AutoRoute: LLMs Instruct Reinforcement Learning Agent

Shuo Yang [1]

Abstract

Quadcopter is a simplified quad-rotor aircraft in reinforcement learning (RL) training, which can keep in a constant position. By modifying the reward structure, we developed a directionally cruising quadrotor, dubbed QuadRoute. With the advent of large language models (LLM), complex problem-solving tasks can be approached in a human-like manner. Despite the power of these models, their output often carries inherent uncertainties, which require effective regulation. In our work, we proposed two novel methods to standardize the output of the LLM, thereby making it more applicable for our use-case. By integrating the LLM-guided instructions, our simplified QuadRoute showcased the ability to accomplish sophisticated tasks, including obstacle avoidance and maintaining a constant cruise speed. This study sheds light on the potential of RL and LLM integration in autonomous systems, paving the way for future research and development in this area.

1. Introduction

Autonomous navigation has long been an aspiration within the realm of machine learning. The burgeoning field of reinforcement learning (RL) is one such avenue that researchers have been exploring to tackle this problem. However, despite significant advancements, pure reinforcement learning approaches alone have yet to demonstrate a comprehensive solution for autonomous driving tasks.

Intuitively, autonomous navigation seems to necessitate elements of human intelligence. Consider, for instance, the need to identify the most optimal route or to avoid obstacles while navigating complex environments. These are tasks that humans navigate with relative ease, guided by experience and cognitive acuity. Replicating these abilities in an autonomous system, however, is nontrivial. The complexity of these tasks makes them challenging to learn within a simulated environment, as they often involve intricate decision-making that transcends the simplistic stimulus-response paradigm.

Building on QuadRoute's foundation, we integrated a Large Language Model (LLM) to serve as an instructor within the testing environment. The LLM takes as input the environment information and generates a route plan to successfully navigate the quadcopter through the environment while avoiding obstacles.

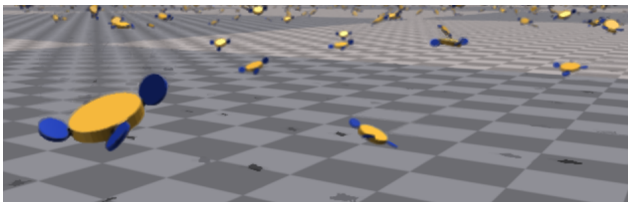

Figure 1. QuadRoute Isaac Gym

For this integration to be effective, we had to regulate the LLM's output. This was achieved through a two-pronged approach: the use of prompts to guide the model's output and employing the embedding of the output to better understand it. The first technique helps in shaping the model's responses to align with the task's requirements, while the latter provides insights into the context and content of the output.

In the subsequent sections of this paper, we delve into the specifics of the QuadRoute model, the method adopted to integrate the LLM as an instructor, and our experimental results. By blending the capabilities of RL agents and the human-like cognition of LLMs, we hope to offer a novel approach to autonomous navigation tasks.

2. Background

The emerging domain of autonomous systems and artificial intelligence has seen a surge in practical applications. This paper focuses on two such advanced technologies: Quadcopters in reinforcement learning tasks, and Large Language

[*]Equal contribution [1]Department of Zhiyuan College, Shanghai Jiao Tong University, Shanghai, China. Correspondence to: Shuo Yang <andy_yang@sjtu.edu.cn>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

Models capable of generating human-like text.

2.1. Quadcopters and Isaac Gym

Quadcopters, characterized by their four rotors, have become a popular subject in the realm of reinforcement learning tasks. Isaac Gym, developed by NVIDIA, is a physics simulation environment ideal for reinforcement learning. One such RL task within Isaac Gym is the Quadcopter task. This example trains a simplified model of a quadcopter to reach and hover near a specific position. The model is generated procedurally, eschewing rotating blades and using thrust forces applied to flat-cylinder 'rotor' bodies to achieve lift. This serves as a compelling illustration of the use of LOCAL SPACE forces. Control over the pitch and roll of each rotor is achieved through DOF (Degrees of Freedom) position targets. This task offers a useful platform to investigate and test RL agents' performance and navigation capabilities in a three-dimensional space.

2.2. Large Language Models: GPT-4 and Vicuna

Large language models such as GPT-4 have revolutionized the field of natural language processing. GPT-4, developed by OpenAI, exhibits an impressive ability to generate coherent and contextually relevant text, performing tasks that range from translation and summarization to answering complex questions and even writing essays. Its versatile capability is realized through a transformer-based architecture that uses layers of self-attention mechanisms to understand and generate text.

Building on the impressive capabilities of GPT-4, the Vicuna model was developed. Vicuna-13B is an open-source chatbot fine-tuned on user-shared conversations collected from ShareGPT. It has demonstrated excellent performance, achieving over 90% quality of renowned models such as OpenAI's ChatGPT and Google Bard in most cases. Furthermore, the cost of training Vicuna-13B is surprisingly affordable, and the model, including code and weights, is publicly available for non-commercial use. Fine-tuning Vicuna with thousands of user-shared ChatGPT conversations has led to the generation of more detailed and well-structured responses compared to other models.

2.3. Language Model Embedding

Language models harness the power of transformer layers to convert text into a fixed-size vector, known as an embedding. Each transformer layer captures different aspects of the text, from the basic grammatical structure at the lower levels to the semantics and context at the higher levels. By passing text through multiple transformer layers, we can condense a wide variety of information into a single vector. This high-dimensional vector is a condensed representation of the text, preserving as much relevant information as possible. The embedding not only captures the raw information present in the text but also the subtle, contextual meaning that underpins human language. These embeddings are essential for many downstream tasks such as text classification, sentiment analysis, and question answering.

3. QuadRoute

QuadRoute is a reinforcement learning task designed to train an agent to perform specific navigational duties within a three-dimensional space.

3.1. Task Goal

The primary objective of QuadRoute is straightforward: given a set of target points in a three-dimensional space, the agent is required to navigate through these points in the order provided, adopting a cruise-like mode of movement. Upon successful completion of one cycle, the agent starts over from the beginning, thus introducing a cyclical pattern in its navigational routine.

3.2. Environment Setting

The environment for QuadRoute is defined within a carefully configured simulation. Here are some key features:

- **Physics Engine**: The physics engine responsible for simulating the environment dynamics and agent's interactions is customized and appropriately defined.
- **Environment Parameters**: The environment accommodates a specific number of concurrent instances (set by ‘numEnvs‘) with a certain spatial separation (‘envSpacing‘). Each episode of the simulation has a maximum length (‘maxEpisodeLength‘), after which it is reset.
- **Observations and Actions**: The environment employs a mechanism to clip extreme values in the observations and actions to ensure stability. Clipping limits are set by ‘clipObservations‘ and ‘clipActions‘.
- **Simulation Parameters**: The simulation has a defined time step (‘dt‘) for discrete simulation updates and runs two computational substeps (‘substeps‘) within each time step. The simulation's gravity is set to mimic Earth's gravity.
- **Physics Parameters**: The simulation is configured to run on a GPU if available (‘use gpu‘) and utilize a number of threads specified by ‘num threads‘. It has settings for solver type, position iterations, and velocity iterations, among others, to ensure accurate physics simulation.

- **Task Parameters**: The task has a randomization parameter, set to 'False' here, which, when set to 'True', could potentially allow for variation in initial conditions or other aspects of the task for each episode.
- **Other Settings**: There are additional settings such as 'gpt instruction' and 'random init points', currently set to 'False'. 'gpt instruction' controls whether the LLM provides instruction to the agent, and 'random init points' determines if the initial target points for the agent are randomized at the start of each episode.

These settings collectively provide the necessary framework for the QuadRoute task and establish a challenging but navigable environment for the agent.

3.3. QuadRoute Agent Design

The QuadRoute agent is designed with particular attention to both its environmental perception and the mechanisms driving its actions. This design allows the QuadRoute agent to operate effectively within a three-dimensional space, reaching defined target points in order and avoiding designated danger points.

3.3.1. Observations

Observations form the basis of the agent's environmental perception. In this context, observations are calculations made based on several key factors:

- **Root Position**: The current position of the Quadcopter within its three-dimensional space. This helps the agent understand its relative location to the target points.
- **Target Points**: These are the 3D coordinates that the Quadcopter aims to reach. These points serve as goals and guide the agent's navigation.
- **Current Speed**: The agent also observes the current speed of the Quadcopter. This information is crucial in determining and adjusting the velocity of the Quadcopter according to different scenarios.

These observations provide a comprehensive snapshot of the Quadcopter's current state and the surrounding environment. They ensure the agent has all necessary information to successfully navigate through target points and avoid danger zones.

3.3.2. Actions

The QuadRoute agent is designed to interact with its environment through a set of actions. These actions determine the movement and orientation of the Quadcopter within its 3D space. They include adjusting the Quadcopter's speed, changing its pitch, roll, and yaw, and altering its altitude. These actions allow the agent to effectively control the Quadcopter's flight path and reach target points while avoiding danger points.

3.3.3. Initialization

During the initialization of the QuadRoute agent, a set of parameters such as the number of target points, the positions of the target points, and danger points are defined. These parameters provide the foundational setting for the agent's operation.

4. Method

Our methodology primarily revolves around the design of two reward systems, the application of Large Language Model (LLM) instruction, and regulation of LLM output. Let's delve into each aspect.

4.1. Reward Design

In designing the reward systems, we sought to find a balance between providing sufficient complexity to handle real-world challenges, while also maintaining tractability for training. This resulted in two reward systems: one for training (without danger points) and one for testing (with danger points and pseudo points).

4.1.1. Training Reward System

In this design, the rewards are dependent on the agent's distance to the target point. The closer the agent gets to the target point, the higher the reward it receives. Moreover, upon reaching a target point, the agent is given an additional reward. This reward structure encourages the agent to navigate towards and reach the designated target points effectively.

$$r_t = \frac{1.0}{1.0 + d_{target}^2} \tag{1}$$

$$r_t = r_t + R \quad \text{if } d_{target} < \text{threshold} \tag{2}$$

4.1.2. Testing Reward System

This system introduces more complexity by including danger points and pseudo points. Here, the agent is penalized for getting too close to a danger point, reflecting the real-world requirement of avoiding obstacles. On the other hand, pseudo points serve as guidance for the agent to circumvent these danger points. However, reaching or getting close to pseudo points does not increase the agent's reward. This ensures the agent is motivated to focus on the actual target points, using pseudo points merely as a means to safely

navigate the environment.

$$r_t = \frac{1.0}{1.0 + d_{target}^2} \tag{3}$$

$$r_t = r_t + R \quad \text{if } d_{target} < \text{threshold} \tag{4}$$

$$r_t = r_t - R \quad \text{if } d_{danger} < \text{threshold} \tag{5}$$

4.2. LLM Instruction

The LLM's role in our methodology is to guide the agent in avoiding danger points during its navigation process. We use an OpenAI GPT-4 model to infuse pseudo points into the series of target points during initialization. These pseudo points serve as instructions for the agent, aiding it in navigating safely around the danger points. However, the agent receives no reward for reaching or getting close to these pseudo points, ensuring its primary focus remains on reaching the actual target points.

```
if self.cfg['setting']['gpt_instruction']:
    self.target_points, self.pseudo_index = gpt4_instruct(self.real_points, self.danger_points, self.device)
else:
    self.target_points = self.real_points
```

4.3. Regulating LLM Output

To make the best use of the LLM's instructions, we need to ensure the output is conducive for the agent's navigation process. For this, we employed two different strategies for GPT-4 and Vicuna.

4.3.1. GPT-4

We used prompt engineering to guide the LLM's output. This process ensures the output from GPT-4 directly presents a new series of target points, including clear indications of the pseudo points added by the LLM.

For example, we can add pseudo points with prompt:

You are now a route planning assistant in a three-dimensional space. You are given a series of three-dimensional coordinates [x, y, z], which include points that must be passed through in order and points that must be avoided. The points that must be passed through in order are enclosed in square brackets, and the points that must be avoided are enclosed in another set of square brackets. For example, if you need to pass through [0.0,0.0,1.0], [1.0,1.0,2.0] and avoid [0.5,0.5,1.5], the user would input [[[0.0, 0.0, 1.0], [1.0, 1.0, 2.0]], [[0.5, 0.5, 1.5]]]. You need to provide a new set of points that must be passed through in order. When passing through these points in order, the distance to all points to be avoided should be greater than 0.2. Also, you need to indicate the indices of the points you added. The new ordered points are enclosed in square brackets, and the indices are enclosed in another set of square brackets. Please do not output any other content. For example, if the new sequence is [[0.0, 0.0, 1.0], [0.0, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 1.0, 2.0]], and the new points added are the first and third points (zero-based), then your output should be [[[0.0, 0.0, 1.0], [0.0, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 1.0, 2.0]], [1, 2]].

Then we can use the prompt to get new target points:

```
resp = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
            {"role": "system", "content": prompt},
            {"role": "user", "content": "[[[0.0, 0.0, 1.0], [1.0, 1.0, 2.0]], [[0.5, 0.5, 1.5]]]"},
            {
                "role": "assistant",
                "content": "[[[0.0, 0.0, 1.0], [0.0, 1.0, 1.0], [1.0, 1.0, 1.0], [1.0, 1.0, 2.0]], [1, 2]]",
            },
            {"role": "user", "content": str([real_points.tolist(), danger_points.tolist()])},
        ],
    temperature=0.3,
)
```

4.3.2. VICUNA

We used embedding cosine similarity to assess the LLM output. We first convert the output into an embedding. Similarly, we convert standard actions into embeddings as well. We then compute the cosine similarity between these embeddings to identify which action most closely aligns with the LLM's output. This process allows us to understand and utilize the Vicuna output effectively.

```
def cosine_similarity(vec1, vec2):
    try:
        return 1 - cosine(vec1, vec2)
    except:
        print(vec1.shape, vec2.shape)


def get_embedding_from_api(word, model="vicuna-7b-v1.1"):
    if "ada" in model:
```

```
        resp = openai.Embedding.create(
            model=model,
            input=word,
        )
        embedding = np.array(resp["data"
            ][0]["embedding"])
        return embedding

    url = "http://localhost:8000/v1/
        embeddings"
    headers = {"Content-Type": "application
        /json"}
    data = json.dumps({"model": model, "
        input": word})

    response = requests.post(url, headers=
        headers, data=data)
    if response.status_code == 200:
        embedding = np.array(response.json
            ()["data"][0]["embedding"])
        return embedding
    else:
        print(f"Error: {response.
            status_code} - {response.text}"
            )
        return None
```

5. Experiment

5.1. Design

Our experimental design primarily encompasses a training phase and a testing phase. In the training phase, we randomly select seven points in the space as target points and then allow the QuadRoute Agent to perform cruising training for a specified period. The objective of this phase is to let the model grasp basic flight control and navigation skills, understanding how to effectively transit between target points.

The testing phase is subdivided into three stages: no danger points stage, random danger points stage, and path danger points stage. The no danger points stage serves as a baseline test, assessing the performance of the QuadRoute Agent under risk-free conditions. The random danger points stage incorporates danger points randomly distributed in the environment. The objective of this stage is to evaluate the adaptability of QuadRoute Agent in a complex environment and its capacity to evade risks. Finally, the path danger points stage embeds a danger point between adjacent target points. The goal of this stage is to simulate the potential challenges encountered in actual flights, such as obligatory avoidance of obstacles.

In these three stages, we tested both the QuadRoute Agent with LLM instruction (LLM-Agent) and the QuadRoute Agent without LLM instruction (Baseline-Agent). The design rationale behind the LLM-Agent is to improve the flight strategy of the Agent, particularly its flight decision-making in dealing with complex and risky environments, through the guidance of the LLM.

5.2. Results

The experimental results demonstrated that the LLM-Agent outperforms the Baseline-Agent in all stages. Specifically, in the no danger points stage, the performance of the LLM-Agent and Baseline-Agent was comparable, which aligns with expectations as the tasks in this stage are relatively simple and do not require complex decision-making.

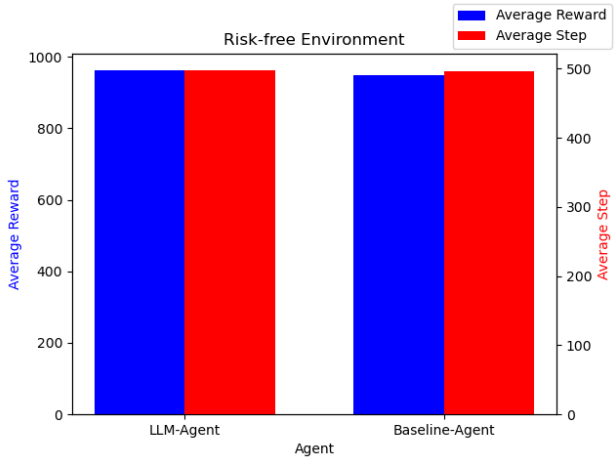

Figure 2. Risk Free Env

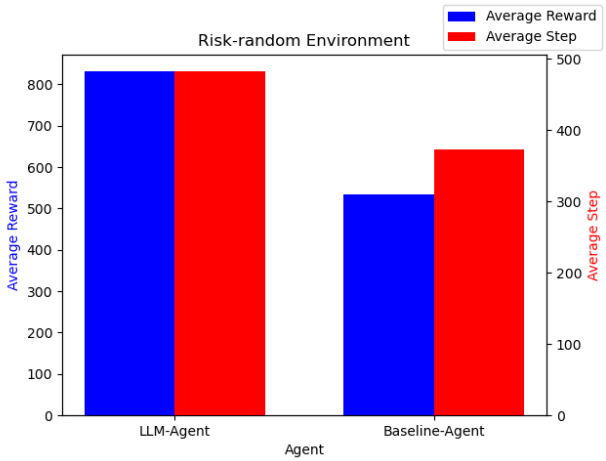

Figure 3. Risk Random Env

However, in both the random danger points stage and the path danger points stage, the LLM-Agent markedly surpassed the Baseline-Agent. Especially in the path danger

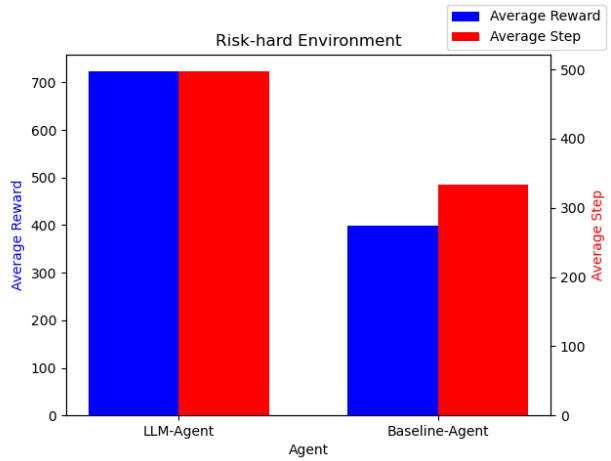

Figure 4. Risk Hard Env

points stage, the advantages of the LLM-Agent were even more pronounced, indicating that LLM guidance has a significant effect on handling complex flight scenarios and enhancing flight safety. These results confirm our hypothesis: the guidance of large language models can effectively enhance the autopiloting capability of quadcopters.

6. Conclusion

This paper presents AutoRoute, a novel quadcopter control framework that seamlessly integrates the capabilities of a Large Language Model (LLM) and a simple Reinforcement Learning (RL) model. Our work introduces the concept of QuadRoute, a simplified quadcopter task designed to reach and hover near a sequence of predetermined positions. We leverage the LLM to provide high-level flight instructions based on the environment information, guiding the QuadRoute Agent in effectively navigating complex flight scenarios.

We design an innovative reward system, distinguishing the training and testing phases, to foster effective learning and better evaluate the model's performance in real-world scenarios. Our work also contributes two methods for regulating the LLM output, one through prompt engineering for GPT-4 and another through embedding cosine similarity for Vicuna, ensuring the LLM's instructions are relevant and beneficial for the quadcopter navigation task.

The experimental results validate our approach, demonstrating that the LLM-instructed QuadRoute Agent significantly outperforms the baseline model in handling complex flight scenarios, especially when danger points are placed along the path. This outcome attests to the efficacy of combining the superior decision-making ability of LLM with the adaptability of RL in the context of autopiloting quadcopters.

Our study provides valuable insights into the potential of LLM-instructed RL agents in the realm of autonomous navigation. The notable performance of the LLM-instructed QuadRoute Agent suggests a promising direction for autonomous driving: coupling a basic RL model capable of basic control with an LLM that can provide higher-level insights and strategies. This simple yet powerful approach offers a practical and effective solution to complex autonomous navigation tasks, paving the way for more advanced developments in the field of autonomous driving.

Acknowledgements

This paper is the course project of CS3316.

References

Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. Guiding pretraining in reinforcement learning with large language models. 2023.