

Reinforcement Learning Project Proposal

Report last updated: October 3, 2025 (Initial Submission)

Project Name: **Application of Reinforcement Learning to Cloud Auto Scaling**

**Student Names**

|  |
| --- |
| Balasubramanyam, Srivatsa |
| *Healy, Ryan* |
| *McGregor, Bruce* |

**Project Aims**

This project will explore whether reinforcement learning can make smarter decisions about cloud auto-scaling than today’s simple threshold rules. We aim to build a small simulator where cloud workloads go up and down over time, then train an RL agent to decide when to add or remove capacity. The goal is to see if basic RL methods like SARSA and Q-learning can keep performance high while avoiding unnecessary cost.

**Framing as a Reinforcement Learning Problem**

* State Space: Each state will describe how busy the system is (low, medium, or high utilization), how many capacity units are currently active (1–5), and whether demand is rising, flat, or falling (the trend feature). The trend is based on whether current workload demand is higher, lower, or the same as the previous step. This addition gives the agent context about the direction of change so it can learn to anticipate scaling needs instead of reacting too late.
* Action Space: At each step, the agent can pick one of three actions: scale up (add capacity), scale down (remove capacity), or hold steady.
* Agent: A reinforcement learning agent using SARSA or Q-learning with ε-greedy exploration.
* Environment: A simulator that replays workload demand from the Kaggle dataset. After the agent takes an action, the simulator updates system capacity, calculates utilization, and returns a reward.
* Reward Function: The agent would earn points for keeping utilization in a healthy range (not too low, not too high), loses points for SLA violations when overloaded, loses a small number of points for wasted capacity, and may also lose points if it changes capacity too often.

**Data Elements and Sources**

We plan to use explore use of both simulated and real-world datasets to drive the cloud auto-scaling environment. This approach lets us prototype quickly with lightweight data while leaving open the possibility of testing against more realistic traces.

* Kaggle – Cloud Computing Performance Metrics https://www.kaggle.com/datasets/abdurraziq01/cloud-computing-performance-metrics
  + Simulated CPU utilization and other system metrics
  + Normalized CPU values will provide workload demand traces
  + Used to build utilization buckets and compute trend features
  + Lightweight, easy to use for prototyping and debugging
* GitHub – Awesome Cloud Computing Datasets - https://github.com/ACAT-SCUT/Awesome-CloudComputing-Datasets
  + Curated list of large-scale, real-world traces
  + Includes Google Cluster Data, Alibaba Cluster Traces, and others
  + Candidate for adding realistic workload patterns
  + May be used to test how well the RL agent generalizes beyond synthetic data

At this stage, our plan is to begin with the Kaggle dataset for initial development, then evaluate the feasibility of incorporating a subset of a real-world trace.

***Resources***

To ground our project in existing research and ensure technical feasibility, we reviewed key papers on reinforcement learning for cloud autoscaling and identified the most relevant tools and software libraries.

**Guiding Papers**

* García et al. (2020): Survey of RL-based autoscaling; frames state, action, and reward design choices.
* Qiu et al. (2023): AWARE system; RL autoscaling in production; highlights challenges like noisy workloads and safe exploration
* Xu et al. (2022): Predictive autoscaling via meta-RL; motivates our use of a trend feature to anticipate demand shifts.

**Tools and Software**

* NumPy & pandas: Data handling, workload preprocessing.
* Matplotlib & Seaborn: Visualization of demand traces, learning curves, policies.
* Gymnasium (OpenAI Gym): Standard RL environment interface (reset(), step()).

**Future Work Plan**

Our work plan for the remainder of the semester in summarized in the table below*.*

|  |  |
| --- | --- |
| Task Description | Target Date |
| Begin exploring the Kaggle dataset, normalize CPU utilization, and experiment with simple demand traces | October 11 |
| Build an initial version of the simulator (states, actions, rewards) and test different ways of including the trend feature | October 18 |
| Try out simple baseline policies; compare how well they track demand; refine reward design if needed | October 25 |
| Start implementing RL agents (SARSA, Q-learning); experiment with different exploration rates and episode lengths | November 1 |
| Run initial experiments with RL policies; evaluate early results and adjust simulator design or state representation as needed | November 8 |
| Explore feasibility of incorporating one of the real-world traces from the GitHub dataset collection; test integration if time permits | November 15 |
| Continue refining experiments, focusing on SLA vs. cost trade-offs and the effect of the trend feature | November 22 |
| Consolidate results, generate plots and visualizations, and begin drafting the report | November 29 |
| Finalize report and prepare presentation | Dec 6 - 9 |