Smart Pacer

Lorenzo Gandini
Internet of Things
University La Sapienza
A.Y. 2024/2025

June 22, 2025

Contents

1 Abstract

Highly motivated runners, from elite to amateurs, soon discover that rigid training plans treat every day as equal—even though sleep, nutrition, hormonal cycles, stress, and travel routinely shift physiological readiness. On a "bad day" an athlete may struggle to complete a prescribed 10×1 min Z5 fartlek, while on a "good day" the same session leaves untapped potential. **Smart Pacer** tackles this mismatch by casting second-by-second pacing as a finite-horizon *Markov Decision Process* and solving it with tabular *Q-learning*.

Each second the agent analyzes a compact environment defined by heart-rate zone, power zone, fatigue, phase of the workout, and slope of the ground, and selects one of three intuitive actions: *accelerate*, *hold*, or *ease*. A multi-term reward function was defined inside the simulation environment to maximize the benefit from each training session.

The model was trained using Q-learning to optimize the reward across different tested parameters, ensuring adaptability to various athlete profiles and workout types. Policies are trained off-line across four canonical workouts (fartlek, progression, endurance, recovery) and three athlete archetypes, then deployed on-line via a 1 Hz MQTT stream that simulates what a smartwatch could prompt in a real-life scenario.

Additionally, a video simulation was developed to visualize athletes' performances on different GPX tracks with the defined training programs, dynamically showing fatigue levels throughout each session.

2 Formalization of the Markov Decision Process

The pacing problem is formalised as a finite Markov Decision Process

$$\langle \mathcal{S},\ \mathcal{A},\ \mathcal{P},\ r,\ \gamma \rangle,$$

where

- $\mathcal{S}$ is the state space, a tuple of seven components that capture the athlete's physiological and contextual status.
- $\mathcal{A}$ is the action set, consisting of three discrete commands: *slow down*, *keep going*, and *accelerate*.
- $\mathcal{P}(s'|s, a)$ is the transition dynamics, which describe how the state evolves given an action.
- $r(s, a)$ is the reward function, which quantifies the desirability of each state–action pair.
- γ is the discount factor, controlling how future rewards are valued relative to immediate ones.

The MDP is implemented in the `RunnerEnv` class, which simulates the athlete's workout environment and decision-making process. The agent interacts with this environment by observing the current state, selecting an action, and receiving a reward while transitioning to a new state.

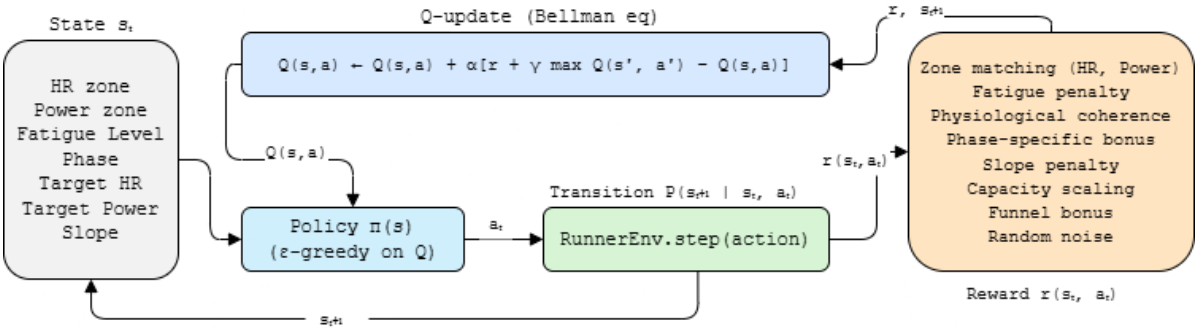


Figure 1: Schema of the Markov Decision Process (MDP) for the pacing problem. The agent observes the current state s, selects an action a, and receives a reward r while transitioning to a new state s'. The process is repeated, allowing the agent to learn an optimal policy over time.

2.1 State space $\mathcal{S}$

The states should encapsulate the athlete's physiological state in a specific moment, which is defined by the training program and the context of where the athlete is running. The components of the state space $\mathcal{S}$ are summarised in Table 1.

Component	**Meaning**
`Heart Rate`	Instantaneous heart-rate zone (Z1–Z5)
`Power Zone`	Instantaneous power zone (Z1–Z5)
`Fatigue`	Categorical fatigue (*low/medium/high*)
`Phase`	Workout phase (warm-up, push, recover, cool-down)
`Target HR_zone`	HR targets of current training segment
`Target Power_zone`	Power targets of current training segment
`Slope`	Terrain label (*uphill/flat/downhill*)

Table 1: State space $\mathcal{S}$ components.

2.2 Action set $\mathcal{A}$.

The three admissible actions that an athlete can take during a workout are defined as:

- `slow down` – reduce pace to lower heart rate and power.
- `keep going` – maintain current pace, allowing physiological drift.
- `accelerate` – increase pace to raise heart rate and power.

2.3 Transition dynamics $\mathcal{P}$

The most intricate component of the MDP is the definition of the state–transition kernel, i.e. how the athlete's physiological and contextual state evolves once an action is taken. In my simulation, this logic is encapsulated in `RunnerEnv.step()`, which invokes four update routines every second, in the following order:

- `_update_power_zone(action)` – the chosen action (*slow down*, *keep going*, or *accelerate*) instantaneously shifts the target wattage, and therefore the power zone. In real life, when you accelerate, your power output increases immediately; when you slow down, it drops right away.
- `_update_hr_zone(action)` – this function updates the target heart rate zone based on the selected action. Unlike power, heart rate is a lagged variable: it does not change instantaneously, but drifts gradually toward the target zone. This models how the cardiovascular system responds over time.
- `_update_fatigue(action)` – a dual-process model accumulates or dissipates fatigue depending on the heart rate, power zone, current workout phase, and the athlete's profile (elite, runner, or amateur).
- `_advance_segment()` – the global time index is incremented, the active workout segment is updated accordingly, and the current slope level is recalculated from the GPX elevation trace.

Applied sequentially, these rules deterministically map the current pair (s, a) to a unique next state s' at a granularity of 1 s; stochasticity is confined to the reward function, which injects small uniform noise to break ties.

2.3.1 Reward function $r(s, a)$

The reward function, together with the `_update_fatigue(action)` routine (see 2.3.2), forms the core of the simulator's logic. These components determine how the agent is incentivised to follow the training plan while accounting for the athlete's physiological state. The reward function is implemented in the `compute_reward()` method of `runner_env.py`, and outputs a scalar value that quantifies the agent's performance in the current state and it's given by the sum of eight domain-specific terms:

- **Zone–matching accuracy** - the absolute distance between the current and target HR / power zones is mapped to a piece-wise score $\{+2.0, +0.5, -1.0, -2.5, -4.0\}$; HR and power contributions are then blended as $0.4\, r_{\text{HR}} + 0.4\, r_{\text{Power}}$.
- **Fatigue management** – We maintain a continuous `fatigue_score` $f \in [0, 10]$ whose dynamics are:

$$\text{if phase} \in \{\text{recover}, \text{cooldown}\}: \quad f \leftarrow \max\big(\text{floor},\, f\, e^{-k} \;-\; d(f)\big),$$

  where

  - $k = 0.05 \times \text{fitness\_factor}$,
  - $d(f) = 0.1 \times \text{fitness\_factor} \times \sigma(f)$,
  - $\sigma(f) = 1/\big(1 + e^{-10\,(f-5)}\big)$,
  - $\text{floor} = 0.1 \times \text{fitness\_factor}$.

  In active phases (warm-up, push) f instead accumulates based on HR-zone gain constants, time in high zones, power coupling, FTP-scaling and session-type modifiers, then is clamped to $[0, 10]$. Finally, we classify f into *low*/*medium*/*high* fatigue by comparing it against the 33rd and 67th percentiles of its own recent history, so that the medium/high labels adapt in real time to the athlete's current strain.

- **Physiological coherence** – The agent computes $\Delta_Z = |Z_{\text{HR}} - Z_{\text{Power}}|$. If Δ_Z does not exceed the athlete-specific tolerance $\{0.5, 1.0, 1.5\}$, a bonus of $+1.0$ is awarded; otherwise a penalty $-1.0 \times (\Delta_Z - \text{tolerance})$ is applied. This term encourages consistency between cardiovascular strain (HR zone) and mechanical output (power zone) without overpowering the other rewards.

- **Phase–action consistency** – This function reward the agent for taking actions that are consistent with the current phase of the workout. For example, in the *warm-up* phase, accelerating while still below the target HR is mildly encouraged (+0.5), while braking is discouraged (-1); in the *recover* phase, slowing down from supra-threshold HR receives +1 while accelerating is harshly penalised (-2).:

- **Terrain-aware pacing** – How the decision taken while the slope is changing can have different impact. Accelerating on an *uphill* costs -2.0, braking on a *downhill* -0.5; all other combinations are neutral.

- **Capacity scaling** – The penalty for exceeding the athlete's Z_{HR} is attenuated by the efficiency factor (which is defined as $\min(1, \text{FTP}/(6\,\text{kg}))$), so that lighter or fitter athletes are less penalised for visiting high zones as it should be in real-life.

- **Dynamic funnel bonus** – The funnel bonus is a dynamic reward that encourages the agent to maintain a precise pacing as the workout progresses. It is defined as follows:

  - During the first half of the workout, the agent receives +2.0 for entering the target zone and +0.5 for remaining inside.
  - After halfway, the tolerance shrinks to ≤ 0, meaning that entering the target zone gives +2.0 only once, while remaining inside yields +0.5.

  This promotes sustained precision pacing.

- **Global fatigue decay & stochasticity** – After summing all partial rewards, we multiply by

  $$1 - \min\left(\tfrac{f}{200},\, 0.4\right)$$

  (capping the fatigue penalty at 40%), and finally add uniform noise $\mathcal{U}(-0.1, 0.1)$ to break ties and mimic real-world variability.

The final scalar is therefore

$$r = 0.4\, r_{\text{HR}} + 0.4\, r_{\text{Power}} + 0.3\, r_{\text{coh}} + 0.2\, r_{\text{phase}} + r_{\text{fatigue}} + r_{\text{cap}} + r_{\text{slope}} + r_{\text{fun}},$$

followed by the multiplicative decay and noise injection, as visible at the end of the `compute_reward` method in `runner_env.py`.

2.3.2 Fatigue model

The fatigue model implements a dual-process system:

- **Recovery/Cooldown Phases:** During these phases, fatigue dissipates through a combination of exponential and sigmoid decay, reflecting the natural recovery process. The decay rate and minimum fatigue floor are modulated by the athlete's

fitness factor, ensuring that fitter athletes recover more efficiently. Constants are used to control the rate of decay and to prevent fatigue from dropping below a realistic minimum.

- **Warmup and Push Phases:** In active phases, fatigue accumulates based on the current heart rate (HR) and power zones. The accumulation rate is determined by zone-specific gain constants, which are further adjusted for the type of training session (e.g., interval, fartlek, endurance), the athlete's functional threshold power (FTP), and the time spent in high-intensity zones. Additional scaling is applied if both HR and power are in high zones, and a small random noise is introduced to simulate physiological variability.

- The resulting fatigue score is capped to $[0, 10]$ and then discretized (low/medium/high) using real-time percentiles of its history, ensuring the labels always reflect the athlete's current relative fatigue.

3 General settings

3.1 Training Programs

The training programs are defined as follows and they are the typical training sessions that a runner would do during his weekly training plan:

- **Fartlek** – Represent a variable-intensity workout where the athlete alternates between high and low intensity segments, typically in a short time alternation pattern.
- **Progression** – A typical workout where the athlete gradually increases the pace over a set distance or time, starting at a comfortable speed and finishing at a faster pace.
- **Endurance** – A long, steady-state run at a moderate pace, designed to build aerobic capacity and endurance.
- **Recovery** – A low-intensity workout aimed at promoting recovery after a hard training session, typically involving easy running or walking, but with a focus on maintaining the athelte moving with a low heart rate and minimizing fatigue.

All these workouts are defined in the `trainings.json` file.

3.2 Athlete Archetypes

The athlete archetypes are defined by their physiological parameters like the heart rate value at rest, the maximal heart rate able to reach and the weight. There are also two `fitness-related` values:

- **FTP** – Functional Threshold Power (FTP) is a key metric used to define an athlete's performance profile. It represents the highest average power an athlete can sustain for about 60 minutes and is crucial for estimating personalized training zones and overall aerobic capacity. While FTP is most commonly associated with cycling, it is also relevant in running and other endurance sports. Because not every athlete knows their FTP, many fitness apps and smartwatches can estimate this value automatically by analyzing data from multiple workouts, regardless of the activity type. This makes FTP a practical and widely accessible measure for assessing and tracking athletic performance.
- **Fitness factor** – A value that represents the athlete's fitness level, which is into the range of 0,7 for an elite athlete, up to 1,3 for an amatour athlete. This value is used to determine the athlete's ability to sustain high-intensity efforts, manage fatigue and how well the athlete it's able to recover.

The archetypes, defined in the `athletes.json` file, are the following:

- **Elite** – Represents a highly trained athlete, like olympic runners with a low resting heart rate, high maximum heart rate, and high FTP. This archetype is characterized by its ability to sustain high-intensity efforts and recover quickly.
- **Runner** – Represents an athlete with average training, having typical resting and maximum heart rates, and a moderate FTP. This archetype can handle standard training sessions and recovers at a normal rate. Could be someone who has been training for a while, like several months, but is not at the elite level yet.

- **Amateur** – Represents an untrained or recreational athlete with a high resting heart rate, low maximum heart rate, and low FTP, typical of someone who begins to train. This archetype is characterized by its limited ability to sustain high-intensity efforts and recover slowly, is the one who suffers more advanced programs.

3.3 GPX Track

Training was performed on a real GPX track of the *Parco degli Acquedotti* in Rome. To probe generalisation, the learned policies were replayed unchanged on two unseen yet topographically comparable routes: a riverside path in *Parco Belfiore* and the *Lago di Mezzo* waterfront loop, both in Mantova. These additional circuits share similar average slope with the first one but differ in curve geometry and surface, allowing verification that the agent's behaviour is track-agnostic rather than over-fitted to the training venue. The results of all these simulation can be seen in the video folder.

4 Q-Learning Training Experiments

To train the reinforcement learning policy for the agent, was adopted a tabular Q-learning approach. Multiple hyperparameter combinations were tested evaluating their performance through the cumulative reward across episodes and comparing athlete-specific training results.

4.1 First Experiment: 500 Episodes

In the initial experiment, training was limited to 500 episodes per combination (reported in table **??**). This number was chosen to keep training time feasible while still allowing the Q-values to begin stabilizing and reveal early trends in learning performance.

Config Label	α	γ	ϵ_0	$\epsilon_{\min}$	**decay**
a10_g95_e20	0.10	0.95	0.20	0.01	0.99
a5_g95_e30	0.05	0.95	0.30	0.01	0.995
a10_g99_e20	0.10	0.99	0.20	0.01	0.98
a1_g95_e20	0.01	0.95	0.20	0.01	0.99
a10_g90_e20	0.10	0.90	0.20	0.01	0.98
a10_g95_e40	0.10	0.95	0.40	0.01	0.97
a5_g99_e30	0.05	0.99	0.30	0.01	0.995
a5_g90_e10	0.05	0.90	0.10	0.01	0.98

Table 2: Hyperparameter sets for 500-episode experiments

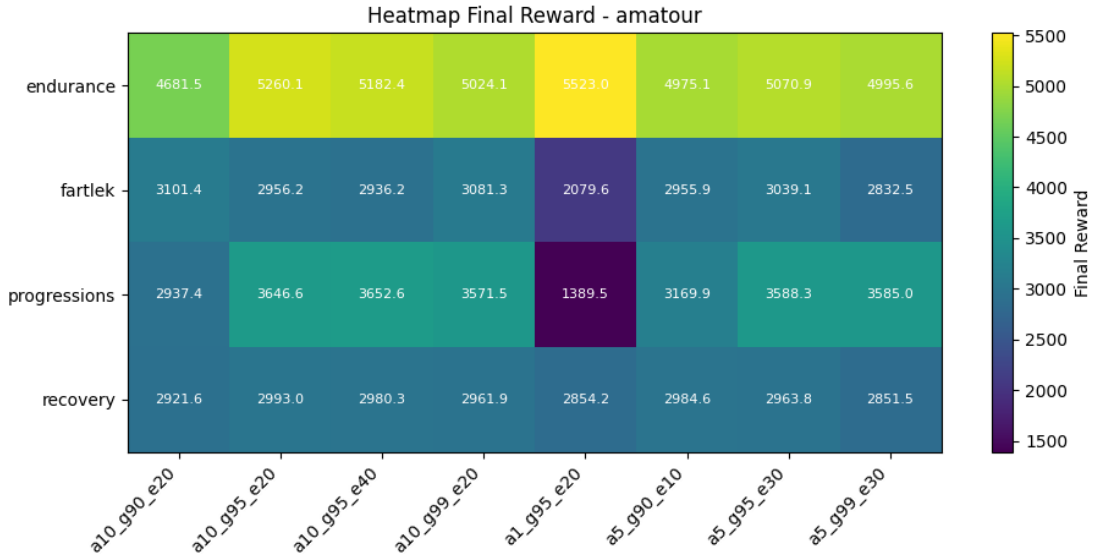
Each configuration was evaluated across the four predefined workouts and the three athlete profiles explained in section 3. For each of them was obtained a heatmap of the total rewards (fig. 2a, 2b, 2c), representing the agent's performance across the different hyperparameter settings. In order to choose the best option, also the convergence of the Q-values was analyzed, which is shown in figures 3. The convergence plots show how the Q-values stabilize over time, indicating that the agent is learning effectively.

Table 2 shows that most settings achieve a stable plateau well before the 500-episode mark; however, overly aggressive "future-looking" combinations (e.g., $\gamma = 0.99$) or highly exploratory starts (e.g., $\epsilon = 0.40$) sometimes dip late in noisy workouts like Fartlek and Progressions. The heatmaps in Figures 2a–2c make it clear that `a10_g95_e20` consistently delivers the highest final rewards across all athlete types. For an *elite* runner, who physiologically can sustain high intensity with minimal fatigue, that configuration tops the Endurance cell at about 5500 total reward, while still keeping fatigue low in recovery phases.

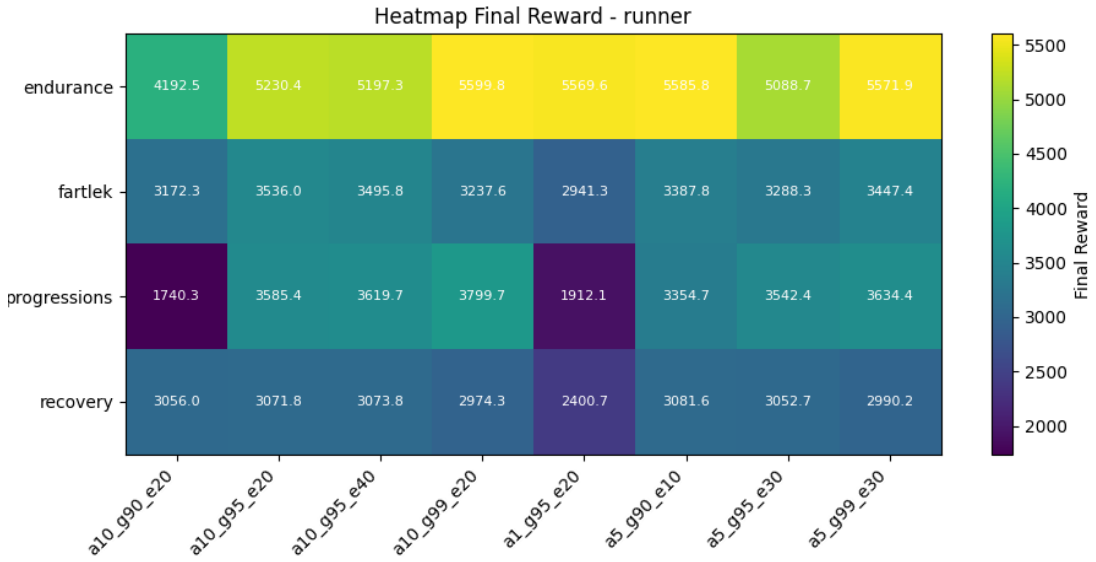
In contrast, an *amateur* is expected to accumulate fatigue almost everywhere; indeed, only `a10_g95_e20` pushes them into medium fatigue zones without driving them into unsustainable high-fatigue levels. Less balanced choices either converge more slowly or produce erratic fatigue spikes, making them less reliable for a real-time pacing assistant.

4.2 Second Experiment: 1000 Episodes

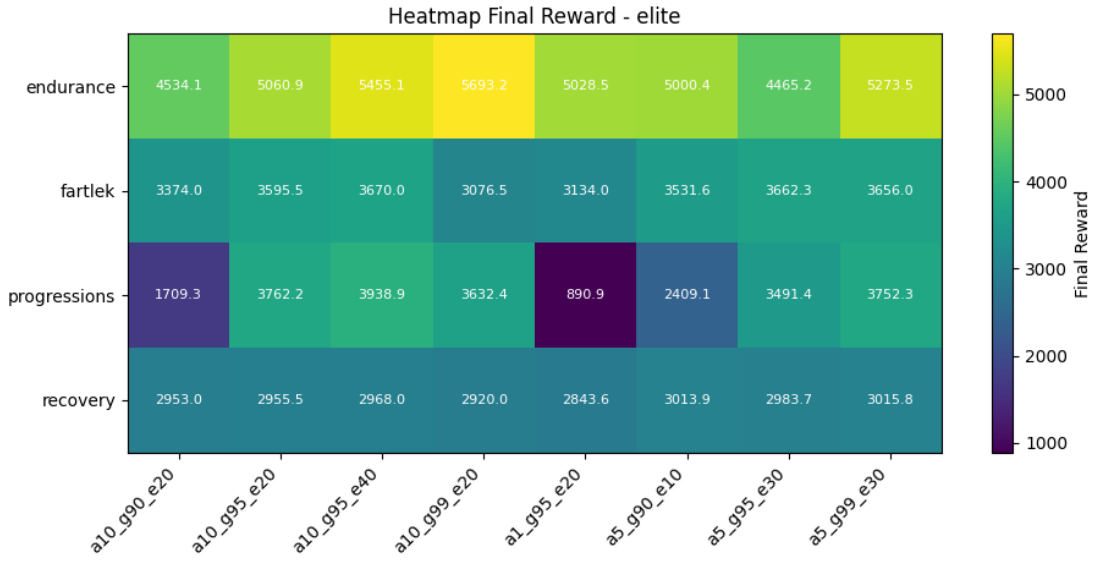
To validate the consistency of the best configurations, or confirm that `a10_g95_e20` is the best combination, a reduced subset of hyperparameter combinations (based on the first



(a) Heatmap for the *amateur* profile after 500 episodes



(b) Heatmap for the *runner* profile after 500 episodes



(c) Heatmap for the *elite* profile after 500 episodes

Figure 2: Heatmaps of total rewards for different athlete profiles after 500 episodes

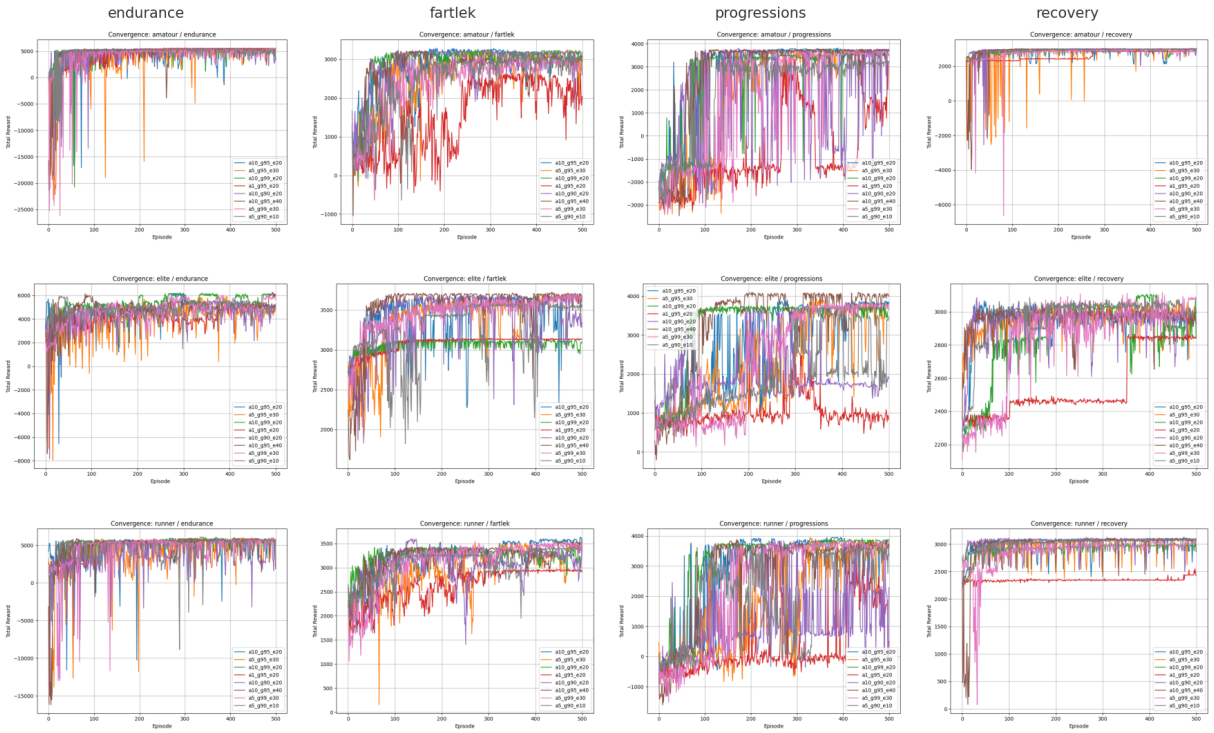


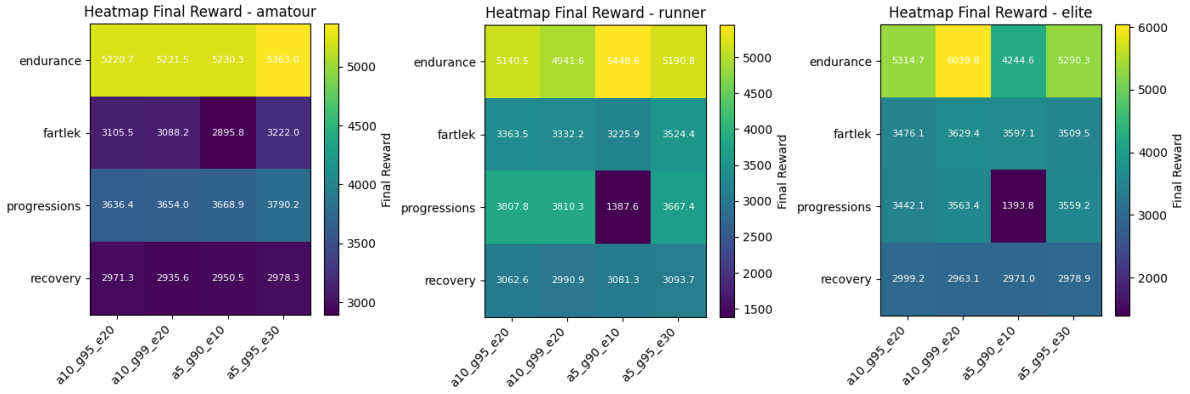
Figure 3: Convergence of Q-values for tested hyperparameter configurations after 500 episodes.

test results) was run with 1000 episodes. This longer training period allowed more stable policy convergence and finer reward differentiation. Even if most of the convergence of the Q-values was already nearly reached, with 1000 episodes the agent was able to explore the state-action space further and refine its policy. The values of the hyperparameters used in this second experiment are shown in Table 3.

Config Label	α	γ	ϵ_0	$\epsilon_{\min}$	**decay**
a5_g90_e10	0.05	0.90	0.10	0.01	0.98
a10_g95_e20	0.10	0.95	0.20	0.01	0.99
a10_g99_e20	0.10	0.99	0.20	0.01	0.98
a5_g95_e30	0.05	0.95	0.30	0.01	0.995

Table 3: Hyperparameter sets for 1000-episode experiments

As shown in Figure 5, all configurations now achieve very smooth, late-episode plateaus, with only minor oscillations remaining in the more complex workouts (*fartlek*, *progressions*). The heatmaps in Figures 4a to 4c reveal that `a10_g95_e20` not only retains its lead but widens the gap: it delivers the highest total reward for every athlete profile. From a physiological standpoint, our *elite* runners benefit most from the balanced focus on immediate and future rewards, achieving peak performance in *endurance* without unnecessary fatigue spikes in recovery. Conversely, *amateurs*, who naturally accumulate fatigue more quickly, see a gentler fatigue profile under this configuration, avoiding severe high-fatigue states even in the toughest segments. The extended training thus confirms `a10_g95_e20` as the most robust and physiologically coherent choice across all scenarios.



(a) *amateur* profile (b) *runner* profile (c) *elite* profile

Figure 4: Heatmaps of total rewards for different athlete profiles after 1000 episodes

5 Final Q-tables and Policies

After choosing the best hyperparameters (`alpha=0.1, gamma=0.95, epsilon=0.2`), a final run with 2000 episodes was performed to ensure the Q-values were fully converged and stable. This final run allowed the agent to refine its policy further, ensuring that it could make optimal decisions during each training session. Different athletes, in the end, respond to the same training program in different ways, adapting their pace and heart rate according to their physiological parameters. The agent is able to adapt its behaviour to the athlete's profile, ensuring that the training is tailored to the athlete's needs. A demonstration of the agent's performance can be seen in the video folder (and also in the figures below 6a, 6b, 6c), where the agent is shown running in all possible combinations of `athlete`, `training program`, and also `different track` and below there

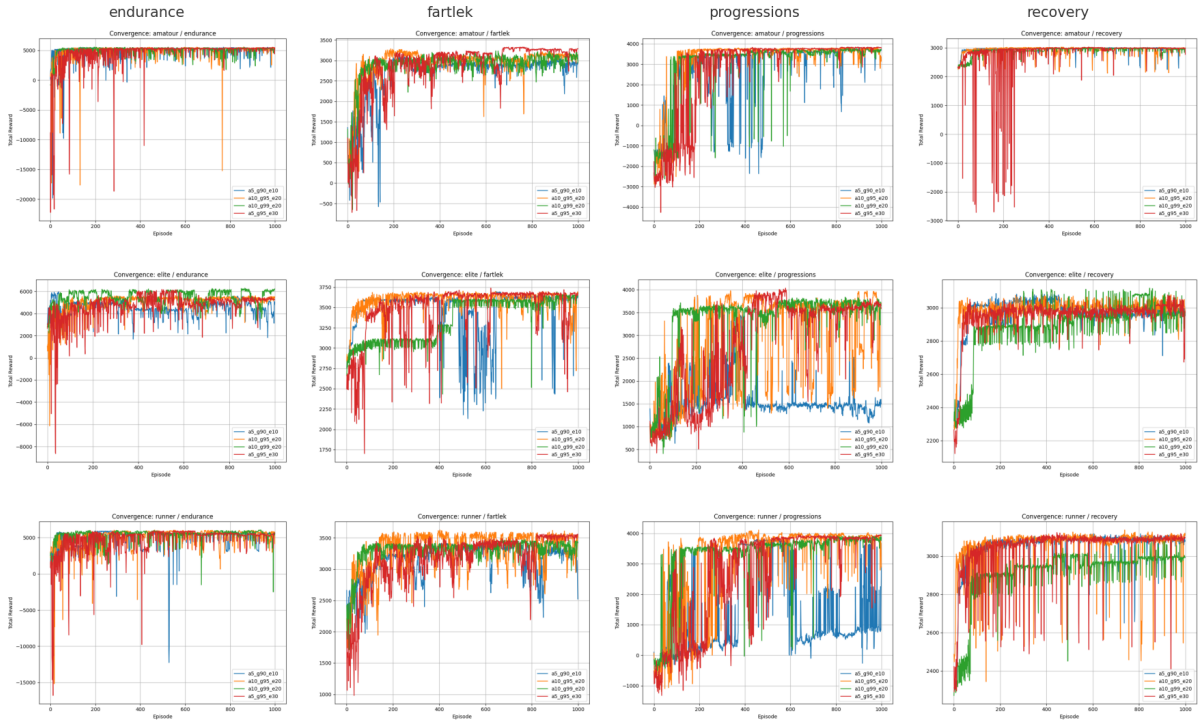
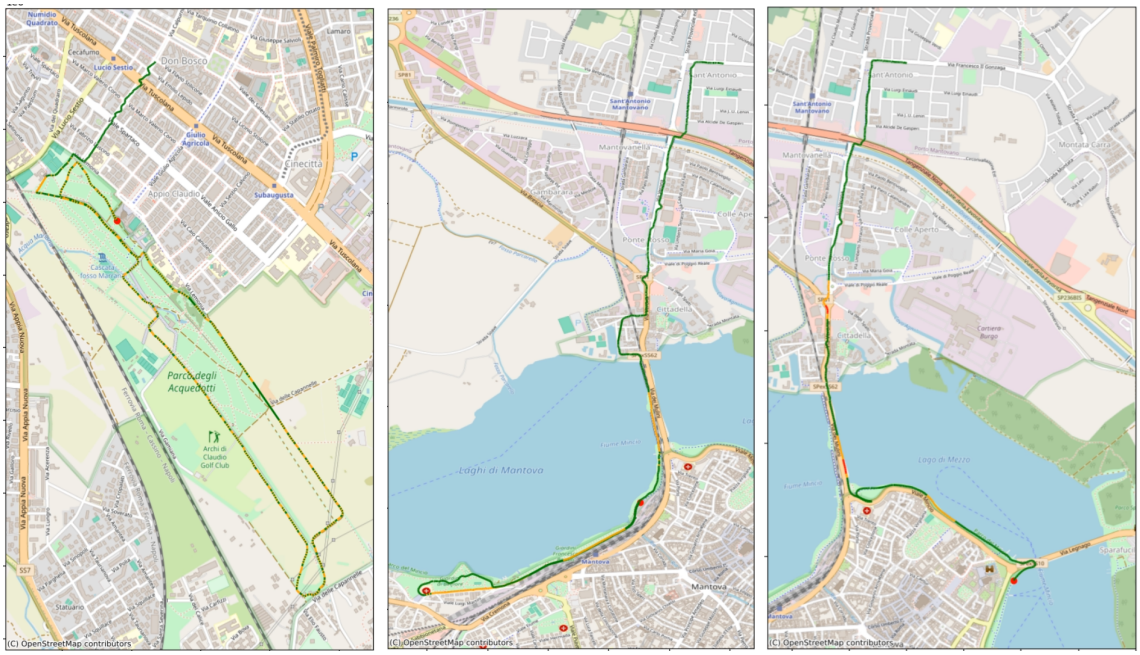


Figure 5: Convergence of Q-values for tested hyperparameter configurations after 1000 episodes.



(a) *amateur* simulation at Parco degli Acquedotti, Rome

(b) *runner* simulation at Parco Belfiore, Mantova of a progression training program

(c) *elite* simulation of the fartlek training program at Mantova, between the three lakes

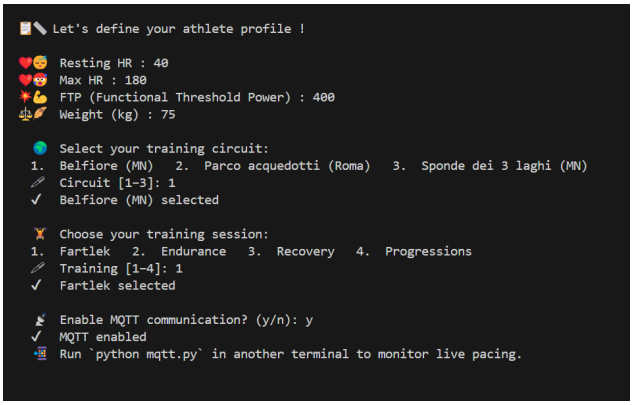
Figure 6: Screenshots from video simulations of the agent's performance across different athlete profiles and training programs. Red lines means high fatigue, yellow means medium fatigue, and green means low fatigue

6 MQTT Communication

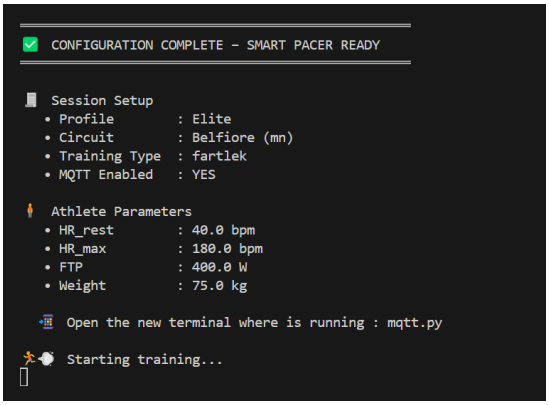
To simulate communication between the user and the ideal smartwatch application (acting as the smart pacer), can be established an MQTT session. This setup allows the user to send data to the smartwatch app, which processes the information (by running the RunnerEnv) and provides feedback, including suggested actions to take during the training.

6.1 Run the simulation

The application is designed to run in a terminal environment, that can be done by exectuing the `main.py` file, which is the entry point of the application. After starting it, the terminal will prompt for information about the athlete to find the closest archetype profile, as defined in sec. 2. The user will be asked to provide the necessary details to build the athlete profile, as explained in sec. 3. After that, the user selects the type of workout to perform, and finally, the application asks whether to create an MQTT session to simulate communication with the smartwatch application. All these steps are shown in the fig. 7a and 7b.



(a) Question about the athlete's parameters in order to choose the right athlete archetype



(b) Recap provided by the application after the user has provided their parameters.

Figure 7: Messages shown in the terminal when starting the application.

If the MQTT session is not created, the application will run the simulation and provide the results inside the folder `training_logs`, with all the information recorded second by second.

The user can then analyze the results and see how the smart pacer would have guided them through the training session.

6.2 MQTT Messages

The communication occurs over the topic `smartpacer/action` using the public broker `broker.emqx.io`. The smart pacer publishes messages at a frequency of 1 Hz, meaning it sends updates every second during the workout session. This frequency is chosen to provide timely feedback to the athlete, allowing them to adjust their pace and actions in real-time based on the smart pacer's suggestions. The payload of each MQTT message is a JSON object containing the following fields:

- `second`: the current second of the workout;
- `phase`: the current phase of the workout;
- `fatigue`: the athlete's current fatigue level;
- `action`: the action suggested by the smart pacer, which can be one of *accelerate*, *hold*, or *ease*.

Each field is represented as a string, accompanied by a relevant emoji to enhance the user experience. An example of messages is shown in figure 8.

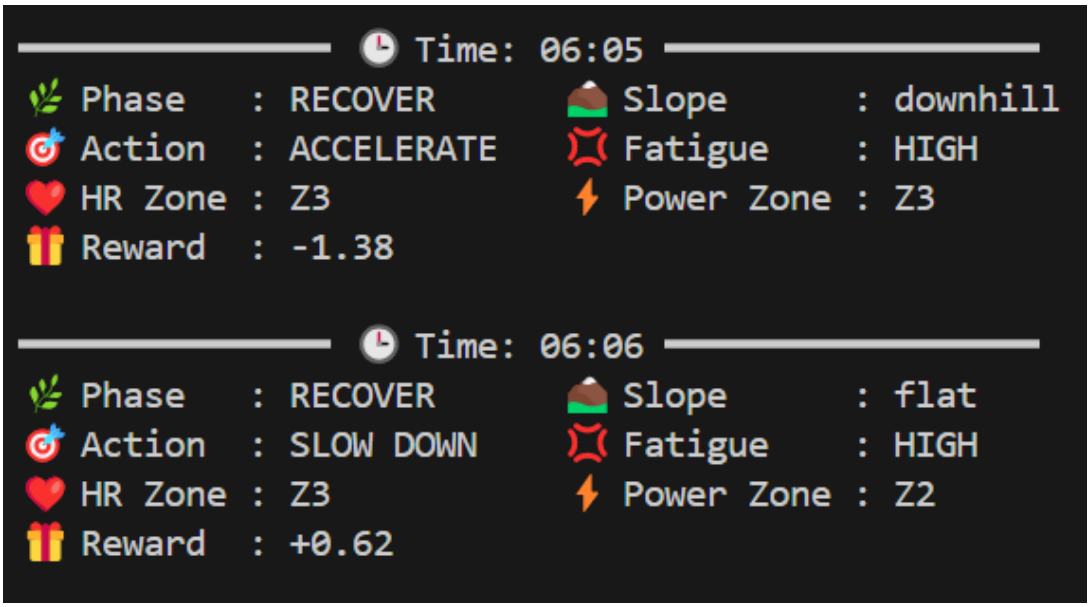


Figure 8: Example of an MQTT message with emojis.

7 Challenges and Future Improvements

7.1 State-space explosion

The current Markov model already spans seven discrete variables, yielding several thousand reachable combinations. Extending realism would require additional factors — e.g. *weather* (temperature, humidity, wind) and *surface type* (asphalt, track, trail, grass). Although essential for real-world fidelity, every new dimension inflates the state complexity.

7.2 Richer athlete profiles

Present profiles depend on basic informations and a FTP valu. A more faithful description would incorporate sex, age, injury history, recent training load, previous-night sleep quality, HRV-derived recovery indices and update them with the evolution of the training process. Such metadata would enable session prescriptions that respect female hormonal cycles, age-related recovery kinetics, and cumulative muscle stress.

7.3 Training program definition

Currently, training program definition follows a uniform logic, adjusting only a few key parameters based on the athlete's level. In practice, however, an elite athlete will face more advanced session types than an amateur: for example, longer endurance sessions, fartlek with more prolonged sprints, or a greater variety of specific workouts. Introducing greater variability and complexity in program generation — differentiating session duration, intensity, and structure according to the athlete profile — would certainly increase the learning task's complexity, but at the same time would better simulate reality and provide more personalized and stimulating recommendations.

7.4 Fatigue and reward refinement

Fatigue and reward terms were the hardest features to properly develop. In order to be even more realistic and coherent with real life behaviours could be implemented also a **Injury-risk term**, which will introduce a weekly load delta component that penalises abrupt increases in training load, reducing overuse-injury likelihood. Another possibl improvement could be the introduction of **sleep quality** and **recovery indices** as additional reward terms, which would allow the agent to adapt its pacing strategy based on the athlete's recovery status. This could be particularly useful for athletes with varying sleep patterns or those recovering from intense training sessions.

7.5 Temporal resolution

All decisions are issued at 1 Hz. Increasing the control loop to 5–10 Hz — or adopting event-driven updates triggered by rapid HR changes would shorten feedback latency and smooth the athlete's perceived guidance.

7.6 Personalised on-line learning

After the first ten sessions, enough data exist to characterise an individual pacing style. Fine-tuning the policy with a small neural network head (e.g. policy-gradient or Soft

Actor Critic) on top of the pre-trained Q-table would capture personal patterns without restarting from scratch. Meta-learning techniques could further shrink the cold-start phase for new athletes.

Addressing the above points will increase both realism, athlete safety but most importantly will move *Smart Pacer* closer to a deployable digital coach that can adapt to individual needs and conditions, providing a more effective and personalized training experience.