

Configuration Manual

MSc Research Project
MSc in Artificial Intelligence

Shayshank Rathore
Student ID: 23348186

School of Computing
National College of Ireland

Supervisor: Abdul Shahid

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Shayshank Rathore
Student ID:	23348186
Programme:	MSc in Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Abdul Shahid
Submission Due Date:	01/09/2025
Project Title:	Configuration Manual
Word Count:	2028
Page Count:	13

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:	Shayshank Rathore
Date:	31st August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
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.	<input type="checkbox"/>

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):	

Configuration Manual

Shayshank Rathore
23348186

1 Introduction

1.1 Project Summary

This configuration manual provides step-by-step guidance on how to set up and reproduce the experiments for the MSc Research Project titled “**Hybrid Reinforcement Learning for Personalized Diabetes Care**”. The manual is intended for researchers and practitioners who wish to replicate the environment, datasets and code execution in order to achieve the same results on Kaggle.

The project addresses two long-standing challenges in the application of Reinforcement Learning (RL) for healthcare:

- **Lack of Explainability:** RL policies often act as black-boxes, making it difficult for clinicians to trust recommendations.
- **High Data Requirements:** RL typically requires large datasets, which are rarely available in healthcare contexts where patient data is sparse and heterogeneous.

To address these issues, the project develops a **Hybrid RL framework** that integrates:

1. **Reward Decomposition:-** Where the total reward is separated into clinically interpretable components such as glycaemic control, hypoglycaemia penalties and treatment cost.
2. **Meta-learning:-** Enabling the agent to adapt rapidly to new patient profiles using limited data, thus improving sample efficiency.

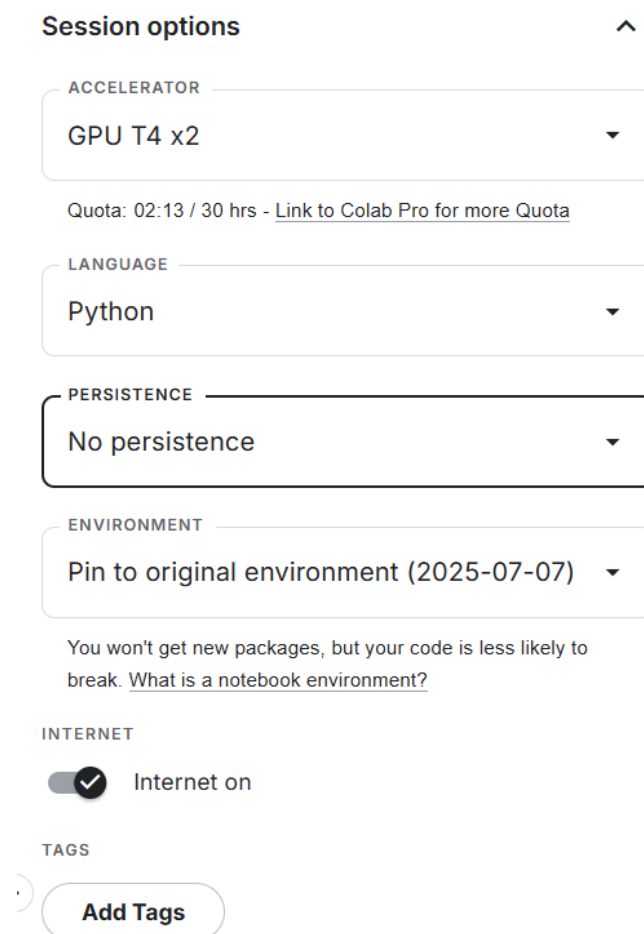
This manual documents the runtime environment, system dependencies, dataset preparation, execution workflow and troubleshooting guidelines, ensuring full reproducibility of results on Kaggle.

2 System Configuration

2.1 Execution Environment

All experiments were conducted on **Kaggle Notebook**, a managed Jupyter-like environment with preinstalled machine learning libraries and GPU acceleration. Kaggle provides ephemeral storage and limited session runtimes, making it suitable for reproducible experiments but requiring results to be downloaded at the end of each session.

- **Kernel type:** Python 3 (CPython backend)
- **Accelerator:** $2 \times$ NVIDIA Tesla T4 GPUs (16 GB VRAM each, CUDA 11.8 support)
- **System RAM:** Up to 20 GB (13 GB by default)
- **Disk space:** ~ 60 GB storage (reset after each session)
- **Operating system:** Ubuntu 20.04 LTS (Kaggle default image)
- **Session limits:** 12 hours maximum runtime per GPU session



The image shows the 'Session options' configuration panel in Kaggle. It includes dropdown menus for 'ACCELERATOR' (GPU T4 x2), 'LANGUAGE' (Python), 'PERSISTENCE' (No persistence), and 'ENVIRONMENT' (Pin to original environment (2025-07-07)). Below these is a toggle for 'INTERNET' (Internet on) and a button for 'Add Tags'.

Session options ^

ACCELERATOR

GPU T4 x2 ▼

Quota: 02:13 / 30 hrs - [Link to Colab Pro for more Quota](#)

LANGUAGE

Python ▼

PERSISTENCE

No persistence ▼

ENVIRONMENT

Pin to original environment (2025-07-07) ▼

You won't get new packages, but your code is less likely to break. [What is a notebook environment?](#)

INTERNET

☒ Internet on

TAGS

Add Tags

Figure 1: Kaggle draft session configuration showing GPU T4 \times 2

2.2 Python and Package Versions

We relied on Kaggle's pre installed Python environment (v3.10.12) and installed additional libraries where required. The core dependencies used were:

- Python 3.10.12
- numpy 1.25.2
- pandas 2.0.3
- scikit-learn 1.3.0
- matplotlib 3.7.2
- seaborn 0.12.2
- gymnasium 0.29.0
- stable-baselines3 2.1.0
- torch 2.0.0 + CUDA 11.8

2.3 Reproducibility Settings

To ensure reproducibility of experiments, the following practices were adopted:

- Fixed random seeds across `numpy`, `torch` and `gymnasium`.
- Stored preprocessing scaler parameters and reused them across train/validation/test splits to prevent leakage.
- Saved model checkpoints, logs and plots at regular intervals for re-analysis.
- Downloaded final outputs before the Kaggle session ended, as storage is ephemeral.

3 Installation & Environment Setup

3.1 Kaggle Notebook Settings

Before running:

- Enable **GPU** (T4). If High RAM is available, select it.
- Turn **Internet** ON only if you need to fetch your own dataset from Kaggle Datasets. We have used our own dataset we created.

3.2 Project Notebook & Data Attachment

1. Open the main notebook in Kaggle.
2. In the right panel, click **Add Data** and attach the provided diabetes dataset so it mounts under:

`/kaggle/input/diabetes-data/`

3.3 Folder Structure During Execution

```
/kaggle/input/diabetes-data/    # raw data (mounted read-only)
/kaggle/working/                # writeable workspace for this run
  data/                         # intermediate/preprocessed files
  models/                       # saved PPO checkpoints
  outputs/                      # logs, tables, plots
```

3.4 Environment Verification (No Extra Installs Needed)

The libraries used are available by default. Verify versions at the top of the notebook:

```
import gymnasium as gym, stable_baselines3 as sb3, torch, numpy, pandas
print("gymnasium", gym.__version__)
print("stable-baselines3", sb3.__version__)
print("torch", torch.__version__)
```

3.5 Persistence

Kaggle storage under `/kaggle/working/` resets when the session ends.

Download models and results from `/kaggle/working/outputs/` after training.

4 Dataset Preparation

4.1 Source Dataset

The experiments were conducted using the **UCI AIM'94 Diabetes dataset**, which contains event logs for 70 insulin-dependent patients. Each record consists of four tab-separated fields:

- **Date** (MM-DD-YYYY)
- **Time** (HH:MM)
- **Code** (categorical identifier for event type)
- **Value** (glucose reading in mg/dL or insulin dose in units)

The dataset includes two data collection modes:

- **Electronic recorder logs**, with accurate timestamps.
- **Paper diaries**, mapped to fixed logical times (08:00, 12:00, 18:00, 22:00).

This mixture produces irregular sampling and missingness patterns, typical of real-world diabetes records.

4.2 Preprocessing Pipeline

To make the raw logs usable for Reinforcement Learning (RL), the following steps were applied:

1. **Ingestion and parsing.** Patient files were loaded with a persistent ID. Date and Time fields were merged into a single timestamp, invalid entries were dropped and records sorted chronologically.
2. **Event harmonisation.** The UCI event codes were mapped to human-readable categories (e.g., glucose measurement, insulin type). Only relevant events were retained, duplicates and timestamp collisions were removed.
3. **Temporal alignment.** Records were resampled to fixed hourly steps. Within each bin:
 - Latest glucose value was retained.
 - Insulin doses within the hour were summed.

Short gaps were forward-filled, longer gaps were excluded.

4. **Type safety and outlier handling.** Non-numeric values were coerced. Extreme outliers were clipped at quantile thresholds.
5. **Feature construction.** Each state vector included:
 - Current glucose and lag values.
 - Rolling statistics of glucose.
 - Insulin given in recent time windows.
 - Time-of-day indicators.
6. **Reward indicators.** Helper flags for hypoglycaemia, hyperglycaemia and insulin burden were created. Reward computation was deferred to the RL environment.
7. **Normalisation.** A `StandardScaler` was fitted on the training split and applied consistently to validation and test sets.
8. **Splitting.** Patients were partitioned into train, validation and test sets *by patient ID*, ensuring no overlap and testing generalisation.
9. **Quality checks.** Confirmed no missing values in essential features, validated distributions and ensured sufficient sequence lengths for rollouts.

4.3 Prepared Dataset Structure

After preprocessing, the dataset for each patient consisted of aligned hourly sequences of the form:

`[state_t] → action_t → [state_{t+1}, reward_t]`

with the decomposed reward:

$$r_{total} = r_{range} + r_{hypo} + r_{cost} \quad (1)$$

This structure was then wrapped into a custom Gymnasium environment.

4.4 Dataset Diagnostics

To validate preprocessing, diagnostic plots were generated:

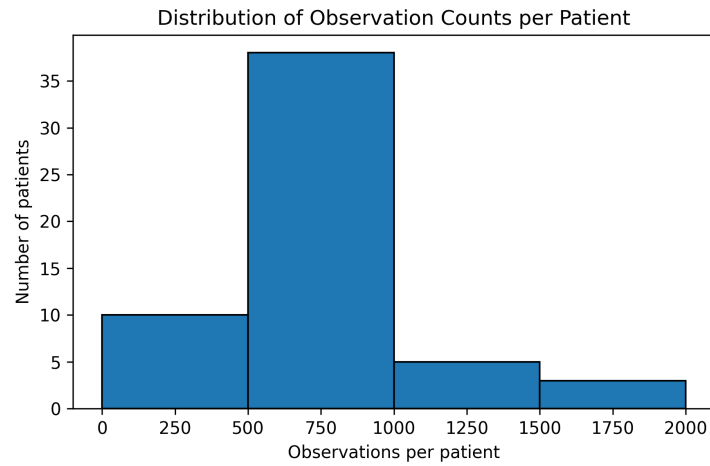


Figure 2: Distribution of observation counts per patient after preprocessing. Variability highlights sparse and irregular logging across the cohort.

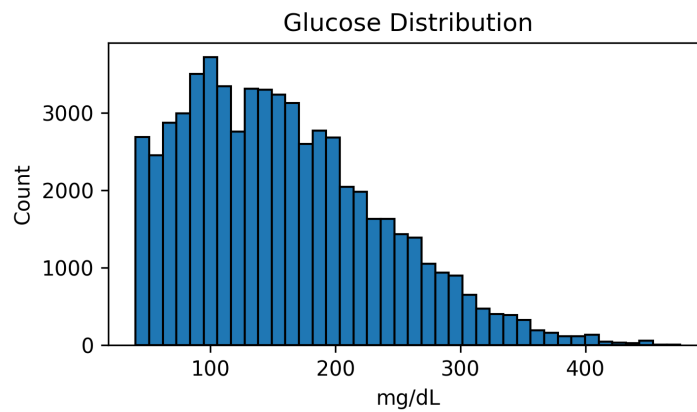


Figure 3: Distribution of glucose values across all patients. The right-skewed profile reflects frequent hyperglycaemic states.

5 Code Configuration

5.1 Notebook Structure

The main implementation was carried out in Kaggle notebook .The notebook follows a consistent modular structure:

1. **Imports and environment setup** Load required Python libraries and fix random seeds.
2. **Data preprocessing** Load raw UCI Diabetes event logs, apply cleaning and resampling. Normalise features and split patients into train/validation/test sets.

3. **Environment construction** Create a custom Gymnasium environment wrapping the preprocessed patient sequences.
4. **Model training** Train a Proximal Policy Optimization (PPO) agent using Stable-Baselines3.
5. **Evaluation and logging** Evaluate trained models on validation and test patients, log decomposed reward channels and save action distributions.
6. **Output saving** Store trained models, metrics and diagnostic plots under the `/kaggle/working/outputs/` directory.

5.2 Configurable Parameters

Several hyperparameters and design choices can be modified directly in the notebook:

- **Training timesteps:** default $\sim 30,000$ per run.
- **Discount factor (γ):** default 0.99.
- **Batch size:** default 64.
- **Reward weights:** Relative importance of `range`, `hypo penalty` and `treatment cost`.
- **Action space:** Discrete with three insulin-intensity levels (low/none, moderate and high).
- **Validation split:** Proportion of patients held out for tuning.
- **Random seed:** Fixed for reproducibility but can be adjusted for robustness checks.

5.3 Output Artefacts

The following artefacts are generated during execution:

- **Models:** PPO checkpoints saved under `models/`.
- **Logs:** TensorBoard-style training logs and reward decomposition logs saved in `outputs/`.
- **Plots:** Learning curves, action distributions and glucose histograms saved as `.png` files in `outputs/`.
- **Tables:** Summary metrics (e.g., TIR, insulin use per step, hypoglycaemic events) saved as CSV files in `outputs/`.

5.4 Reproducibility Notes

- All experiments are self-contained inside a single notebook; no external scripts are required.
- Outputs are overwritten with each new run, download them after completion to retain results.
- Changing reward weights or action space design alters both training dynamics and evaluation outcomes.

6 Execution Workflow

6.1 Step-by-Step Procedure

The project can be reproduced by running the Kaggle notebook sequentially from top to bottom. The main execution stages are:

1. **Imports and environment setup** - Verify GPU is active in Kaggle session. - Import all required libraries (`numpy`, `pandas`, `torch`, `gymnasium`, `stable-baselines3`). - Fix random seeds for reproducibility.
2. **Dataset loading** - Mount the UCI Diabetes dataset from `/kaggle/input/diabetes-data/`. - Inspect sample rows for format consistency.
3. **Preprocessing** - Parse event logs into unified timestamps. - Map UCI event codes to glucose/insulin categories. - Resample into hourly sequences. - Construct state vectors (glucose, rolling stats, insulin history, time-of-day). - Fit and apply `StandardScaler`. - Split patients into train/validation/test sets.
4. **Environment creation** - Instantiate the custom Gymnasium environment wrapping patient sequences. - Define state space, action space (3 insulin levels) and decomposed reward channels ($r_{range}, r_{hypo}, r_{cost}$).
5. **Agent training** - Configure PPO hyperparameters (e.g., $\gamma = 0.99$, batch size = 64). - Train agent for $\sim 30,000$ timesteps. - Save model checkpoints and logs.
6. **Evaluation** - Evaluate trained PPO agent on validation and held-out test patients. - Record performance metrics: Time-in-Range (TIR), hypoglycaemic steps, hyperglycaemic steps, insulin units per step, mean glucose. - Compare against rule-based baseline.
7. **Reward decomposition analysis** - Log per-channel rewards (range, hypo penalty, treatment cost). - Generate action distribution plots and glucose histograms for interpretability.
8. **Meta-adaptation (optional)** - Apply lightweight per-patient fine-tuning (Reptile-style). - Report change in per-task reward (Δ reward).
9. **Save and export results** - Store plots, CSV tables and logs in `/kaggle/working/outputs/`. - Download outputs before session ends (Kaggle storage is reset after logout).

6.2 Expected Outputs

By following the above workflow, the user should obtain:

- PPO checkpoints stored under `models/`
- Logs of training and evaluation in `outputs/`
- Plots: learning curves, action histograms, glucose distributions
- Tables of key metrics (TIR, hypo/hyper steps, insulin units per step)
- Reward decomposition summaries showing trade-offs in agent behaviour

7 Output & Results

7.1 Generated Artefacts

Each notebook execution generates artefacts stored in the `/kaggle/working/outputs/` directory. These are grouped as follows:

- **Model checkpoints:** PPO weights saved periodically during training and final checkpoint under `models/`.
- **Logs:** Training progress (timesteps, loss, KL divergence, entropy) and evaluation results saved as text and CSV logs.
- **Plots:**
 - Learning curves (reward vs. timesteps)
 - Action distribution histograms
 - Glucose and insulin trajectories (agent vs. baseline)
 - Reward decomposition summaries (range, hypo penalty, cost)
 - Meta-adaptation reward improvements
- **Tables:** Summary metrics exported as CSV files and rendered as tables:
 - Time-in-Range (TIR, %)
 - Hypoglycaemic steps (per episode)
 - Hyperglycaemic steps (per episode)
 - Insulin units per step
 - Mean glucose level (mg/dL)

7.2 Example Results

On the held-out test set, representative outcomes (mean over 30 episodes) include:

- PPO eliminated hypoglycaemic steps (0.0 vs 1.0 for rule baseline).
- PPO reduced insulin use by $\sim 86\%$ compared to the rule baseline (1.34 vs 9.72 units/step).
- PPO maintained conservative dosing behaviour, favouring safety over aggressive glucose correction.

7.3 Representative Figures

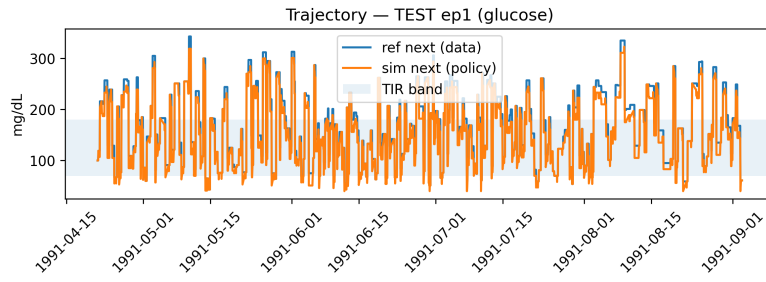


Figure 4: Glucose trajectory on test patients: PPO vs. rule baseline. PPO avoids hypoglycaemia but under-corrects hyperglycaemia.

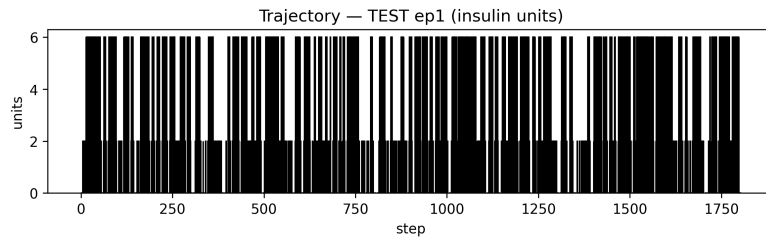


Figure 5: Insulin dosing trajectory on test patients: PPO vs. rule baseline. PPO uses $\sim 86\%$ less insulin.

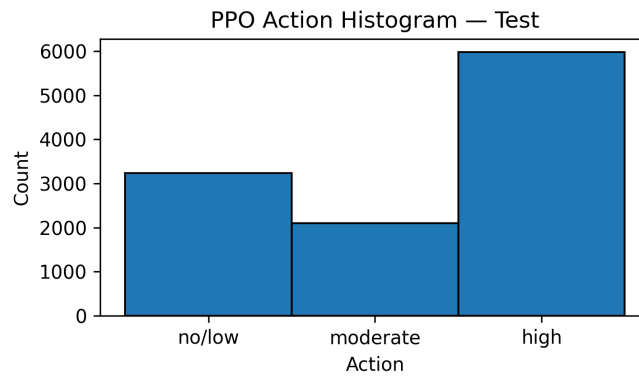


Figure 6: Distribution of insulin-intensity actions chosen by PPO. The policy strongly favours low/moderate dosing.

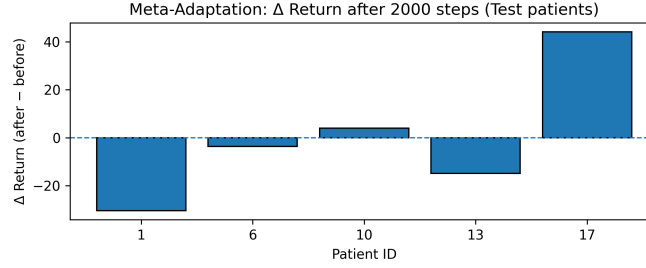


Figure 7: Per-task reward improvements after meta-adaptation. Modest but consistent gains indicate potential for personalisation.

Metric	PPO (val)	Rule (val)	PPO (test)	Rule (test)
TIR (%)	51.95	51.80	46.43	45.13
Hypo steps/ep	139.55	117.00	104.92	87.69
Hyper steps/ep	339.18	370.91	355.54	383.15
Insulin units/step	2.99	1.46	3.47	1.88
Mean glucose (mg/dL)	162.82	162.82	179.28	179.28
Avg return	-94.77	-4.81	-261.62	-196.61

Figure 8: Summary metrics (TIR, insulin use, hypo/hyper steps) comparing PPO vs. rule baseline.

7.4 Access and Persistence

- All outputs are session-specific; Kaggle clears `/kaggle/working/` after each run.
- Users must **download results locally** or commit them to a Kaggle Dataset for persistence.
- Zipped outputs (`all_outputs_YYYYMMDD.zip`) include logs, plots, and models for offline inspection.

8 Troubleshooting & Common Issues

8.1 GPU and Memory Errors

- **Problem:** Out-of-Memory (OOM) error during training.
Solution: Reduce batch size (e.g., from 64 to 32), or disable High RAM if unnecessary. PPO training is not extremely memory-heavy, so this usually occurs only if other large datasets are attached.
- **Problem:** GPU not detected.
Solution: Check Kaggle Notebook Settings → Enable GPU. Verify with:

```
!nvidia-smi
```

8.2 Dataset Issues

- **Problem:** Notebook cannot locate the dataset.
Solution: Attach the dataset manually from the right panel (**Add Data**) in Kaggle and ensure it is mounted at: `/kaggle/input/diabetes-data/`
- **Problem:** Inconsistent patient files or missing timestamps.
Solution: Re-run preprocessing cells. The pipeline drops invalid rows and forward-fills short gaps automatically.

8.3 Reproducibility

- **Problem:** Results vary slightly across runs.
Solution: Ensure seeds are fixed:

```
import numpy as np, torch, random
np.random.seed(42)
torch.manual_seed(42)
random.seed(42)
```

Note that PPO is inherently stochastic, so minor variations are expected.

- **Problem:** Different results when rerunning after session reset.
Solution: Download and reuse saved scalars, models and checkpoints from the `outputs/` folder to ensure identical conditions.

8.4 Kaggle Session Limits

- **Problem:** Outputs are lost after notebook session ends.
Solution: Always download results before session expiry. Alternatively, save them as a new Kaggle Dataset for persistence.
- **Problem:** Notebook stopped due to time limit.
Solution: Kaggle GPU sessions run for up to 12 hours. Re-run from the last saved checkpoint to continue training rather than restarting from scratch.

9 References

The following resources were referenced for environment setup, dataset access, and library usage:

- Kaggle. (2025). *Kaggle Notebooks and GPU Runtimes*. Available at: <https://www.kaggle.com/docs/notebooks>
- Kahn, M. (1994). *Diabetes Dataset*. UCI Machine Learning Repository. DOI: <https://doi.org/10.24432/C5T59G>
- Stable-Baselines3. (2023). *Reinforcement Learning Implementations in PyTorch*. GitHub repository. Available at: <https://github.com/DLR-RM/stable-baselines3>

- Gymnasium. (2023). *A standard API for reinforcement learning environments*. Farama Foundation. Available at: <https://github.com/Farama-Foundation/Gymnasium>
- Paszke, A. et al. (2019). *PyTorch: An imperative style, high-performance deep learning library*. NeurIPS 32, pp. 8024–8035.
- Pedregosa, F. et al. (2011). *Scikit-learn: Machine Learning in Python*. JMLR, 12, pp. 2825–2830.
- Harris, C. R. et al. (2020). *Array programming with NumPy*. Nature, 585, pp. 357–362.
- McKinney, W. (2010). *Data Structures for Statistical Computing in Python*. Proceedings of the 9th Python in Science Conference, pp. 51–56.
- Hunter, J. D. (2007). *Matplotlib: A 2D Graphics Environment*. Computing in Science & Engineering, 9(3), pp. 90–95.