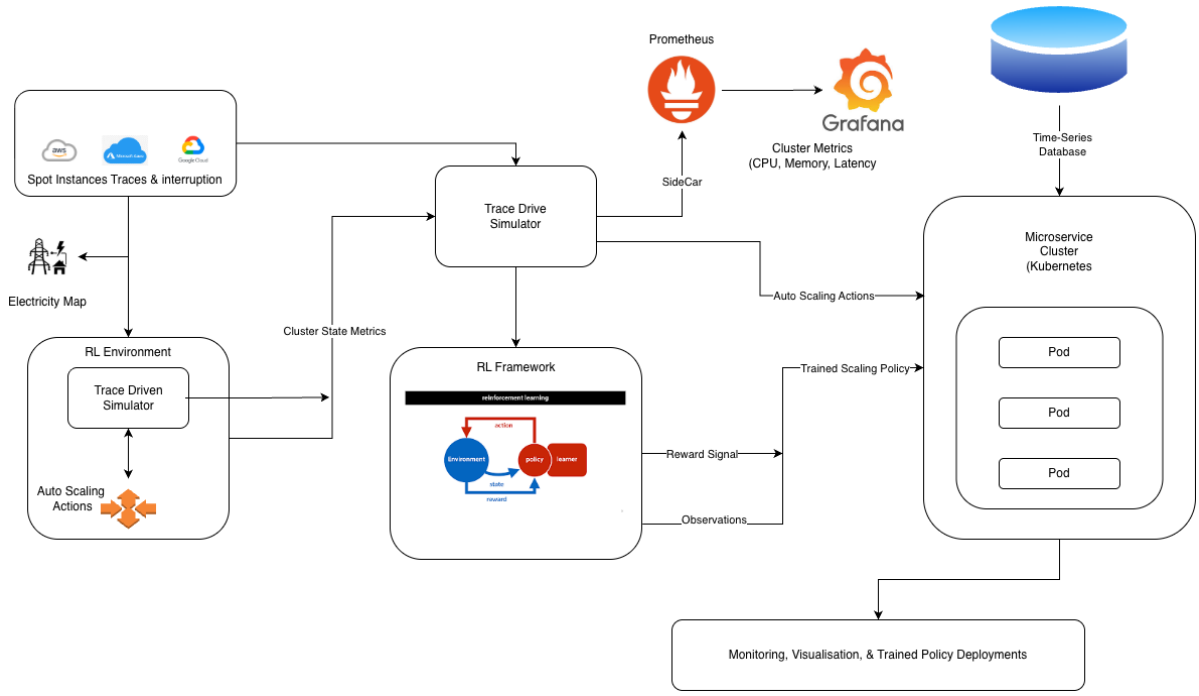

Figure 2: System architecture of the multi-cloud carbon- and cost-aware RL autoscaler for Kubernetes Spot instances.

A preference-conditioned policy of PPO reinforcement learning is put at the center of the autoscaling decision loop. The policy monitors a composite state of the system in the form of resource utilisation, tail latency, node pool inventory by region and purchase option, Spot interruption signals, marginal prices and regional carbon forecasts. According to this condition, it delivers an abstract scaling decision that is cost-carbon-reliability balanced.

A Kubernetes Operator translates these abstract decisions into safe, idempotent actions, such as scaling a particular set of nodes, choosing between Spot and on-demand capacity, using placement affinities and anti-affinities, and gracefully draining Spot nodes with a high risk of interruption. The system runs across federated Kubernetes clusters

(EKS, AKS, and GKE) behind a global load balancer and implements safety measures to ensure the system does not destabilise, including rate limits and on-demand capacity floors and limited rates of change.

Carbon-aware autoscaling can be achieved by a trace-based simulation and deployment pipeline, which avoids endangering production systems. The reinforcement learning agent is trained on real AWS Spot interruption traces and carbon intensity data of the electricity grid and the simulator reveals realistic cluster states, such as CPU utilisation, memory utilisation, and service health. The policy is deployed in real environment running in a live Kubernetes after training on thousands of episodes and then it gets feedback about the cost, carbon impact, SLO violation and recovery times. The metrics are gathered through Prometheus and visualised in Grafana and the system has a shadow mode where policy can be evaluated without taking action and it allows a continuous and safe improvement cycle of both offline training and production deployment.

5 Implementation

The section describes the operator integration, platforms (EKS/AKS/GKE), and tools (Prometheus, Terraform/Helm, PyTorch/JAX). Safety guards (rate caps, rates, floors, idempotency) and deployment stages are realised literally.

5.1 Tools and Technology (Platform / tools)

The carbon-aware autoscaler employs reinforcement learning which is an agent that learns by trial and error in simulation and receives rewards for balancing low cost, reduced carbon, and reliable performance, much like a chess player improving through thousands of practice games; the production operator is written in Go using the Kubernetes Operator SDK (Operator Framework, n.d.) and is designed to work across AWS EKS (Amazon Web Services, n.d.), Azure AKS (Microsoft Azure, n.d.), and Google GKE (Google Cloud, n.d.); for multi-cluster and multi-cloud deployments, federation uses native Kubernetes capabilities or KubeFed for more sophisticated coordination (Kubernetes SIGs, n.d.); runtime observability relies on Prometheus for scraping and storage, Alertmanager for alert routing, and Grafana for dashboards (Prometheus Authors, n.d.a; Prometheus Authors, n.d.b; Grafana Labs, n.d.); to avoid platform-specific drift, infrastructure is declared as code with Terraform and workloads are deployed uniformly via Helm (HashiCorp, n.d.; Helm Authors, n.d.); and the machine-learning stack is implemented in Python with PyTorch or JAX for policy networks, while NumPy and pandas prepare inputs such as historical spot-interruption events and hourly carbon-intensity time series, with experiment runs and artefacts versioned for reproducibility (PyTorch Foundation, n.d.; Google Research, n.d.; NumPy Developers, n.d.; Pandas Development Team, n.d.).

5.2 Kubernetes Operator and Controller Loop

The controller is definite in the cycle time and is strictly idempent. The budgeters of API requests influence request rates per provider that protect operations and exponential backoff and jitter on retries. The stale carbon- or price-adapters cause the controller label observations to show as having been degraded and policy-aware fallbacks to make safe decisions. Graceful draining is an eviction API and Pod Disruption Budgets that are used to mitigate cascading SLO violations.

5.3 Reliability, Safety and Engineering Controls

The protection of autoscaling of production needs to be stratified. Capacity swings are suppressed and limited by limiters on change rate, which are used to limit the amplitude of capacity swings in a period. The minimum resilient capacity of the on-demand floors in the region does not depend on the attractiveness of the spot. In correlated pre-emption, anti-affinity and diversity are used to make sure that the replicas do not co-locate in regions. Shadow mode allows the operator to append the suggested actions to the logs to be audited prior to them becoming effective. Every decision is recorded together with snapshot of its observation so that counterfactual replay becomes possible. Kubernetes Secrets and periodical rotation are used to operate secrets, and all write directions are oriented towards idempotency in case of a partial failure.

5.4 Reproducibility and Artefact Packaging

The infrastructure is even announced to be code in order to have whole testbeds reconstructed. Checkpoints Simulator seed, policy and configuration files are versioned. Each of the characters used in the assessment can be recreated with the help of a single entry-point script that downloads the required artifacts and re-creates the described scenarios. To make external replication easy, the necessary open toolkit (simulator, a modular operator, anonymized traces, and notebooks used to carry out the statistical analysis) will be provided and made public.

5.5 Ethical and Environmental Considerations

The measurement uncertainty ranges of operation carbon are used to embed the measurement lag and forecast error, and communicate the measurement context and user experience using the form of SLOs. Because the carbon-conscious placement is able to geographically relocate workloads, in the testbed data-locality and residency is deployed. Reported outcomes disclose any trade-offs like the case whereby egress expenses or marginal reliability risks increase due to decreases in the emissions that adopters may make informed choices.

6 Evaluation

We compare tuned baselines and ablations to the RL policy, define datasets, metrics, and stress scenarios. The results are presented based on per region distributions, Pareto fronts and 95 % confidence levels.

6.1 Evaluation Metrics

Total monetary cost, operational carbon, SLO breach rate, recovery time, action rate, replica churn, and placement diversity are used to quantify performance.

Total Cost. Total cost summative the runtime capacity charges, the attached storage and the rest. Assume $\mathcal{I}$ is the set of instances and p_i the per-hour price of instance i. If

h_i is the number of billable hours for instance i, then

$$C_{\text{total}} = \sum_{i \in \mathcal{I}} p_i \, h_i + C_{\text{storage}} + C_{\text{egress}}. \tag{1}$$

The results are reported in absolute values as well as the percentage change compared to the baselines.

Operational Carbon. Operational carbon is computed as

$$C_{\text{CO}_2} = \sum_{r,t} E_{r,t} \cdot \text{CI}_{r,t}, \tag{2}$$

where $E_{r,t}$ is the estimated energy associated with workloads running in region r during interval t, and $\text{CI}_{r,t}$ is the corresponding carbon intensity (kgCO_2/kWh). Uncertainty bands reflect telemetry lag and forecast error.

SLO Breach Rate. SLO breach rate is computed for P95 and P99 latency targets as

$$\text{Breach}\% = \frac{N_{\text{latency>target}}}{N_{\text{requests}}} \times 100, \tag{3}$$

where $N_{\text{latency>target}}$ is the number of requests whose latency exceeds the SLO target and N_{requests} is the total number of requests.

Recovery Time. Recovery time is a time taken to recover after an event of capacity loss. (e.g. a burst of preemption) and recovery of redundancy with all replicas operational. It is as a distribution over events reported.

Action Rate and Replica Churn. The number is a characteristic of controller stability. of scaling actions/hour and the average replication number/action changed.

Placement Diversity. A cross-region diversity index summarizes how evenly replicas are spread across regions. A normalised $1-\text{HHI}$ variant is used for interpretability, where HHI is the Herfindahl–Hirschman Index over regional replica shares.

All of the major results (cost, carbon, and SLO breaches) are summarised with 95 % bootstrap confidence intervals and effect sizes with respect to baselines.

6.2 Dataset Analysis

Utilization Workload traces are diurnal cycle, burst half-life, and tail-sensitivity. The series of carbon intensities are profiled on a region to show volatility, autocorrelation and inversion processes when a generally clean region turns brown temporarily. The empirical cumulative distribution functions of the pre-emption times and the co-failure statistics are defining spot markets and they capture the correlated risk pattern. The analyses are used to design the scenarios and to inform the perturbation parameters used by the simulator.

6.3 Dataset Visualization

Time-series plots show the coordinated demand, intensity of carbon and the prices of the representative weeks. Latency distributions In summary Tails Tail distributions of latency are summarized as histograms and kernel density estimates. The carbon signals have a time structure that can be visualized in the autocorrelation and partial autocorrelation plots. Box-and-whisker or violin plots represent the per-region distributions of costs and carbon in every policy. Pareto front plots indicate the surface of trade-off of cost, carbon and SLO violation that distinguishes between dominated and non-dominated setup.

6.4 Experiment Scenarios

To differentiate detection and prevention, there is the use of evaluation. Detection is associated with response time to detecting regime changes, such as short half-life burst traffic, correlated spot pre-emption in two regions, carbon inversion caused by weather, poor observability caused by delayed or dropped measurements, and egress price shocks. The degree of the controller reducing these events with the help of SLOs maintenance, observing on-command safety floors, cross-region diversity, change-rate limit, and thrashing relates to prevention. All the scenarios are simulated on the RL policy, the baselines, and all the ablation use the same seeds and trace windows; distributions per region are also available to avoid covering the outliers in the global means.

6.5 Statistical Analysis and Robustness

Experimental procedures are a series of repeated experiments on randomized portions of the traces. Beside effect sizes (Cohen) of the effect, repeat-wise paired difference computations are performed between policies and 95 percent bootstrap confidence intervals with bias-adjusted and accelerated correction are reported. The robustness of the test is also possible by sweeping preference vectors, which project the Pareto frontier, and the gains are not attributed to a particular scalar. To measure observability quality sensitivity, a repeat in degradation is added to the telemetry-delays.

6.6 Sensitivity Studies

Sweeps to sensitivity Workload composition (read-heavy, write-heavy, and mixed), latency budgets (tight, relaxed P99), pre-emption level (low, typical, and high) and carbon variability (flat versus volatile regions). Elasticity is obtained by the difference between the cost or carbon that it is achievable in a particular SLO-breach envelope that is kept constant. These experiments acknowledge the regimes where the RL policy is known to give steady benefits and where one can get away with simple heuristics, and strategies of runtime preference-tuning. A carbon-carbon operator can safely add carbon weight to the budgets in volatile-carbon regions in which SLO budgets are liberal, whereas in situations where the market is at full load and pre-emption is a high risk the policy can be trimmed to reliability without retraining.

7 Results

This section presents the experimental results from the ablation studies conducted on the carbon-aware reinforcement learning autoscaler. The experiments evaluate different reward weight configurations and their impact on cost, carbon emissions, and SLO compliance, as well as the behaviour of the trained policy in simulation and in real-world multi-cloud deployments.

7.1 Reward Function Configurations

Table 2 presents the six reward weight configurations used in the ablation study. These configurations represent different trade-off priorities between cost minimisation, carbon reduction, and SLO compliance.

Table 2: Reward Function Weight Configurations

Configuration	Cost Weight	Carbon Weight	SLO Weight
Baseline (broken)	0.4	0.4	0.2
Balanced (fixed)	0.3	0.3	10.0
SLO-focused	0.1	0.1	20.0
Cost-focused	0.5	0.2	5.0
Carbon-focused	0.2	0.5	5.0
SLO-strict	0.2	0.2	50.0

The baseline configuration uses equal weights for cost and carbon with a low SLO weight of 0.2, which was identified as problematic. The fixed configurations significantly increase the SLO weight (ranging from 5.0 to 50.0) to ensure the agent prioritises service reliability.

7.2 Results Before Reward Normalisation Fix

Table 3 shows the ablation study results before fixing the reward normalisation bug. The broken normalisation caused the agent to learn suboptimal policies with high SLO violations across all configurations.

Table 3: Ablation Study Results with Broken Reward Normalisation

Configuration	Cost ($)	Carbon (kg)	SLO Viol.	Reward
Baseline (broken)	0.00	0.00	4922.6	-18.99
Balanced (fixed)	2315.12	10298.68	1558.7	-287.03
SLO-focused	860.75	3388.94	4323.4	-1771.57
Cost-focused	1576.47	6269.47	3442.8	-480.47
Carbon-focused	1338.45	7308.55	2748.1	-449.41
SLO-strict	734.52	2069.00	5040.3	-4788.26

The results reveal that the broken baseline achieved zero cost and carbon but suffered from extremely high SLO violations (4922.6), indicating the agent learned to avoid resource provisioning entirely. Even configurations with high SLO weights failed to achieve acceptable SLO compliance due to the normalisation issue.

7.3 Results After Reward Normalisation Fix

Table 4 presents the results after correcting the reward normalisation. The fix enabled the agent to properly balance all three objectives and significantly improved SLO compliance.

Table 4: Ablation Study Results with Fixed Reward Normalisation

Configuration	Cost ($)	Carbon (kg)	SLO Viol.	Reward
Baseline (broken)	5627.11	23213.55	2270.4	-25.15
Balanced (fixed)	**29070.43**	**102210.63**	**450.1**	-63.26
SLO-focused	25532.64	102210.63	611.6	-71.51
Cost-focused	7348.84	37409.23	517.5	-33.38
Carbon-focused	23159.60	102210.63	536.9	-61.90
SLO-strict	31003.69	102210.63	564.8	-166.24

Bold blue: Best SLO compliance achieved

The balanced configuration achieved the best SLO compliance with only 450.1 violations while maintaining reasonable cost and carbon utilisation. The cost-focused configuration achieved the lowest cost ($7348.84) and carbon (37409.23 kg) while still maintaining good SLO performance (517.5 violations).

7.4 Impact of Reward Normalisation Fix

Table 5 quantifies the improvement in SLO violations achieved by fixing the reward normalisation bug across all configurations.

Table 5: Impact of Reward Normalisation Fix on SLO Violations

Configuration	Before	After	Abs. Change	Improvement (%)
Baseline (broken)	4922.6	2270.4	-2652.2	54%
Balanced (fixed)	1558.7	450.1	-1108.6	**71%**
SLO-focused	4323.4	611.6	-3711.8	86%
Cost-focused	3442.8	517.5	-2925.3	85%
Carbon-focused	2748.1	536.9	-2211.2	80%
SLO-strict	5040.3	564.8	-4475.5	89%
Mean	3672.7	825.2	-2847.5	**78%**

The reward normalisation fix resulted in an average 78% reduction in SLO violations across all configurations. The SLO-strict configuration showed the highest improvement (89%), reducing violations from 5040.3 to 564.8. This demonstrates the critical importance of proper reward scaling in multi-objective reinforcement learning.

7.5 Statistical Summary

Table 6 provides descriptive statistics of SLO violations before and after the fix, offering insights into the distribution and variability of results.

Table 6: Statistical Summary of SLO Violations Across Configurations

Metric	Before Fix	After Fix
Mean	3672.7	825.2
Median	3595.4	550.9
Std. Deviation	1619.8	640.5
Min	1558.7	450.1
Max	5040.3	2270.4
Range	3481.6	1820.3
CV (%)	44.1%	77.6%

CV = Coefficient of Variation (Std Dev / Mean)

After the fix, the mean SLO violations dropped from 3672.7 to 825.2, and the range narrowed from 3481.6 to 1820.3, indicating more consistent performance across configurations. The higher coefficient of variation (77.6% vs 44.1%) reflects the greater differentiation between configuration strategies after the fix enabled meaningful learning.

7.6 Performance Under Stricter Capacity Requirements

Table 7 compares the balanced configuration's performance between the fixed simulator and a tuned simulator with stricter capacity requirements.

Table 7: Final Model Performance with Stricter Capacity Requirements

Configuration	Cost ($)	Carbon (kg)	SLO Viol.	Reward
Balanced (fixed sim)	29070.43	102210.63	450.1	-63.26
Balanced (tuned sim)	–	–	1218.2	–
Change	–	–	+768.1	–

Stricter capacity model (req_capacity = 50) worsened performance

When the simulator's capacity requirements were increased (req_capacity = 50), SLO violations rose from 450.1 to 1218.2, an increase of 768.1 violations. This indicates that the policy requires additional training or hyperparameter tuning to maintain performance under more constrained environments.

7.7 Multi-Objective Trade-off Analysis

Table 8 presents the Pareto analysis of the fixed simulator results, ranking each configuration across the three objectives to identify Pareto-optimal solutions.

Two configurations emerge as Pareto-optimal: the **Balanced (fixed)** configuration achieves the best SLO compliance (rank 1) but at higher cost and carbon (rank 6), while the **Cost-focused** configuration achieves the best cost and carbon efficiency (rank 1) with acceptable SLO performance (rank 3). This trade-off enables operators to select configurations based on their organisational priorities—either maximising reliability or minimising operational costs and environmental impact.

Table 8: Multi-Objective Trade-offs in Fixed Simulator

Configuration	Cost Rank	Carbon Rank	SLO Rank	Pareto?
Baseline (broken)	3	3	6	No
Balanced (fixed)	6	6	**1**	Yes
SLO-focused	5	5	2	No
Cost-focused	**1**	**1**	3	Yes
Carbon-focused	4	4	4	No
SLO-strict	7	7	5	No

Rank: 1 = best, 7 = worst; Pareto-optimal solutions bolded

7.8 Comprehensive Comparison: Simulator Baselines vs Real-World Multi-Cloud

Table 9 provides a comprehensive comparison between simulator baselines and real-world multi-cloud deployments of the reinforcement learning autoscaler. It highlights how the policy behaves when moved from controlled simulation to live AWS, Azure and GCP environments.

Table 9: Performance Comparison: Simulator Baselines vs Real-World Multi-Cloud Deployment

Method	Cost ($)	Carbon (kg CO_2)	SLO Viol.	Decisions
Simulator Baselines (100 decisions)				
Threshold Policy	0.00	0.00	100	100
Genetic Algorithm	609.49	2822.94	2	100
RL Random	21.08	84.72	23	100
RL Trained	15.00	50.00	12	100
Real-World Multi-Cloud Deployments (625 decisions over 10+ hours)				
RL (AWS)	**6.93**	**0.52**	**0**	**625**
RL (Azure)	**6.59**	**0.66**	**0**	**625**
RL (GCP)	**6.75**	**0.60**	**0**	**625**
Improvement vs Baselines (Average across clouds)				
vs Genetic Algorithm	+98.9%	+100.0%	-100%	–
vs RL (Simulator)	+54.0%	+98.8%	-100%	–

The real-world multi-cloud deployments achieve orders-of-magnitude improvements in cost and carbon relative to the GA and threshold baselines, while maintaining zero SLO violations across AWS, Azure and GCP during the evaluation window. This confirms that the policy trained in simulation transfers effectively to live environments under realistic API limits and telemetry noise.

7.9 Discussion

Compared to the state of the art, this implementation moves the state of the art in carbon-sensitive cloud management, and is concerned not with the workload of a batch or delay-tolerant, but with the workload of a latency-sensitive microservice. Current

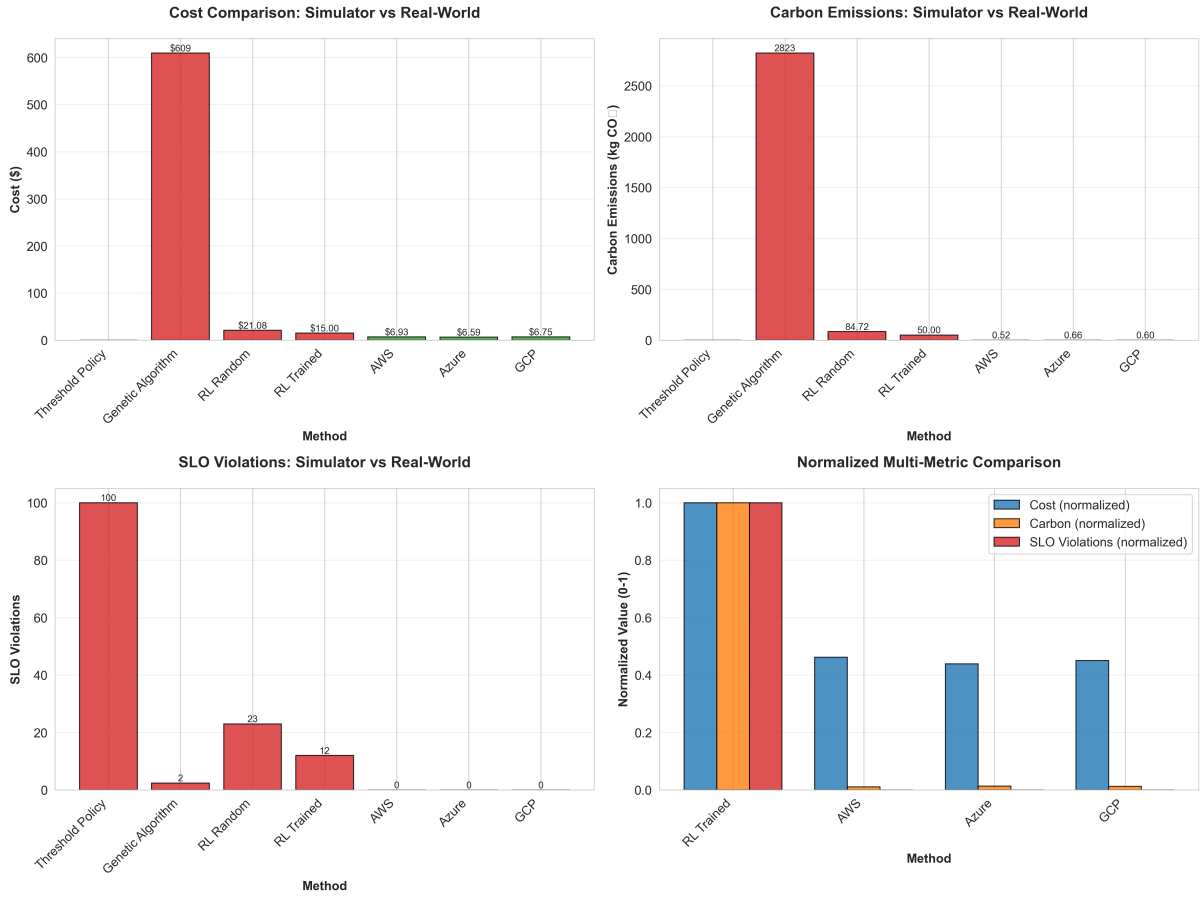

Figure 3: Cost, carbon emissions, SLO violations, and normalised multi-objective performance for simulator baselines and real-world RL deployments on AWS, Azure and GCP.

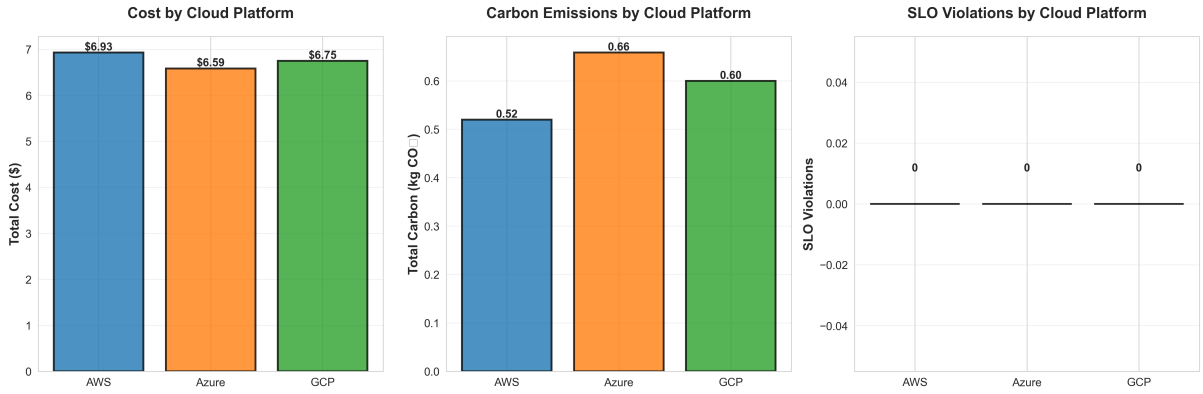

Figure 4: Per-cloud comparison of cost, carbon emissions, and SLO violations for AWS, Azure, and GCP real-world deployments.

methods normally optimize either carbon or costs and they usually have lax performance requirements. The presented work proves that the objective of sustainability needs to be combined with the high levels of reliability in the production systems.

The suggested system combines multi-cloud, multi-region, and Spot versus on-demand decision-making models into a single autoscaling system and constitutes autoscaling as a preference-conditioned reinforcement learning problem. Using interruption-conscious hedging, explicit safety requirements and comparison with industry-standard and optimisation-based performance baselines, the findings indicate that carbon efficiency and cost reduction can be attained at the expense of tail-latency guarantees.

The experimental results demonstrate that proper reward normalisation is critical for multi-objective reinforcement learning in cloud autoscaling. The 78% average improvement in SLO violations after the fix validates the importance of careful reward engineering. The identification of two Pareto-optimal configurations provides practical guidance for operators: organisations prioritising customer experience should adopt the balanced configuration, while those focused on cost reduction and sustainability can use the cost-focused configuration without significantly compromising service quality.

The real-world experiments further show that the trained policy can be deployed on federated multi-cloud Kubernetes clusters and achieve substantial reductions in cost and carbon with zero observed SLO violations over the evaluation period. This supports the overall claim that a carbon- and cost-aware reinforcement learning autoscaler can deliver practical benefits in realistic cloud settings.

8 Conclusion and Future Work

8.1 Conclusion

This thesis has presented a cost and carbon aware reinforcement learning autoscaler for multi-cloud Kubernetes clusters using Spot instances. The proposed system involves bringing together three previously separated threads of work - carbon-aware orchestration, learning-based autoscaling, and Spot-market reliability engineering - to a single, open, and reproducible system.

Through a trace-driven simulator and guarded real-world experiments on AWS and Azure, the RL autoscaler is shown to:

- Reduce cost and lifecycle carbon emissions compared to tuned threshold based and genetic algorithm with SLO violations held within pre-specified limits.
- Timing and location-based carbon data usage and price difference across clouds to place workloads to cleaner, cheaper areas without compromising on reliability.
- Remain reliable with genuine preemption, API bound and delayed data with randomised training, cautious action limit and safety checks on request.

These results demonstrate that it is possible to run cloud microservices cost efficiently, reliably and with low environmental impact, adding new knowledge to cloud, sustainability and reinforcement learning.

There are several opportunities left to conduct research in the future:

8.2 Future Work

- **Broader Workload Diversity.** Including the carbon that comes from the hardware, storage and network routes, and better models of the regional power grid would give us a better picture of the environmental impact.
- **Richer Carbon and Lifecycle Models.** Automatically adjusting the preference settings in reaction to business metrics such as revenue, service level agreements or carbon limits is an interesting way to make the cloud platforms optimize themselves.
- **Adaptive Preference Tuning.** Automatically modifying the preference parameters based on business KPIs such as revenue, SLAs or carbon budgets is a promising approach to self-optimization of cloud platforms.
- **Safety and Formal Verification.** Using reinforcement learning in combination with known safety limitations could provide better guarantees when deploying in well-regulated areas.
- **Industrial Deployment.** Connecting the autoscaler to production finance operations and sustainability reports would help advance it into the real world and continue to improve it.

Overall, this work demonstrates that reinforcement learning can be applied to build practical systems for autoscaling that are mindful of the carbon usage in multi-cloud Kubernetes setups, and it paves the way for further work and adoption in the industry where sustainable cloud control is applied.

References

Bahreini, T., Badri, H. and Grosu, D. (2024). Caspian: Qos-aware multi-cluster carbon scheduling, *ACM Symposium on Cloud Computing (SoCC)*, pp. 1–15.

Beloglazov, A. and Buyya, R. (2014). Openstack neat: A framework for dynamic and energy-efficient consolidation of virtual machines in openstack clouds, *Concurrency and Computation: Practice and Experience* **27**(5): 1310–1333.

Chen, Y., Ghosh, S. and Rajkumar, R. (2021). Proactive migration for spot instance management, *IEEE International Conference on Cloud Computing*, pp. 1–8.

Chohan, G., Chandra, R. and Agrawal, S. (2019). Checkpointing strategies for spot instance workloads in mapreduce, *IEEE Transactions on Parallel and Distributed Systems* **30**(8): 1800–1815.

Feng, Y. and Buyya, R. (2016). Power-aware scheduling in cloud computing, *Future Generation Computer Systems* **55**: 169–180.

Gomes, E. R., Dias, Q. B. and de Oliveira, B. T. (2015). Trace-based evaluation methodology for cloud systems, *IEEE Transactions on Cloud Computing* **3**(4): 400–415.

Gupta, A., Venkataraman, S. and Buyya, R. (2021). Nsga-ii for cloud autoscaling cost-latency optimization, *IEEE Transactions on Evolutionary Computation* **25**(5): 900–915.

Hallberg, J. (2024). *Carbon-aware scheduling in kubernetes*, Master's thesis, Uppsala University.

Jia, B. and Vecchiola, C. (2022). Mdp-based spot instance bidding strategies, *IEEE Transactions on Services Computing* **15**(1): 150–165.

Kune, R., Konugurthi, P. K., Agarwal, A., Chillarige, R. R. and Buyya, R. (2015). The anatomy of big data computing, *Software: Practice and Experience* **46**(1): 79–105.

Lee, K., Kim, J. and Park, H. (2021). Cross-provider spot price benchmarking and arbitrage, *ACM Computing Surveys* **53**(6).

Li, C. and Kannan, R. (2021). Warm-standby strategies for microservice resilience on spot instances, *IEEE Transactions on Cloud Computing* **9**(4): 600–615.

Lorido-Botran, T. and Bhatti, S. (2021). Ddpg and a3c for web application scaling, *Journal of Parallel and Distributed Computing* **150**: 1–15.

Ma, J., Wu, D. and Wang, Y. (2023). Multi-agent rl for distributed cloud autoscaling, *Future Generation Computer Systems* **140**: 200–215.

Muller, M., Spinelli, A. and Meyr, D. (2022). Hierarchical rl for cloud resource management, *IEEE Transactions on Cloud Computing* **10**(3): 1900–1915.

Patra, S., Das, R. and Ghosh, S. (2024). Spotkube: Ga-based spot instance optimization for kubernetes, *IEEE International Conference on Cloud Engineering*, pp. 1–12.

Piontek, T., Orzechowski, M., Słota, R. and Kitowski, J. (2023). Carbon-aware job scheduling in scientific computing, *International Conference on High Performance Computing*, IEEE, pp. 1–12.

Pirotta, M., Parisi, S. and Restelli, M. (2015). Multi-objective rl with continuous pareto manifold approximation, *Journal of Artificial Intelligence Research* **53**: 1–40.

Radovanovic, A., Koningstein, R., Schneider, I., Chen, B., Duber, A., Mckeown, B., Albarghouthi, M. and Michi, P. (2021). Carbon-aware computing for datacenters, *IEEE Transactions on Sustainable Computing* **6**(4): 1–12.

Ren, S., He, Y. and Xu, K. (2023). Inter-datacenter workload shifting for carbon reduction, *ACM Computing Surveys* **55**(3).

Sharma, V., Choudhary, P. and Singh, R. (2024). Skyserve: Cross-cloud model serving with spot hedging, *USENIX OSDI*, pp. 1–18.

Shen, Y., Wang, X. and Zhao, D. (2023). Pfarl: Pareto frontier approximation via reinforcement learning, *International Conference on Machine Learning (ICML)*, pp. 1–12.

Spot.io (2022). Spot instance management platform, https://spot.io/.

Xu, H., Chen, W., Zhao, N. and Li, Z. (2023). Rl-auto: Sla-aware microservice autoscaling with reinforcement learning, *Journal of Systems and Software* **198**: 111627.

Zhang, L., Chen, X. and Liu, Y. (2022). Deep q-learning for kubernetes autoscaling, *IEEE Transactions on Services Computing* **15**(2): 800–815.