Generalization of reinforcement learning environments into natural language using LLM

Erik Dziekonski Bautista (s231752)[1], Alberto Saiz Fernández (s231933)[1], Alejandra Navarro Castillo (s222712)[1]

1 DTU Compute, Technical University of Denmark

Introduction and Objectives

The use of Reinforcement Learning (RL) in closed environments allows the automatic learning of elements towards a predetermined task or goal. Therefore, the main limiting element is the reward function, defined on the basis of the predefined task in each case. Therefore, since it is necessary to define a function for each task to be assigned to the environment, the generalization of a closed environment is not trivial and it is subject to a thorough understanding of the framework and environment parameters.

The aim of the project is therefore to be able to generalize a RL environment using natural language by means of a Large Language Model (LLM), so that the task defined for the environment can be edited by means of a simple text prompt, accessible to any user.

In order to achieve this goal, it will be necessary to first define the static characteristics of the Reinforcement Learning environment in which we will apply our models, both the framework and the training algorithms that we will use. Once the environment has been defined and tested, the next step will be to train a LLM using the static characteristics of the chosen framework (observation and action space), so that, from a text prompt with the description of an action, it will be able to generate a reward function associated in order to train the environment to perform it.

RL Gym Framework and methodology

- The chosen RL framework is the **Humanoid-v4** environment of MuJoCo (Multi-Joint dynamics with Contact) engine of the Gym API for reinforcement learning.
- The act of **running** by a humanoid was defined as a basic task to test the environment. The reward function implemented is based on the x-position, orientation and x-velocity of the torso of the humanoid. The reward function substitutes the default function defined by the environment by means of the TransformWrapper library.
- The agent's policy used to train our model is **MLP policy**.
- Four potential training algorithms (**SAC**, **TD3**, **A2C** and **PPO**) were compared for the task. These algorithms differ in how they estimate the expected cumulative reward and update the agent's policy. The reward obtained in tests carried out on trained models for each algorithm is the criterion for the comparison.
- The hyperparameters used in the training were chosen as 200 episodes and 5000 timesteps in the learning process. The model with the highest average reward through the training is validated afterwards in a 500-frame time lapse.

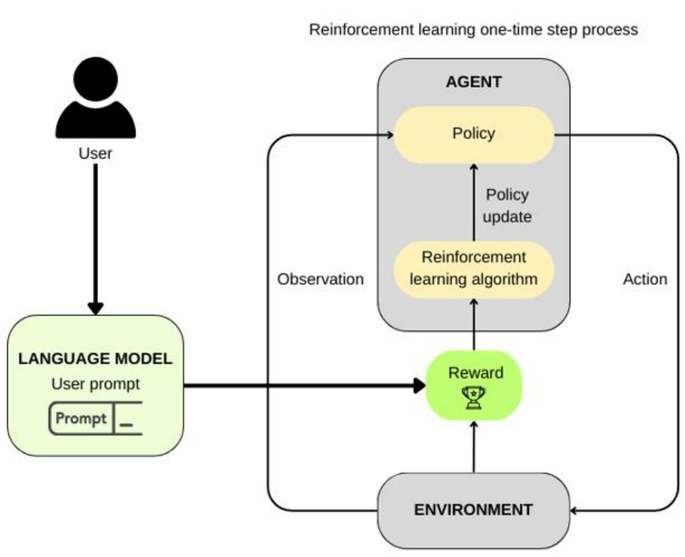

Figure 1: Model architecture representation.

Stable-Baselines3 RL Algorithm comparison

As stated in the methodology, the RL algorithms to test in our framework are SAC, PPO, A2C and TD3. All algorithms will use MLP as its policy and will be implemented through the Stable-Baselines3 library. Metrics show that the most suitable algorithm for our interests is SAC.

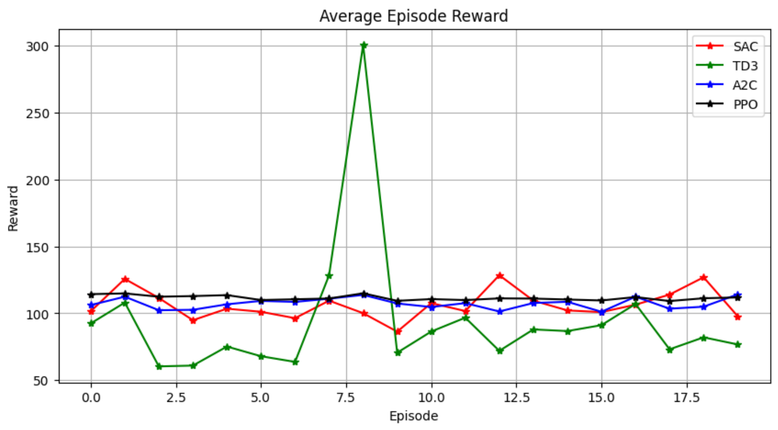

Figure 2: Average episode reward per training algorithm

Algorithm	Avg Episode Reward	Avg Length	Regression Slope
SAC	**106.278**	**21.390**	**0.250**
TD3	94.369	17.139	-0.235
A2C	107.29	21.859	0.019
PPO	111.52	22.562	-0.162

Table 1: Algorithm performance

LLM implementation

The implementation of the generalization large language model (LLM) is based on the LangChain framework. It runs on *gpt-4-1106-preview* model that has been fine-tuned with the observation space of the environment, the number of steps of each episode and examples of reward function.

The output of this model is a string with the generated reward function. Using a compiler, the string can be converted into a python reward function that can be easily implemented in the environment. The buffer memory of LangChain has been included as an extra feature of the LLM model too.

An API connected to the model accessible from the QR code (3) has been developed for obtaining a reward function from a natural language prompt. This API could be easily extensible to an application on RL training and rendering from a simple text prompt.

Trained Basic Task Validation

The performance of the simple task of **running** using the reward function human-coded and the one LLM-generated is compared in the following figures. Full performance can be visualized on QR code (1).

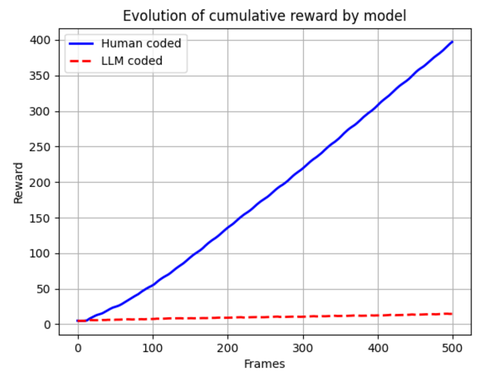

Figure 3: Cumulative reward over time in test

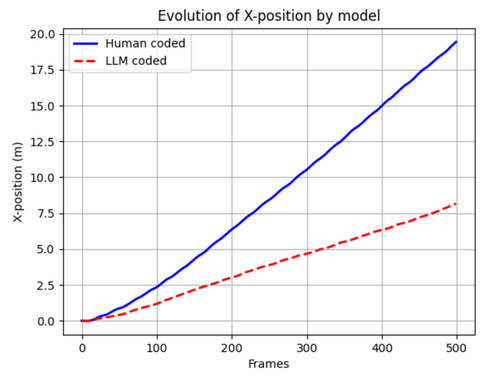

Figure 5: X-Position over time in test

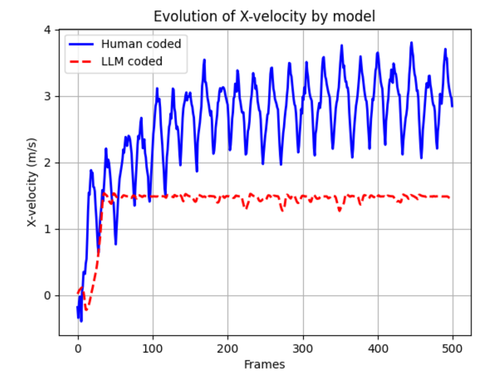

Figure 4: X-Velocity over time in test

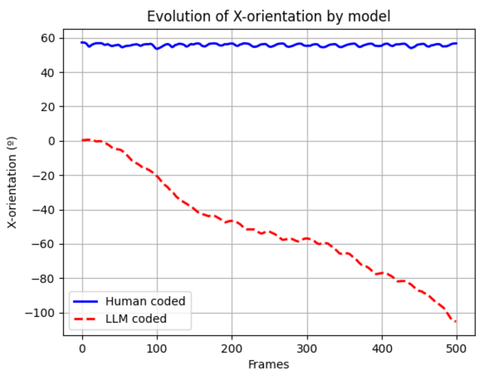

Figure 6: X-Orientation over time in test

LLM Reward function validation

In order to validate the performance of the generalized model, new tasks are defined from natural prompt. Figures 7 and 8 show the successful performance of a "walk" (i.e. running slower) task.

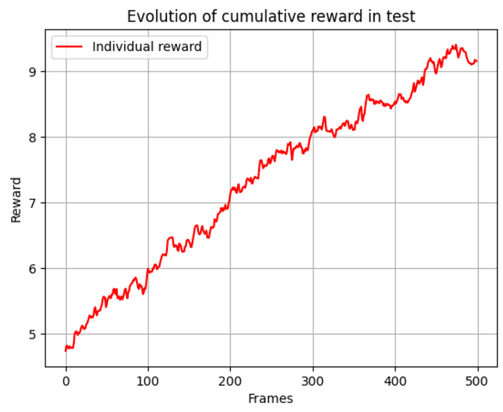

Figure 7: Cumulative reward over time in test

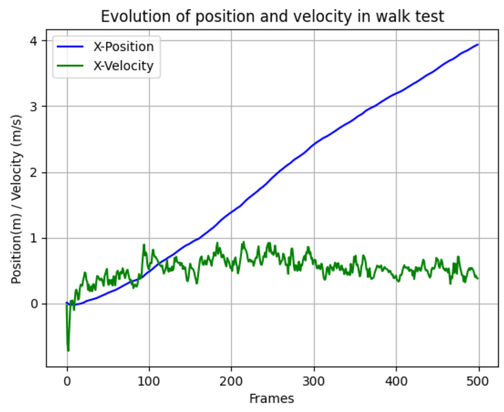

Figure 8: Position and velocity over time in test

The task of "jumping" was also given to our trained LLM model. The humanoid was finally not able to complete a jump. The frames of the rendered humanoid performing this attempt can be seen in the following pictures.

Figure 9: Frames of failed jump task

Full performance of both validation tasks can be seen on QR code (2).

Demos

Running Human vs LLM

LLM Validation Tasks

LLM-Demo API