Universiteit Leiden
The Netherlands

Bachelor DSAI

Large Language Model as model
in Model-Based Reinforcement Learning

Alexander Scheerder

Supervisors:
Aske Plaat & Álvaro Serra-Gómez

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

08/07/2025

Abstract

This thesis aims to improve model-based reinforcement learning (MBRL) using large language models (LLMs). In this thesis, a new MBRL algorithm: Dyna-LLM, is made which uses the LLM: Tiny-Llama, as transition and reward function. In the main experiment, Dyna-LLM is compared with Q-learning: a model free RL algorithm, and Dyna-Q: a model based RL algorithm. Afterwards, an additional experiment comparing the LLM outputs of Tiny-Llama and Qwen3 is done to explain the results of the main experiment. The results show that Tiny-Llama does not improve sample efficiency as transition and reward function of MBRL. Additionally, large LLMs are found to be likely better than small LLMs in model-based RL. Furthermore, Tiny-Llama appears to not understand environment mechanics, given the prompts used in this experiment. While not improving the classical RL methods, this study provides insights in the process of aiding MBRL with LLMs.

The code can be found at GitHub[1], and can be downloaded using the following command in the command prompt when it is opened at the desired location:

`git clone https://github.com/Alex3124536745869708/thesis`.

[1] https://github.com/Alex3124536745869708/thesis

Contents

1 Introduction

Reinforcement learning (RL) is used to continuously improve the RL agent's behaviour, with the goal of optimal decision making [UXL+22]. Tabular RL methods often struggle with high-dimensional state spaces and complex environments [Rao00]. In these cases, during the training process, a high amount of sampling is needed to gather enough data to make accurate predictions. Because of the high amount of sampling, it is important to reduce the cost of such systems.

Since Large language models (LLMs) have extensive pre-trained knowledge and have high-level generalization capabilities, LLMs could be used to improve the RL training process [CZC+24]. For example, they can be used to generate environments, using the reasoning capabilities of LLMs, where the RL agent can learn and improve skills that it is weak at [ZCL+24].

In the discussed related work, LLMs with more than 7 billion parameters are tested to aid the RL process and have shown to improve the sampling efficiency [BBK+24], [KSR+23], [ABB+22], [DYZ25], [LHZ+24]. A small LLM such as Tiny-Llama[2] can have benefits, as this LLM is light weight and allows to be run offline on systems that have limited computation availability [Lee24]. The lower system requirements make small LLMs well suited for robotics developers and useful for applications in remote settings such as caves, where there is no internet connection. Because of this, testing if a small LLM can help improve environment sampling efficiency in RL is important.

1.1 Thesis overview

This chapter contains the introduction; Section 2 discusses related work; Section 3 describes the main experiment; Section 4 discusses their outcome and an additional experiment; Finally, Section 5 provides conclusions and discusses future research.

This bachelor thesis is a project at LIACS and is supervised by Aske Plaat and Álvaro Serra-Gómez.

1.2 Problem statement

The problem statement is:

PS: Can LLM improve model-based reinforcement learning?

In order to answer the problem statement the following research questions (RQ's) are used:

RQ1: Can Tiny-Llama improve sample efficiency as the Transition and Reward function of model-based reinforcement learning in a grid world problem?

RQ2: Are large LLMs better than small LLMs in model-based reinforcement learning?

RQ3: Does Tiny-Llama understand the environment mechanics, given the prompts used in this experiment?

RQ1 will be answered using the results of the main experiment. **RQ2** and **RQ3** are created after the main experiment and are used to explain the results of the main experiment.

[2] https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0

2 Related work

In this chapter, related work related to the research questions will be introduced.

2.1 Background

In this subsection, background information is introduced. In order to compare the effect on sampling efficiency of a newly constructed algorithm using a small LLM which I call Dyna-LLM, Q-learning [WD92], a temporal difference (TD) based reinforcement learning (RL) algorithm is used. As well as the model based RL algorithm Dyna-Q [SB20]. Q-learning is chosen to compare with the Dyna-LLM algorithm, because Q-learning is a basic TD RL algorithm, providing a baseline in the comparison. Dyna-Q is chosen as well, because Dyna-Q is a basic Model-based RL algorithm that is almost the same as Dyna-LLM. Given that Dyna-LLM is based on Dyna-Q, comparing Dyna-LLM to Dyna-Q can provide specific insights.

2.1.1 MDP

A Markov-decision process (MDP) [Bel57] is a mathematical model that can be used to describe the environment of a sequential decision problem. On the other hand, RL aims to find the optimal solution to a sequential decision problem using the MDP. A sequential decision problem is a problem where the eventual outcome of the agent's actions are influenced by a sequence of decisions. An MDP consists of:

- a state space representation denoted as S
- an action space representation denoted as A
- a transition function $T(s'|s,a)$, $T : S \times A \to p(S)$
- a reward function $r(s,a,s')$, $r : S \times A \times S \to \mathbb{R}$
- a discount factor γ, $\gamma \in [0,1]$,

where S is a discrete set of size $|S|$ containing all the possible states which are denoted as s. A is a discrete set of size $|A|$ and contains all the possible actions. $T(s'|s,a)$ gives the probability of the next state denoted as s' given a certain state s action a combination. T is of size $|S| \times |A| \times |S|$. $r(s,a,s')$ gives the reward given a certain (s,a,s') tuple. r is of size $|S| \times |A| \times |S|$. Discount factor γ is a constant and regulates how much influence later rewards have. When γ is 0, the agent only uses the immediate reward. When γ is 1, the later rewards have as much influence as the early rewards.

2.1.2 Q-learning

In tabular reinforcement learning (RL), the agent learns a task by trying actions at certain states. By trying actions, the agent receives rewards that it is trying to maximize [SB20]. By giving the tools to the RL agent to solve the problem, but not give the solution to the problem, the RL agent aims to find the optimal solution. To find the optimal solution Q-learning uses the functions of the MDP. RL-tasks can have imperfect information, making the outputs of the T and r functions

unknown to the agent before sampling. Thus, while the RL-agent knows the S, A, and γ of the MDP, the T and r are given by the environment after interacting with it.

To find the optimal policy, Q-learning fills in the Q-table with estimates of the highest cumulative reward on a state action combination. Because the Q-table consists of the value per state action combination, the agent can simply choose the best action to do at a state by choosing the action that corresponds to the highest value in the Q-table given said state. The update rule for the Q-table, defining Q-learning is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha * [r + \gamma * \max_{a'} Q(s',a') - Q(s,a)],$$

where Q denotes the Q-table with states and actions as dimensions, r denotes the reward as a result of the agent sampling action a on state s, α denotes the learning rate, s' denotes the next state, a' denotes the next action, γ denotes the discount parameter and $\max_{a'} Q(s',a')$ denotes the best Q-table value of the next state.

The Q-value is updated by adding the learning rate α times the temporal difference error (TD error). The most important part of the TD error is the difference between the estimated best next value ($\max_{a'} Q(s',a')$) and the current value estimate ($Q(s,a)$). The TD error represents how much better or worse the result of the action was compared to what was previously expected. If the result of an action is better than expected, the Q-value increases. If it's worse, the Q-value decreases. When the Q-table (the table of Q-values) converges, the Q-values change very little. The little change in Q-values means that the agent is finished with collecting rewards in order to find the optimal solution to the task.

The γ influences how much impact the later rewards have on the earlier values in the Q-table. With a γ of value: 1 having the most impact and a γ of value: 0 having the least. Because of this, increasing γ increases the importance of long-term rewards, which is beneficial when trying to find the most valuable path. The learning rate α controls how much the Q-table values are adjusted.

A high α value updates the Q-table value more drastically which lets the Q-table converge possibly more quickly. However, this also makes the algorithm more unstable. Additionally, a low α value makes the learning update more stable but less drastic and thus makes the algorithm need more timesteps to converge.

The agent performs the optimal solution by following the optimal policy. A policy determines what action should be performed at a certain state by the agent. The ϵ-greedy policy is used and has the following formula:

$$\pi_{\epsilon-greedy}(s,a) = \begin{cases} 1-\epsilon, & \text{if } a = \arg\max_{a \in A} Q(s,a) \\ \frac{\epsilon}{|A|}, & \text{otherwise} \end{cases},$$

where A is the set of all possible actions on the state s, ϵ is the chance the best action is not explicitly chosen, $\arg\max_{a \in A} Q(s,a)$ is the best action and a is an action. Here the $\pi_{\epsilon-greedy}$ denotes the action selection policy and the chance a certain action is chosen. The agent explores with a rate of ϵ and exploits with a rate of 1-ϵ. When an agent selects an action, the action is given by the policy. When exploring, the agent chooses one of the possible actions randomly for a rate of ϵ.

It should be noted that the actions to select from during exploration includes the currently best known action as well. By selecting the best action for a rate of 1-ϵ and a random action for a rate of ϵ, ϵ denotes what the rate of (possible) exploration is. This means that the ϵ value regulates the balance between exploitation and exploration.

2.1.3 Dyna-Q

The Dyna-Q algorithm consists of Q-learning with the addition of creating a model of the world. Model-based RL makes a model of the world so that the RL agent can learn offline (learn without interacting with the environment). The model estimates the transition function $T(s'|s,a)$ and reward function $r(s,a,s')$. The model of the world is used to update the Q-table during the planning steps for multiple state-action pairs without interacting with the environment. This offline Q-table improvement can speed up the convergence to the optimal policy, increasing sample efficiency with respect to Q-learning. However, this only happens when the estimated model of the world is accurate enough. Otherwise, the agent updates the Q-table with wrong information. This could result in values in the Q-table that are further away from the correct values for those states. The correct values are the values of the Q-table when it has converged to the optimal policy.

The model of the world has the following notation:

$$\hat{p}(s', r_{avg}|s,a),$$

this notation denotes the probability that the agent ends up at the next state (s') when doing an action (a) on the current state (s). When the agent is at the next state, it has an average reward (r_{avg}). This notation is divided into four equations, namely:

$$n(s,a,s') \leftarrow n(s,a,s') + 1$$

$$R_{sum}(s,a,s') \leftarrow R_{sum}(s,a,s') + r$$

$$\hat{p}(s'|s,a) = \frac{n(s,a,s')}{\sum_{s'} n(s,a,s')}$$

$$\hat{r}(s,a,s') = \frac{R_{sum}(s,a,s')}{n(s,a,s')},$$

where $n(s,a,s')$ denotes the amount of times an agent arrived at state s' after performing action a at state s; $R_{sum}(s,a,s')$ denotes the total amount of rewards the agent got at state s' after performing action a at state s, $\hat{p}(s'|s,a)$ denotes the chance that the agent has of ending up at state s' after performing action a at state s, $\hat{r}(s,a,s')$ denotes the average reward the agent receives when ending up at state s' after performing action a at state s. $\hat{p}(s'|s,a)$ and $\hat{r}(s,a,s')$ are the estimations of $T(s'|s,a)$ and $r(s,a,s')$ respectively.

For every planning step, the model of the world is used to update the Q-table. This is done by randomly selecting a state action combination where the agent has sampled the environment with. Afterwards, the next state s' and reward r using $\hat{p}(s'|s,a)$ and $\hat{r}(s,a,s')$ are simulated respectively. Finally, the s,a,s' and r are used to update the Q-table using: $Q(s,a) \leftarrow Q(s,a) + \alpha * [r + \gamma * \max_{a'} Q(s',a') - Q(s,a)]$.

On a side note, the performance of another Model-based RL algorithm: Prioritised Sweeping [MA93] is not measured in this thesis. The Prioritised Sweeping algorithm is briefly discussed as it appears that Dyna-LLM behaves similar to Prioritised Sweeping as is shown in the results (section 4). The Prioritised Sweeping agent chooses the state-action pairs to be simulated, based on priority using the following formula:

$$\mathbf{P} \leftarrow |r + \gamma * \max_{a'} \hat{Q}(s', a') - \hat{Q}(s', a')|,$$

where $\mathbf{P}$ is the priority. In the planning updates, the state-action pair with the highest priority is used to update the Q-table. If the difference is high then the priority is high as it could prove to be useful to update that state-action pair, since it would change the Q-table more than when the difference was low. The Prioritised Sweeping algorithm aims to more efficiently update the Q-table, since it chooses the most important or promising state-action pairs to update in the planning updates rather than choosing random state-action pairs, as is done in Dyna-Q.

2.1.4 LLM

Large language models (LLMs) can generate complex text about a wide range of topics and answer questions [ABB+22]. The ability to answer questions appropriately given a prompt emerges during unsupervised pre-training [BMR+20]. The LLM can recognise the task in the prompt by doing in-context learning. In-context learning gives a description of the task or examples of possible answers, aiming to make the LLM answer appropriately by predicting what word comes next.

Transformer-based LLM architectures [VSP+23], such as GPT-3 [BMR+20] use multi-head self-attention, allowing them to make internal contextual representations. The representations are used to determine which word fits best as the next word given the previous words (the query) [MCH+20]. The transformer based LLMs have demonstrated the ability to reason.

Using prompting strategies such as Chain of thought (CoT) prompting [WWS+23], LLMs can generate reasoning steps that improve accuracy on tasks that can be divided into smaller reasoning steps. The idea of CoT is to let the LLM think step by step, splitting the bigger (more difficult) task into smaller (easier) tasks. When using few-shot prompting (using multiple examples in the prompt) the prompt consists of examples on how to perform the task with intermediate steps and the reasoning behind them.

Because of the reasoning capabilities of LLMs, LLMs can be used to make decisions or suggestions, which can be useful in RL.

2.1.5 Tiny-Llama

Tiny-Llama[3] is used in the main experiment of this thesis.

Llama2 is an open-source LLM that allows researchers and others to experiment with LLMs.

[3] https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0

Tiny-Llama has the same architecture and tokenizer as Llama2[4] and has less parameters then Llama2. Tiny-Llama has 1.1 billion parameters and Llama2 has two versions with 13 billion or 70 billion parameters respectively. Tiny-Llama is made for the same reason as Llama2 with the added benefit of less computational requirements and less memory needed to use the model.

Tiny-Llama is trained on the UltraChat[5] dataset. The UltraChat dataset contains ChatGPT generated conversations. Afterwards, Tiny-Llama is finetuned on the UltraFeedback[6] dataset using DPOTrainer[7]. UltraFeedback [CYD+24] is a GPT-4 generated dataset consisting of 250 thousand user-assistant conversations.

2.2 LLM in RL for reduction of sample complexity

Previous research has been done to reduce the amount of environment sampling needed to converge to the optimal RL solution. Some methods are:

- using an LLM in order to shape rewards
- using an LLM as action suggester
- using an LLM as transition function

2.2.1 Shaping rewards

Bhambri et al. [BBK+24] aimed to improve the sample efficiency of Q-learning by introducing subgoals in order to shape rewards. Their method uses an LLM to propose actions for given states, generating state-action pairs as subgoals as part of a partial plan and a complete plan (to reach the goal). A partial plan does not reach the goal, while a complete plan does. These subgoals are integrated into the training method by assigning modified rewards via a custom reward shaping function. The shaped rewards are used during the training process using Potential-Based Reward Shaping (PBRS) [NHR99], a machine learning technique that allows to converge to the optimal policy despite changing the reward function. This approach increased sample efficiency without compromising optimal policy convergence. They showed the most improvement of the sample efficiency when using a complete plan. Additionally, using a partial plan still improved the sample efficiency. Showing that LLMs can be used to improve the sample efficiency of the tabular RL method: Q-learning.

2.2.2 Suggesting actions

Karimpanal et al. [KSR+23] aimed to improve the sample efficiency of RL by utilising an LLM to suggest actions, preferably only when needed. These suggested actions were then explored first. Additionally they used a second RL agent. The second RL agent is trained to make the decision of prompting the LLM or not. This decision is based on the current state of the first RL-agent. Since prompting an LLM can be expensive and time consuming, this secondary RL-agent improves

[4] https://www.llama.com/llama2/
[5] https://huggingface.co/datasets/stingning/ultrachat
[6] https://huggingface.co/datasets/openbmb/UltraFeedback
[7] https://huggingface.co/docs/trl/dpo_trainer

the average usefulness of the LLM output. To determine if the LLM output was helpful, they compared the LLM predicted state and actual next state in combination with an evaluation score on how desirable the LLM output was. Their results show that using LLMs while selecting the right moments to do so, can improve sample efficiency.

Ahn et al. [ABB+22] also used the reasoning capabilities of an LLM to suggest actions in order to guide exploration. These suggestions are used to reduce the action space while maintaining the ability to converge to the optimal solution. This increases the exploration efficiency and thus the sample efficiency while possibly decreasing the accuracy. These suggested actions are generated at the beginning of the task. Rather than exploring the suggested actions first, they are assigned a usefulness score. The usefulness score reflects the LLM's confidence in the usefulness of each action. This score is then combined with a skill affordance value score, which represents the agent's own learned knowledge about which actions are useful in a given context. The combined score is used to determine which action to explore, combining policy knowledge of the RL-agent and reasoning of the LLM to likely improve exploration efficiency. Thus, while the new algorithm is not compared to the classical RL algorithm, the likely improvement of exploration efficiency will likely improve sample efficiency.

Duong et al. [DYZ25] uses an LLM to suggest actions that an on policy RL-agent would normally use often, reducing the action space. The suggestions are based on what action the LLM thinks the RL-agent should on a certain state, which is represented by a location. The actions are then followed by an online RL-agent and a model of the world using that data is estimated, the RL-agent is not learning at this stage. Afterwards, they pre-train the RL-agent on the constructed model of the world (offline learning). This results in the pre-trained policy. With knowledge of the RL-agent, the Q-table (or equivalent) is meant. Using the pre-trained policy as starting knowledge for the RL-agent, the model free RL-agent explores further and learns as usual to learn the optimal policy. This results in an increase of sample efficiency by utilizing the LLM suggestions to explore the environment at the start of the learning process.

2.2.3 LLM as Transition function

Liu et al. [LHZ+24] designed a framework called: RAFA and stands for: reason for future, act for now. This framework helps the LLM give appropriate outputs and help during the reasoning process of the LLM. Their RL algorithm uses an LLM for planning and for giving information about state-action pairs, replacing the Q-table of Q-learning. During the process of finding the optimal solution using RL, the LLM learner-planner (LLM-LR-PL) constructs a plan (as a) search tree. It uses one LLM instance to suggest an action, another one to generate the next state and another one to estimate the value function. It appears that the separation of the functions over different LLM instances reduces hallucinations as the LLM validates its other instances output.

The LLM that estimates the value function generates the expected cumulative reward, which is used to determine which plan (the path in the search tree) is the best plan. The starting action of the best plan is chosen because this action is (following the reasoning of the LLM) the best long term action to take at the current state. After sampling the environment (after doing an action) with the RL-agent, a new plan search tree is constructed again by the LLM learner-planner,

which returns the supposedly best action. The RL-agent only uses the suggested actions of the LLM learner-planner. Therefore, the RL-agent has no need to have a Q-table to keep track of the usefulnesses of the state action combination. Instead, the *state, action, next state and reward* are saved in the memory, which is used in the prompt of the LLM to give feedback. After the chosen action gives an unexpected result, the feedback is given to the LLM which generates a feedback summery. This summary is added to the prompt to alter the planning of the LLM which alters the policy. This algorithm, using the LLM gpt-4 and gpt-3.5, has improved sample efficiency by mitigating hallucinations and by following the reasoning of the LLM.

Thus, this approach uses an LLM as a transition function and reward function. The LLM generates the expected next state and reward. RAFA does not use this generated data to alter its policy. It only alters the suggested action that may alter the policy after sampling the environment. This makes RAFA an algorithm that uses an LLM for primarily action suggesting, by using the LLM as a transition function.

2.3 Differences between small LLM and LLM

While it would improve explainability of the results to discuss the work of others using a small LLM instead of a (big) LLM in a RL context, those papers were not found. Discussed related work have used LLMs having more than 7 billion parameters. A small LLM such as Tiny-Llama has 1.1 billion parameters. To give background information on **RQ2**: Are large LLMs better than small LLMs in model-based reinforcement learning?, the differences between a small LLM and (big) LLM will be discussed instead.

Mahapatra and Garain [MG24] compare LLMs of sizes between 0.1 bilion and 13 bilion parameters. The work of Mahapatra and Garain show that increasing the parameter count of an LLM increases the overall performance of the LLMs. Specifically the readability, informativeness, but not faithfulness of the LLM output improved when increasing the parameter count. The readability tells how grammatically correct and easy to read the LLM output is. The informativeness tells how similar the LLM output and source text are. The faithfulness tells how factual the LLM output is compared to the source data.

While the general performance of the LLM output is better for models with more parameters [MG24], the increased number of parameters increases the inference times and computational costs [JSL+24].

2.4 Conclusion

Discussed related work uses the LLM in the LLM assisted reinforcement learning as action suggester. Additionally, the discussed related work never uses the LLMs to directly alter the knowledge (the Q-table) of the RL-agent. They verify the result of the move on a certain state using the environment. By nudging the RL-agent in the right direction using the LLM and by verifying the proposed policy using the environment, they improve sample efficiency while maintaining the property to always be able to have optimal policy convergence. Unlike the related work, Dyna-LLM updates the Q-table directly using the LLM output as can be seen in the methodology (section 3.4).

The related work approach that comes the closest to directly altering the knowledge of the RL-agent without validating the suggestion of the LLM using the environment is during the reward shaping of Bhambri et al. [BBK+24]. Since only the Q-table is used by the policy to choose an action, making the Q-table the knowledge of the RL-agent. In this related work approach the LLM proposes actions on certain states. These state action pairs are then used by a custom function to shape rewards while maintaining optimal policy convergence. This shows that using the LLM to directly alter the knowledge of the RL-agent might result in the RL-agent not finding the optimal policy. For example, in the work of Bhambri et al. [BBK+24], the rewards are shaped, while they could have chosen to shape the Q-table itself. However, by shaping the rewards instead, they have Q-table that is influenced by the LLM but only indirectly, not altering the Q-table as much to loose the ability to converge to the optimal policy.

While the general performance of the LLM output is better for models with more parameters [MG24], the increased number of parameters increases the inference times and computational costs [JSL+24].

The purpose of this thesis is to add scientific knowledge to the related work by exploring the use of a small LLM. Using the LLM to directly alter the Q-table, using the LLM as transition function as small LLMs are not used in the found related work. Not finding related work that fails to reduce the sample efficiency could be explained by the fact that those papers are less popular as they are possibly seen as less useful. It could also be that those papers are not published because they are not exciting. For example, the results are not significant since the attempt to achieve something new failed (publication bias).

3 Method

In order to answer the problem statement (**PS**) and research question 1 (**RQ1**) a comparison is made. The comparison between Dyna-LLM, Q-learning and Dyna-Q is used to answer **RQ1**, answering the **PS**.

RL is used to solve a task, the RL-task. The RL-task is finding the best (often times the shortest) path in a grid world. The best path, or most valuable path, is the path where the expected cumulative result would be the highest given perfect information. The agent that follows the best path generally follows the optimal policy.

The new algorithm that uses the LLM as transition and reward function is called Dyna-LLM. The other algorithms that are compared are the Q-learning and Dyna-Q algorithms. Parts of the used experiment code were already made by me and a fellow student Kenji Opdam as part of assignments from professor Thomas Moerland. The code parts were used since the the Q-learning and Dyna-Q algorithms were already implemented in those assignments.

3.1 Environment

During the experiment two environments are used. The two environments are exactly the same, except that one of the two has wind blowing downwards (pushing the agent one state down around 50% of the time). This windy environment is called: Windy-shortcut environment. The other environment without wind is called: Shortcut environment.

The grid world environment as is shown in the fig 1 the environment is a five by five grid world that consists of 25 states (counting from left to right and from top to bottom) where every state is one of the following: a tile, a boulder, a cliff or a goal. A tile state is a place where the agent can stand. A boulder state is impossible to end up on, as it is supposed to acts like a wall in a maze. More precisely, if the agent would have ended up at the boulder state, the next state for the agent is the state before doing the move. If the agent ends up on a cliff state (by walking of a cliff), the next state for the agent is the begin state.

It should be noted that checking of the agent is on a boulder state or cliff state is not done before blowing the wind. This makes the agent able to use the wind to skip over one boulder state or one cliff state, which was unintended. However, given the environment layout and because the wind possibly only blows downwards, the ability to skip those states would never be part of the optimal solution of the task. This is because the state-action combinations where skipping a boulder or cliff would be possible, the resulting state would always result in following a path that would be longer than when the agent would not choose said action on those states. If the environment layout would be different then it could pose problems.

The goal state is the state where the agent has to find the most valuable path to. Additionally, the goal state is an absorbing state, making the episode end. An episode consists of all the timesteps, measuring from the first move (when the agent is at the start state), until the agent ends up at the goal state. The transition function $T(s'|s, a)$ of the environment links these states to

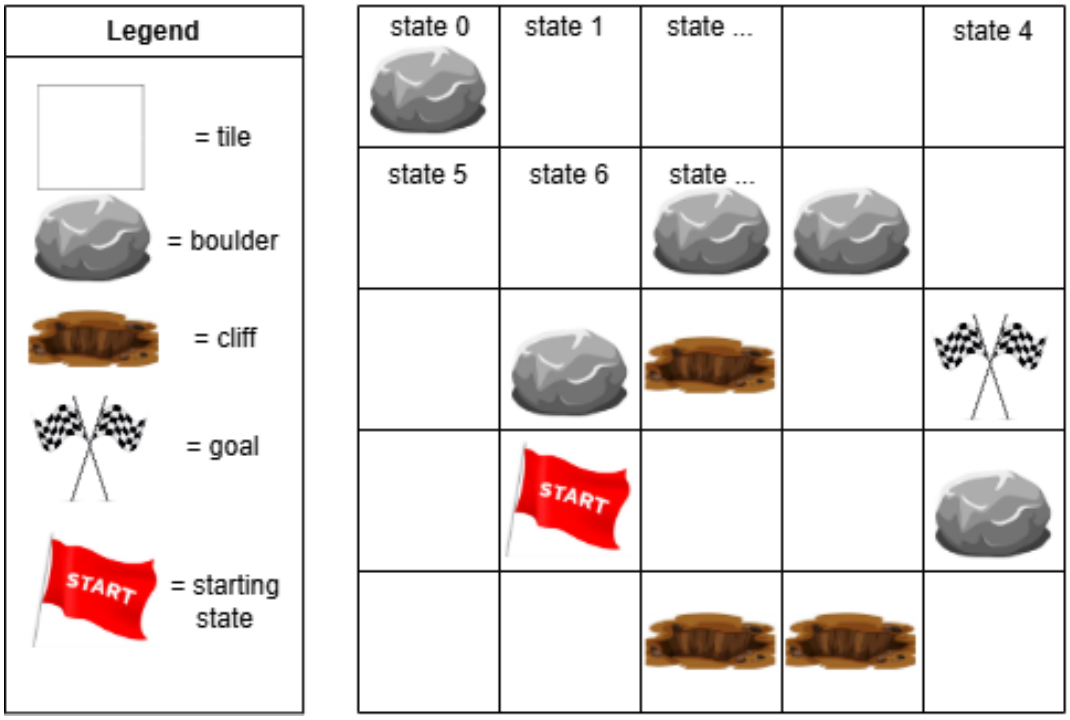

Figure 1: The layout of the environment with and without wind. The states are counted from left to right, from top to bottom. For example, the top left state is called: state 0, and the bottom right state is called: state 24.

each other in the arrangement that can be seen in Figure 1. For example, when the agent is on state 5, when moving down without the wind blowing, function T gives as next state: state 10 and with the wind blowing: state 15.

Every (state, action, next state) tuple is coupled to a reward using the reward function $r(s, a, s')$. The reward function returns -1 for all tile states as next states and the goal state. When the agent tries to go to a boulder state, it ends up at a tile state, which results in a reward of -1. Furthermore, the reward function returns -100 when the agent falls in a cliff. In this case, using the MDP the (state, action, next state) tuple is needed to determine if the agent fell in the cliff, since the next state of the agent would be the start state. Normally the reward for the goal state is set to be some positive number instead of -1. However, because the goal state is an absorbent state, the cumulative reward of the path that involves the goal state sooner than later is less negative. Such a path contains less actions or moves. Therefore making the path have a less negative cumulative reward, which makes the goal state valuable for the RL-agents.

3.2 Q-learning

Q-learning is used in the main experiment. The following Algorithm 1 shows the implementation of the Q-learning algorithm.
In line 11 of Algorithm 1, the `select_action`() function of class instance π is used to output action a given state s. This function uses `np.random.choice` for both the exploration action selection and

Algorithm 1 Q-learning algorithm

Initialize `n_timesteps`
$\alpha \leftarrow 0.1$
$\epsilon \leftarrow 0.1$
$\gamma \leftarrow 1$
Initialize environment `env`
$s \leftarrow$ `env.state()` ▷ get starting state
`n_actions` $\leftarrow$ `env.action_size()` ▷ get amount of actions
`n_states` $\leftarrow$ `env.state_size()` ▷ get amount of states
$\pi \leftarrow$ `QLearningAgent`$(n\_states, n\_actions, \alpha, \epsilon, \gamma)$ ▷ initialize Q-learning agent, init Q-table, α,ϵ,γ
for $t = 0$ to `n_timesteps` **do**
    $a \leftarrow \pi$.`select_action`(s) ▷ select action using $\pi_{\epsilon-greedy}(s, a)$
    $r \leftarrow$ `env.step`(a) ▷ perform the action, get the reward and update the state of the agent
    $s' \leftarrow$ `env.state()` ▷ get the updated state, which is the next state
    π.`update`(s, a, r, s') ▷ update the Q-table using: $Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$
    if `env.done()` **then**
        $s \leftarrow$ `env.reset()` ▷ if the goal has been reached, place agent at the starting state of the grid world
    else
        $s \leftarrow s'$ ▷ else: update the state of the agent
    end if
end for

the exploitation action selection, instead of using `np.argmax` for the exploitation action selection. If more than one action a at state s has the same value, `np.argmax` gives the lowest corresponding index. This creates a bias towards choosing the actions with lower indexes. By randomly choosing between the best actions in the `select_action`() function, the action selection bias is eliminated.

3.3 Dyna-Q

During the `update()` function on line 14 of Algorithm 1, the Dyna-Q algorithm updates the model of the world, updates the Q-table the same way as Q-learning and performs planning steps.
For the Dyna-Q algorithm, the variables to estimate the world: $n(s, a, s')$, $R_{sum}(s, a, s')$, $\hat{p}(s'|s, a)$ and $\hat{r}(s, a, s')$, are initialised at line 9 of Algorithm 1 in the Dyna-Q class instead of the Q-learning class. For every planning step the model of the world is used to update the Q-table. This is done by randomly selecting a state-action pair that the agent has sampled the environment with. Afterwards, the next state s' and reward r using $\hat{p}(s'|s, a)$ and $\hat{r}(s, a, s')$ are simulated respectively. Finally, the s,a,s' and r are used to update the Q-table using: $Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$.

3.4 Dyna-LLM

The Dyna-LLM algorithm is almost the same as the Dyna-Q algorithm. The only difference is that the Dyna-LLM algorithm uses an LLM to predict the next state s' and reward r instead of using the estimated model of the world of Dyna-Q. The aim of using an LLM to predict the consequences of an action on a state is that the LLM might learn the mechanics of the environment faster by reasoning, instead of only relying on sampling the environment to estimate the probability of the consequences as is done in the estimated model of the world in Dyna-Q. For example, when a certain state action gets sampled, the Dyna-Q algorithm would only know how that specific state action combination behaves while not being able to predict a state action that it has not sampled before. The LLM might be able to predict the consequences of the unsampled state action combination based on previously sampled other states. Or the LLM might be faster in estimating the probability of the consequences using reasoning.

The Dyna-LLM Q-table update algorithm with planning steps is shown in Algorithm 2. Where the LLM updates the history prompt with past interactions in order to generate next state s' and reward r during the planning updates. The Dyna-Q and Dyna-LLM algorithms update the equivalent parts of the algorithm at the same time. A while loop is used for the planning updates so that the Dyna-LLM agent can generate the s' and r again if the generated output is found to be invalid. This while loop ensures that both algorithms always have the same amount of Q-table updates during the planning phase of the Q-table update function.

The LLM output is validated to be of the right syntax by the `check_output_of_LLM` function. This function does the following:

- Checks if the input prompt without the last character is equal to the beginning of the LLM output.

  - The last character of the input is not checked during this step, because the LLM changes that character sometimes. For example, the LLM does not place a dot at the end of that sentence in the output while it is present in the prompt (the LLM input). The change of the last input character does not give room for errors. If the beginnings (the input prompt and the beginning of the LLM output) are not equal, it does not mean that the LLM output is invalid. So the LLM output is saved in the LLM_output_validation_errors.txt file and after the experiment checked for this error. The error is flagged with the string: "ERROR:Beginnings are not equal", which is not found after the main experiment in the text file.

- Matches the regular expression (RE) in the LLM output after the input prompt (the answer of the LLM).
  - The regular expression (RE) that is used for the validation is (all the content of the following colour box):

    Regular expression 1. Validation expression

    Response: "State: \d+, Action: Agent moves (up|down|left|right), Reward: -?\d+, Next state: \d+\."

    The RE matches the exact string in the answer, except for the placeholders: {"\d+", "(up|down|left|right)", "-?\d+", "\d+\."}. Using: \d+, makes the RE match for one or multiple numbers at that location in the string. Using: (up|down|left|right), makes it match for one of the the following strings: "up", "down", "left", "right". Using: -?\d+, makes it match for zero or one: "-", followed by one or multiple numbers. Using: \d+\., makes it match for one or multiple numbers, followed by a dot.

    After a match of the RE is found in the answer, the reward and next-state values can be found using indexes and are returned to the update function. If there is no precise match for the RE in the LLM output (after the input prompt), the answer is of invalid syntax and a flag for an invalid answer is returned to the update function.

- In the case of a beginnings not equal error (input prompt is not equal to the start of the LLM output), the returned reward and next-state are saved in the LLM_output_validation_errors.txt file.

If the beginnings are equal and because the `check_output_of_LLM` function uses the first RE match as the location in the string to get the reward and next-state from, the returned reward and next-state must be of the right syntax. However, the `check_output_of_LLM` function does not check if the values of the next-state are between: [0,24]. Therefore, there is a try and except statement for the Q-table update that gives an error when a wrong next-state value was returned by the `check_output_of_LLM` function. Thus, the `check_output_of_LLM` function (after checking the beginnings errors and the corresponding rewards and next-states), together with the try and except statement for the Q-table ensure valid rewards and next-states.

Algorithm 2 update function of Dyna-LLM

s, a, r, s', `done` and `n_planning_updates` received as parameters
`action` $\leftarrow$ `action_to_string`(a) ▷ get action as a word instead of as integer
`history_prompt` $\leftarrow$ `history_prompt` + `history_addition`
if `history length > 37 samples` **then**
    `history_prompt` $\leftarrow$ `the 37 latest samples in history` ▷ make the history have at most 37 samples to stay under the maximum amount of input tokens (2048 tokens).
end if
`history_copy` $\leftarrow$ `history_prompt` ▷ making a copy is not necessary however, this variable was used during the experiment
$n(s, a, s') \leftarrow n(s, a, s') + 1$ ▷ keep track of what the agent has sampled
$Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$ ▷ update Q-table
if `done` **then**
    return ▷ if the goal has been reached, stop the update function
end if
`i` $\leftarrow 0$ ▷ the planning loop counter
while `i` $<$ `n_planning_updates` **do**
    $s \leftarrow$ `np.random.choice(np.nonzero(`$n()$`)[0])` ▷ get a previously visited state
    $a \leftarrow$ `np.random.choice(np.nonzero(`$n(s)$`)[0])` ▷ get a previously used action at state s
    `action` $\leftarrow$ `action_to_string`(a) ▷ get action as a word instead of as integer
    Initialize `LLM_input_prompt` using `history_copy`, s and `action`
    tokenize input
    `generated_text` $\leftarrow$ generated text ▷ use the LLM to generate the next state s' and reward r
    r, s', `invalid_answer` $\leftarrow$ `check_output_of_LLM(LLM_input_prompt, generated_text)` ▷ check the validity of the output of the LLM and return the r, s' and if the answer was valid with `invalid_answer`
    if `invalid_answer` **then**
        continue ▷ if the answer is invalid, try the loop again
    end if
    Try
        $Q(s, a) \leftarrow Q(s, a) + \alpha * [r + \gamma * \max_{a'} Q(s', a') - Q(s, a)]$ ▷ update Q-table, which could fail as the validity does not check for valid values of r and s'
    Except
        continue
    `i` $\leftarrow$ `i`$+1$ ▷ increase the planning loop counter to continue to the next loop if all went well
end while

The Dyna-LLM algorithm uses an LLM instead of $\hat{p}(s'|s, a)$ and $\hat{r}(s, a, s')$ to predict s' and r by generating text given a prompt. While the complete prompt can be seen in the appendix, an interpretation of the prompt can be seen in the coloured box and is constructed in the following matter:

Prompt 1. LLM input prompt

1. Role:
2. You are an AI that simulates an environment for model-based reinforcement learning.
3.
4. Environment Rules:
5. The agent is in a 5×5 grid maze with states numbered from 1 to 25, structured as follows:
6. Top-left corner: State 1.
7. etc...
14. If the action is new, apply logical reasoning based on the grid structure:
15. Moving left decreases the state number by 1, unless at the left boundary.
16. Moving right increases the state number by 1, unless at the right boundary.
17. ...
21. You must return the following after an action:
22. state, location, action, reward, next_state, next_location.
23.
24. Past Interactions:
25. Context: The action from a certain state might result in a different next_state than the following action because of environment circumstanses.
26. In case this happens, write: "Circumstance" in the output.
27. """+`history_copy`+"""
28. Now Continue:
29. Follow past interaction patterns if available. If the interaction is new, use the movement rules to determine the correct outcome.
30.
31. State: """+str(s)+""", Action: Agent moves """+`action`+""".

where the history part of the prompt (`history_copy`) is maximally the last 37 sampled interactions of the agent with the environment. This limitation is because the LLM can have a certain prompt

length. While this limitation is a disadvantage given that Dyna-Q can save all the interactions, the LLM hopefully gives correct answers using the history it does have available. The rest of the prompt consists of the context for the LLM (lines 1-23) and the start of the answer to make the LLM generate the answer in the right syntax (lines 28-31).

The `history_copy` prompt is the concatenation of the `history_addition` prompts which can be seen in the following coloured box:

Prompt 2. LLM history addition prompt

1. State: """+str(s)+""", Action: Agent moves """+`action`+""" →
2. Response: "State: """+str(s)+""", Action: Agent moves """+`action`+""", Reward: """+str(r)+""", Next state: """+str(s')+"""."

The `action` is the action represented as a word: {"up","down","left","right"} instead of as a number: {0,1,2,3} respectively.

Furthermore, the start of the answer (LLM input prompt,lines 28-31) in combination with the format of the LLM history prompt make the LLM often give the output of the same format as line 2 in the LLM history prompt. This start of the answer will be referred to as the question that is asked to the LLM.

The prompt consists of context in lines 1-26, with the aim to let the LLM understand the task, using in-context learning [BMR+20]. The role and environment rules section are aimed to give the structure of the default answer. While the classical RL-agents (Q-learning and Dyna-Q agents) can not reason with the wind, the Dyna-LLM agent hopefully can. To keep the comparison between the classical RL-agents and the Dyna-LLM agent fair, the environment rules consist mostly of the rules that are true in every five by five grid world. Additionally, in the environment rules, it is mentioned that the wind could have an effect on the outcome. While this extra information could be seen as unfair compared to the classical-RL, I wanted to make it easier for the LLM to start reasoning with the effect of the wind. Few-shot prompting [BMR+20] is used in the history. Few-shot prompting means having more than one example question and answer in the prompt. Line 27 consists of the history. The history consists of environment interaction samples (the examples). Chain of thought (CoT) [WWS+23] could be used to potentially improve the reasoning of the LLM. CoT, together with few-shot prompting gives examples with reasoning steps. Placing the reasoning steps in the prompt (without the use of an LLM) could be done by using the environment as feedback. For example, the environment could return that there was a boulder at a certain location. However, this would give the Dyna-LLM agent increased perception compared to the the Q-learning and Dyna-Q agents, because the classical RL-agents do not have that information available. Because I want it to be a fair comparison between the Q-learning, Dyna-Q and Dyna-LLM agents, CoT was not used.

3.5 Gathering and plotting of data

To answer **RQ2** and **RQ3**, an additional experiment is performed. The experiment to answer the problem statement (**PS**) and research question 1 (**RQ1**) is called the main experiment. The

experiment to answer **RQ2** and **RQ3** is called the mini experiment. The motivation, methodology and results of the mini experiment is discussed in the LLM output comparison section (section 4.5).

As was previously mentioned, the problem statement is: What is the effect on the environment sampling efficiency of using Tiny-Llama as the transition and reward function during the planning updates of Model-based RL? To answer **RQ1** the evaluation rewards of the following algorithms are compared:

- a non model based RL algorithm: Q-learning
- a model based RL algorithm: Dyna-Q
- a model based RL algorithm with the LLM as model: which is called Dyna-LLM

The evaluation rewards are provided by the evaluation function. The evaluation function gives the arithmetic mean reward of (`n_eval_episodes` which is) 30 episodes. One episode in this case is the cumulative reward from timestep 0 until 100 or until the goal has been reached. Thus, the evaluation reward is the mean amount of steps the agent takes when using a greedy policy on the timestep of the evaluation. For example, an evaluation reward of -4 means that 4 moves were done on average per episode. In this thesis, only the arithmetic mean is used, opposed to using other ways of calculating the mean. The comparison of the evaluation rewards of Q-learning, Dyna-Q and Dyna-LLM, together with the answers of **RQ2** and **RQ3** are used to measure and explain the sample efficiency of Dyna-LLM. Because Dyna-LLM is the most similar to Dyna-Q, primarily the comparison between Dyna-LLM and Dyna-Q is used to determine the effect on the sample efficiency. The Dyna-LLM and Dyna-Q comparison will show the difference in evaluation rewards using different models. Q-learning is compared with Dyna-LLM to see the difference between not using a model and using the Tiny-Llama as transition and reward function.

To clarify, the method consists of gathering data on grid world mazes: one without and one with wind. After gathering the results of Q-learning, Dyna-Q and Dyna-LLM, their performances are compared and explained in section 4. At the beginning of the main experiment, a seed is set. The main experiment is divided into sub-experiments. A sub-experiment is performed for every agent-environment combination and gathers all the results of (`n_repetitions` which is) 40 or (`llm_n_repetitions` which is) 10 repetitions.

The `n_repetitions` or `llm_n_repetitions` loop consists of Algorithm 1, or the respective algorithm for the Dyna-Q or Dyna-LLM agents. Additionally, the code to select the corresponding agents and to save the results is present in the sub-experiment loop as well. Such a sub-experiment consists of initialising one algorithm from the variable `agent_list` which contains: Q-learning, Dyna-Q and Dyna-LLM with one environment from the variable `env_list`. Afterwards, it performs (`n_timesteps` which is) 601 timesteps for (`n_repetitions` which is) 40 repetitions or (`llm_n_timesteps` which is) 301 timesteps for (`llm_n_repetitions` which is) 10 repetitions in case the Dyna-LLM agent is selected. At the beginning of every repetition the algorithm and the environment is reset. For every repetition, every (`eval_interval` which is) 15 timesteps starting at timestep 0, an evaluation is done. The evaluation rewards are saved in data-frames for every repetition. This is then saved in a .csv file so that the min, max, standard deviation and mean can be calculated.

Additionally the time duration of every timestep is measured and saved in a .csv file as well, using data-frames to eventually save it in the .csv file. The time duration is collected by getting the time right before line 11 and directly after line 19 in Algorithm 1. This excludes the evaluation time duration, because it is not part of the RL algorithm. The difference is then saved.

The min, max, standard deviation and mean are calculated for every agent environment combination for the rewards and time duration. Because the results of the data collection parts of the code are saved in .csv files, the graphs can be made without having to rerun the data collection part of the experiment. Individual graphs as well as combined individual graphs are made. The combined individual graphs are made in order to compare the performance of the algorithms regarding the environments. For the means in the graphs, a Savitzky-Golay filter is used to smoothen the plotted lines.

Since the windless and windy environments are used, calculating what the average evaluation reward for the optimal path for each environment is useful. By calculating the expected optimal paths, it can be determined if the agent has updated the Q-table in such a way that results in a greedy agent following the optimal path. A greedy agent is an agent that only chooses the best known action, opposed to the ϵ-greedy agents which do not always choose the best known action. The ϵ-greedy agents are used to update the Q-table during the main experiment. Since the environment without wind is deterministic, the optimal path has an average evaluation reward of -4 because the shortest path has a length of 4. The optimal path for a windy environment is approximately an evaluation reward of -17. The reward average evaluation reward of -17 denotes the length of the most valuable windy path, because the optimal path goes around the cliff instead of crossing the cliff.

The optimal path in the windy environment has approximately an evaluation reward of -17 because of the following logic. At the states where the wind causes the agent to end up at the current state (the agent essentially did not move because of the wind), then the agent has to do: $1 * 0.5 + 1 * (0.5)^2 + 1 * (0.5)^3 + 1 * (0.5)^4 + 1 * (0.5)^5 + 1 * (0.5)^6 + 1 * (0.5)^7 ... \approx 2$ moves to progress to the desired next state. The calculation is based on the probability that the agent has to do one move to continue to the desired next state, plus the probability that it has to do two moves, plus the probability that it has to do three moves continuing this infinitely. More intuitively, the agent is blown back once, every two timesteps because of the 50% chance of wind.

Given the transition function of the environment, ending up at the same state after doing the move also happens when the agent would have ended up at a boulder state (because of the wind) as next state, because the transition function places the agent at the last current location if the next state is invalid. In the most valuable or optimal path of the windy environment, there are two states where the wind helps the agent: state 3 and 4. The wind in only possibly halts the agent at all the other states of the optimal path. Because of state 3 and 4, going from state 3 to the goal state (state 19) uses on average 3 moves. This makes the average amount of moves: 17 (from the start state go to state 15, 2 moves total performed; state 10, 4 moves total performed; state 5, 6 moves total performed; state 6, 8 moves total performed; state 1, 10 moves total performed; state 2, 12 moves total performed; state 3, 14 moves total performed; state 19, 17 moves total performed).

3.6 Conclusion

In conclusion, the mean rewards, given by the evaluation function, and time duration for all environment-agent combinations are saved. The possible environments are a windy environment or the same environment without wind. The possible agents are the Q-learning, Dyna-Q and Dyna-LLM agents. The results from the main experiment are used to compare the performance of the environment-agent combinations, answering the problem statement.

4 Results

The parameters for the experiment were set to the values shown in Table 1.

Parameter	Value	Not used for
`n_repetitions`	40	Dyna-LLM
`llm_n_repetitions`	10	Q-learning and Dyna-Q
`n_timesteps`	601	Dyna-LLM
`llm_n_timesteps`	301	Q-learning and Dyna-Q
`eval_interval`	15	-
`n_eval_episodes`	30	-
`max_episode_length`	100	-
`smoothing_window`	5	-
ϵ	0.1	-
α	0.1	-
γ	1.0	-
`n_planning_updates`	10	Q-learning

Table 1: The parameters and values. The algorithm that does not use the parameter is noted in the most right column. A dash (-) is placed when all the algorithms use the variable with the corresponding value.

With the hyperparameters being shown in Table 2:

Hyperparameter	Value
`agent` in `agent_list`	(Dyna-LLM, Q-learning, Dyna-Q)
`env` in `env_list`	(Windless, Windy)

Table 2: The hyperparameters and the values. Note that the windy environment has a 50% chance of the wind blowing the agent one state downwards.

These parameters and hyper parameters were used to create the results and graphs in this experiment. Firstly, the performance of Q-learning will be discussed for both environments. Secondly, the performance of Dyna-Q will be discussed for both environments. Thirdly, the performance of Dyna-LLM will be discussed for both environments. Lastly, all the algorithms are compared, going over the most notable differences of the results.

4.1 Q-learning

In Figure 2, the average mean cumulative reward as well as the standard deviation can be seen for each evaluation. The line denotes the average value, and the area denotes the standard deviation. In Figure 3, the minimum and maximum evaluation reward from the 40 repetitions can be seen.

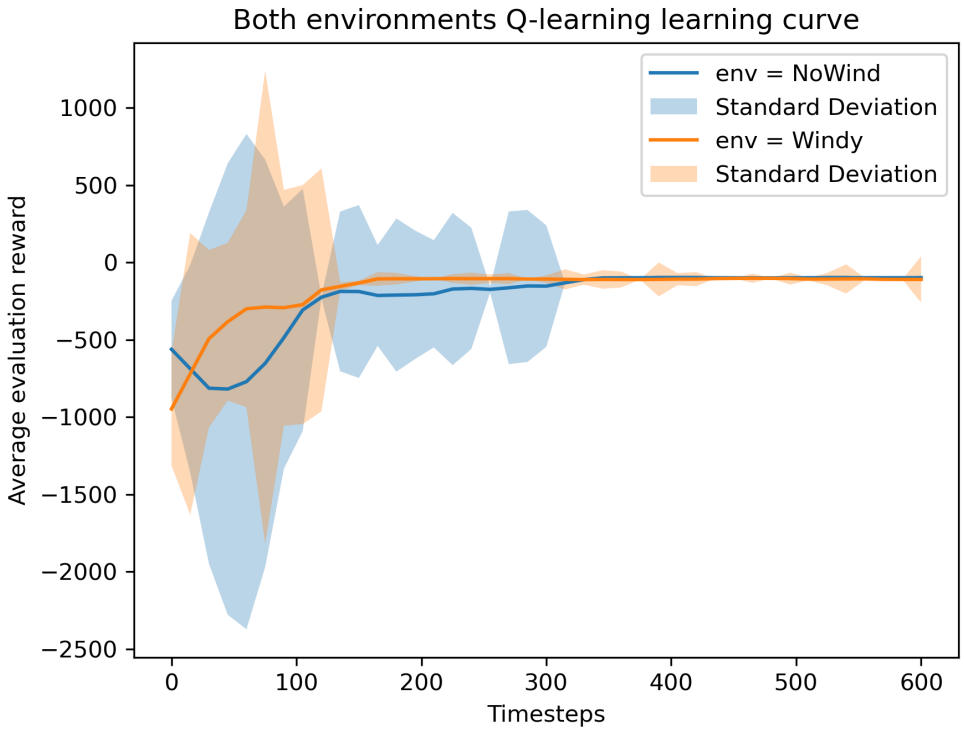

Figure 2: The average evaluation reward and standard deviation over timesteps for both environments. The average evaluation reward curves are calculated for 40 repetitions on 41 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. The evaluation is done every 15 timesteps. This figure shows that Q-learning mostly got higher average evaluations in a windy environment before timestep 330. Likely, because of the safer exploration in the windy environment after the initial cliff state exploration. After timestep 330, the average evaluation reward is mostly higher for the windless environment (denoted in the Figures as NoWind).

Figures 2 and 3 show that Q-learning mostly got higher average evaluations in a windy environment before timestep 330. After timestep 330, the average evaluation reward is mostly higher for the windless environment (denoted in the Figures as NoWind).

It could be the case that the windy environment has less dangerous states to explore, since going the short route often results in the agent being blown to a cliff-state. A state is dangerous if the agent can be blown into a cliff by the wind. Being blown into a cliff state when performing the correct action (going right) on the first dangerous tile state of the shortest path makes that tile state less valuable than the states of the path that goes around the cliff. It appears that the agent explores the states that cross the cliff less, resulting in less states that are also less dangerous to explore in total. Given that the path around the cliff has less cliff states, the agent is less often send back to the starting state, making the agent explore more safely in the windy environment after sampling the first cliff-crossing-state. Safer exploration after sampling the first cliff-crossing-tile-state likely leads to higher rewards on average in the timesteps before timestep 330.

With mapping the optimal path, I mean updating the Q-table in such a way that the optimal path is followed by greedy action selection. While the Q-learning algorithm did not regularly map the optimal path in both environments, Figure 3 shows that Q-learning mapped the optimal path for the environment without wind but not for the environment with wind, by getting maximum values of -4 but not a value close to -17 for the windless and windy environments respectively. An evaluation reward of -4 means that the path that the greedy agent took during the evaluation has an average length of 4, calculated from 30 episodes as was discussed in section 3.5. This is likely because the windy environment often resets the agent (the agent ends up at the same state after doing a move). This resetting makes exploration slower, likely resulting in not mapping the optimal path in 600 timesteps.

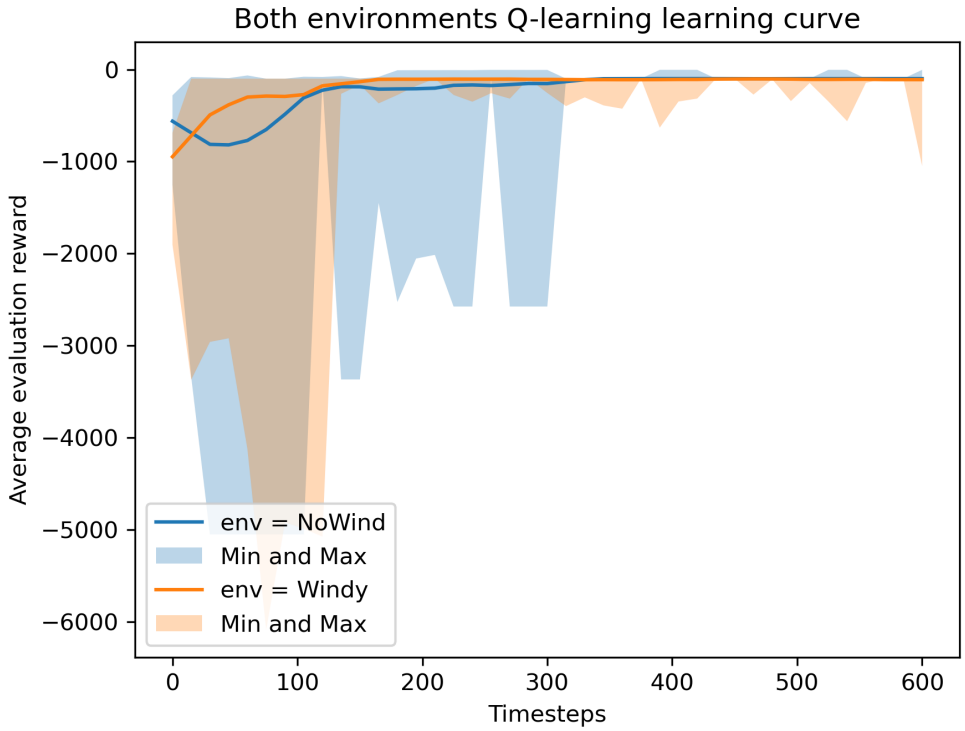

Figure 3: The average evaluation reward, minimum and maximum over timesteps for both environments. The average evaluation reward curves are calculated for 40 repetitions on 41 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. The evaluation is done every 15 timesteps. This figure shows that Q-learning mapped the optimal path for the environment without wind but not for the environment with wind, by getting maximum values of -4 but not a value close to -17 for the windless and windy environments respectively. The not mapping of the optimal path is likely caused by the slower exploration because of the wind.

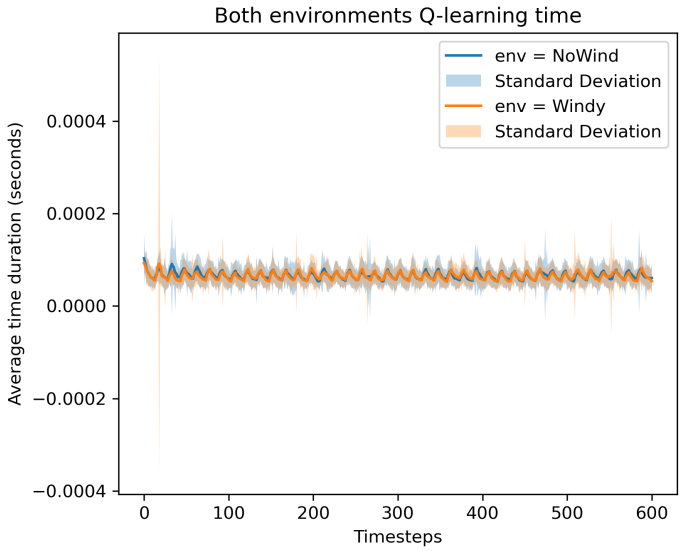

Figure 4: The average time duration and standard deviation over timesteps for both environments. The average time duration curves are calculated for 10 repetitions on 601 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. This figure shows that the Q-learning algorithm uses a time duration of approximately $6 * 10^{-5}$ seconds to perform the action and update the Q-table.

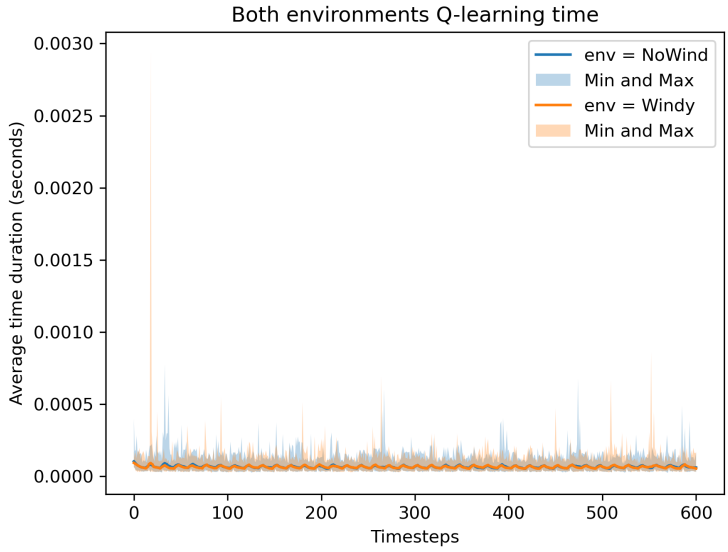

Figure 5: The average time duration, minimum and maximum over timesteps for both environments. The average time duration curves are calculated for 10 repetitions on 601 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. This figure shows that the Q-learning algorithm uses a time duration of approximately $6 * 10^{-5}$ seconds to perform the action and update the Q-table.

Figures 4 and 5 show that the Q-learning algorithm uses a time duration of approximately $6 * 10^{-5}$ seconds to perform the action and update the Q-table.

4.2 Dyna-Q

In Figure 6, the average mean cumulative reward as well as the standard deviation can be seen for each evaluation. The line denotes the average value, and the area denotes the standard deviation. In Figure 7, the minimum and maximum are plotted instead of the standard deviation.

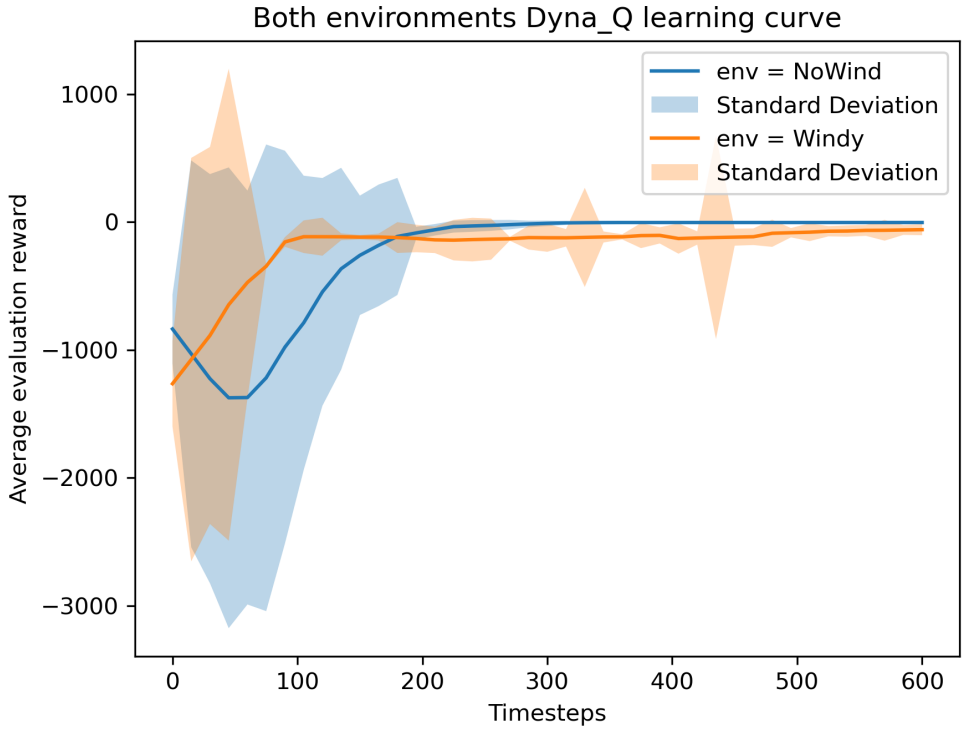

Figure 6: The average evaluation reward and standard deviation over timesteps for both environments. The average evaluation reward curves are calculated for 40 repetitions on 41 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. The evaluation is done every 15 timesteps. This figure shows that Dyna-Q mostly got higher average evaluations in a windy environment before timestep 160. After timestep 160, the average evaluation reward is mostly higher for the environment without wind. The safer exploration in the windy environment likely makes the average evaluation rewards higher in the beginning for the windy environment, compared to the environment without wind. At timestep 330 for the environment without wind, the algorithm has mapped the optimal path in the Q-table, since the average is -4 with a standard deviation of 0 for all timesteps higher than timestep 320. This is likely because the windy environment slows down exploration. The agent converges in the windless environment, but not in the windy environment.

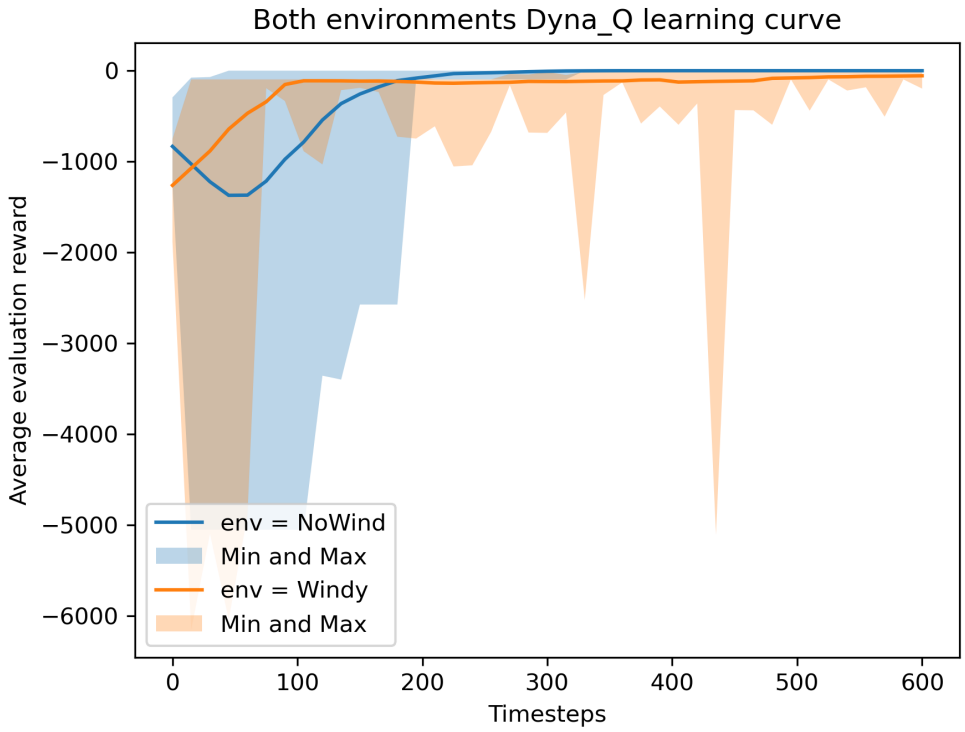

Figure 7: The average evaluation reward, minimum and maximum over timesteps for both environments. The average evaluation reward curves are calculated for 40 repetitions on 41 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. The evaluation is done every 15 timesteps. This figure shows that Dyna-Q mostly got higher average evaluations in a windy environment before timestep 160. After timestep 160, the average evaluation reward is mostly higher for the environment without wind. The safer exploration in the windy environment likely makes the average evaluation rewards higher in the beginning for the windy environment, compared to the environment without wind. At timestep 330 for the environment without wind, the algorithm has mapped the optimal path in the Q-table, since the average is -4 with a min and max of -4 for all timesteps higher than timestep 320. This is likely because the windy environment slows down exploration. The agent converges in the windless environment, but not in the windy environment.

Figures 6 and 7 show that Dyna-Q mostly got higher average evaluations in a windy environment before timestep 160. After timestep 160, the average evaluation reward is mostly higher for the environment without wind.

In Figures 6 and 7, like in Figures 2 and 3, the Dyna-Q algorithm gets higher average evaluation rewards in the earlier timesteps and thus performs better in the beginning in the windy environment, compared to the environment without wind. Having higher average evaluation rewards that are still lower than -100 likely means that the wind could steer the agent to explore the path around the cliff rather than the path crossing the cliff. The safer exploration likely makes the average evaluation rewards higher in the beginning for the windy environment, compared to the environment without wind.

In Figures 6 and 7, at timestep 330 for the environment without wind, the algorithm has mapped the optimal path in the Q-table, since the average is -4 with a standard deviation of 0 and the min and max being -4 for all timesteps higher than timestep 320. This means that the Q-table of Dyna-Q converged for all 40 repetitions or that the changes in the Q-table do not alter the result of the evaluation. In the windy environment, Dyna-Q does not converge for all repetitions but has at least one run that maps the optimal path in the Q-table at timestep 330 as well, which can be seen in Figure 7. Not converging is likely because of the slower exploration, since the agent gets reset by the wind (the agent ends up at the same state after doing the move because of the wind).

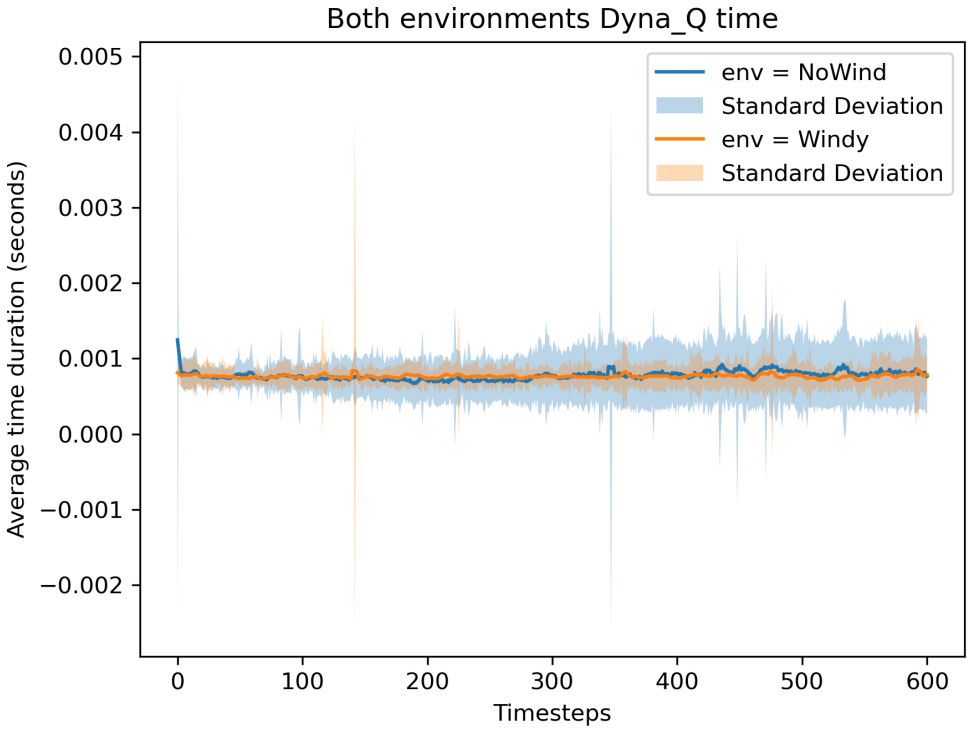

Figure 8: The average time duration and standard deviation over timesteps for both environments. The average time duration curves are calculated for 10 repetitions on 601 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. This figure shows that the Dyna-Q algorithm uses a time duration of approximately $8 * 10^{-4}$ seconds to perform the action and update the Q-table including the planning steps.

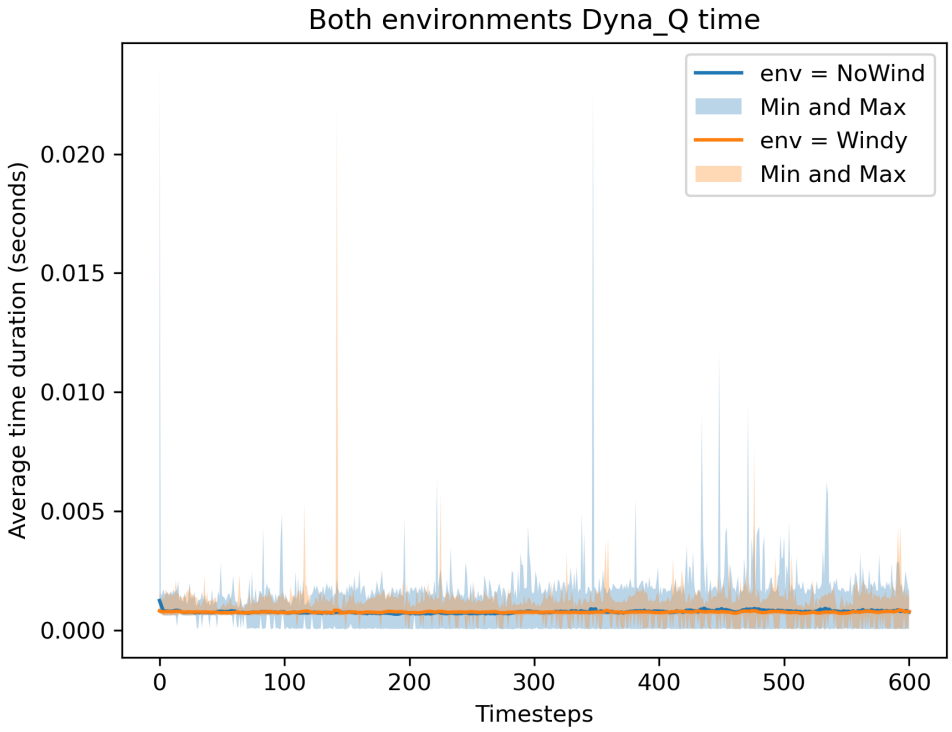

Figure 9: The average time duration, minimum and maximum over timesteps for both environments. The average time duration curves are calculated for 10 repetitions on 601 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. This figure shows that the Dyna-Q algorithm uses a time duration of approximately $8 * 10^{-4}$ seconds to perform the action and update the Q-table including the planning steps.

Figures 8 and 9 show that the Dyna-Q algorithm uses a time duration of approximately $8 * 10^{-4}$ seconds to perform the action and update the Q-table including the planning steps.

4.3 Dyna-LLM

In Figure 10, the average mean cumulative reward as well as the standard deviation can be seen for each evaluation. The line denotes the average value in both figures. In Figure 10, the area denotes the standard deviation. In Figure 11, the area denotes the minimum and maximum value for each of the 10 repetitions.

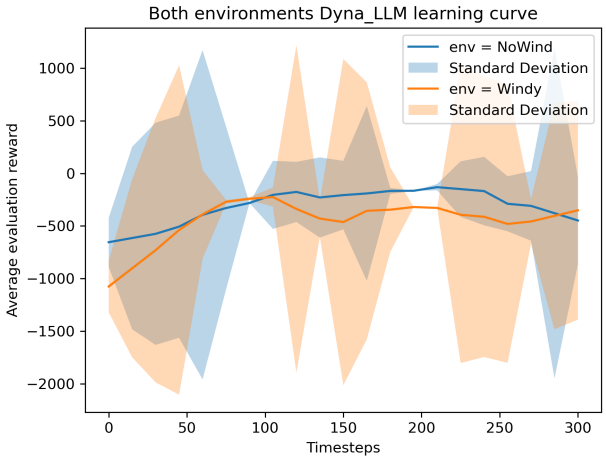

Figure 10: The average evaluation reward and standard deviation over timesteps for both environments. An evaluation is done every 15 timesteps. The average evaluation reward curves are calculated for 10 repetitions on 21 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. This figure shows that the Dyna-LLM algorithm performs better (gets higher average evaluation results) in the environment without wind for almost all timesteps except timesteps 75 until 100. The relatively large standard deviation in combination with the suboptimal average evaluation rewards could mean that the LLM outputs are incorrect.

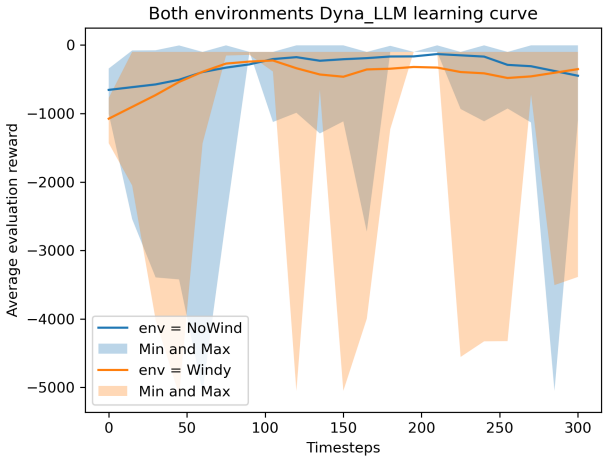

Figure 11: The average evaluation reward, minimum and maximum over timesteps for both environments. An evaluation is done every 15 timesteps. The average evaluation reward curves are calculated for 10 repetitions on 21 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. This figure show that the Dyna-LLM algorithm performs better (gets higher average evaluation results) in the environment without wind for almost all timesteps except timesteps 75 until 100. Additionally, this figure shows that the Dyna-LLM algorithm does map the optimal path for the first time in timestep 75 and never maps the optimal path for the windy environment. Not mapping the optimal solution is likely caused by the incorrect outputs of the LLM.

In Figures 10 and 11 can be seen that the Dyna-LLM algorithm performs better (gets higher average evaluation results) in the environment without wind for almost all timesteps except timesteps 75 until 100. Thus, Dyna-LLM performs better at the first timesteps for the environment without wind, unlike the Q-learning and Dyna-Q algorithms that both perform better in the windy environment at the first timesteps. As can be seen in Figure 11, Dyna-LLM never maps the optimal solution in the Q-table for the windy environment. Thus, in the windy environment the Q-table never converges to the optimal solution. Additionally, in the environment without wind it did not map the optimal path in the Q-table on average. The agent does map the optimal path for the first time at timestep 75 in the windless environment. Not mapping the optimal solution is likely caused by the incorrect outputs of the LLM.

As can be seen in Figure 10, the average evaluation reward is never -4 or approximately -17. The relatively large standard deviation in combination with the suboptimal average evaluation rewards could mean that the LLM outputs are incorrect. Incorrect outputs result in updating the Q-table with wrong rewards for certain transitions. While these rewards are not correct, the policy chooses the action as if the rewards were correct. The incorrect Q-table values could result in the greedy agent performing unwanted behaviour, such as choosing an suboptimal action. Thus, the potentially incorrect outputs of Tiny-Llama likely result in the suboptimal performance of the Dyna-LLM algorithm. The output quality and accuracy of the Tiny-Llama LLM, which is used as the transition and reward model in the Dyna-LLM algorithm, is discussed in section 4.5.

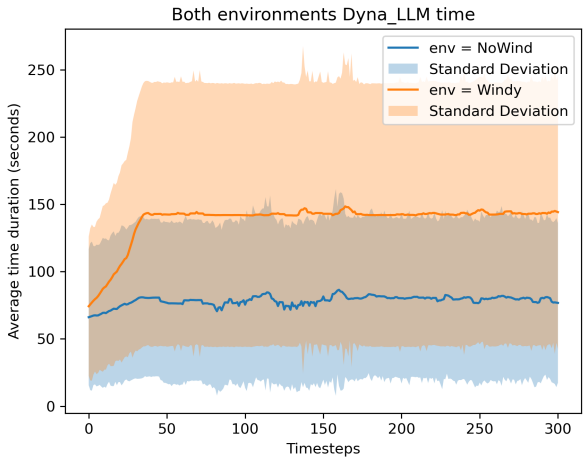

Figure 12: The average time duration and standard deviation over timesteps for both environments. The average time duration curves are calculated for 10 repetitions on 301 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. This figure shows that the average time duration increases until timestep 37 and then becomes roughly constant for both environments. It appears that the time duration is likely dependent on the input prompt length of the LLM. Additionally, it appears that the difference in roughly constant average time duration is primarily caused by the length of the shortest routes. The figure shows that the Dyna-LLM algorithm uses a time duration of approximately 78 and 138 seconds for the windless and windy environment respectively. Performing the action, updating the Q-table including the planning steps and LLM prompting in that time frame.

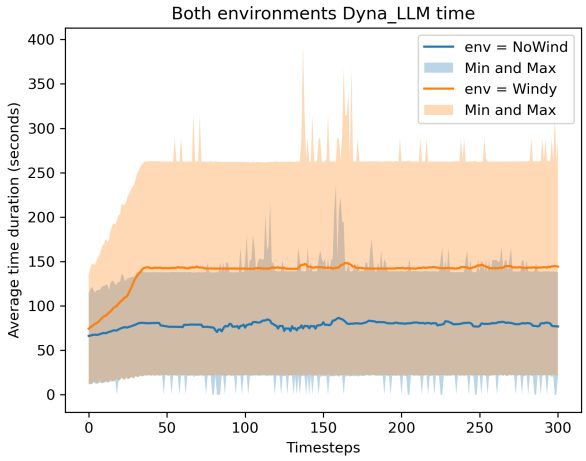

Figure 13: The average time duration, minimum and maximum over timesteps for both environments. The average time duration curves are calculated for 10 repetitions on 301 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. This figure shows that the average time duration increases until timestep 37 and then becomes roughly constant for both environments. It appears that the time duration is likely dependent on the input prompt length of the LLM. Additionally, it appears that the difference in roughly constant average time duration is primarily caused by the length of the shortest routes. The figure shows that the Dyna-LLM algorithm uses a time duration of approximately 78 and 138 seconds for the windless and windy environment respectively. Performing the action, updating the Q-table including the planning steps and LLM prompting in that time frame.

Figures 12 and 13, show that the average time duration increases until timestep 37 and then becomes roughly constant for both environments. It appears that this is because the history part has a maximal length of 37. At timestep 0, the sample length is 1 as one environment interaction is saved in the history at the end of the timestep. This history length grows with one interaction per timestep until it has 37 samples or interactions in the history part of the prompt, making the time duration measured, likely dependent on the input prompt length of the LLM.

It appears that the difference in roughly constant average time duration is primarily caused by the length of the shortest routes. The optimal path is shorter in the environment without wind than the optimal path in the windy environment. The difference in optimal path lengths could result in a difference in episode lengths. If on average an episode is shorter in the windless environment, compared to the windy environment, then more episodes are completed in the same amount of timesteps. Thus, the difference in episode lengths could result in the difference of average time duration.

Additionally, Figures 12 and 13 show that the Dyna-LLM algorithm uses a time duration of approximately 78 and 138 seconds for the windless and windy environment respectively. Performing the action, updating the Q-table including the planning steps and LLM prompting in that time frame.

4.4 Q-learning vs Dyna-Q vs Dyna-LLM

In Figures 14 and 15, the learning curves for the Dyna-LLM, Q-learning and Dyna-Q algorithms are plotted, denoted by the lines. Additionally, the associated standard deviation is plotted, denoted by the area.

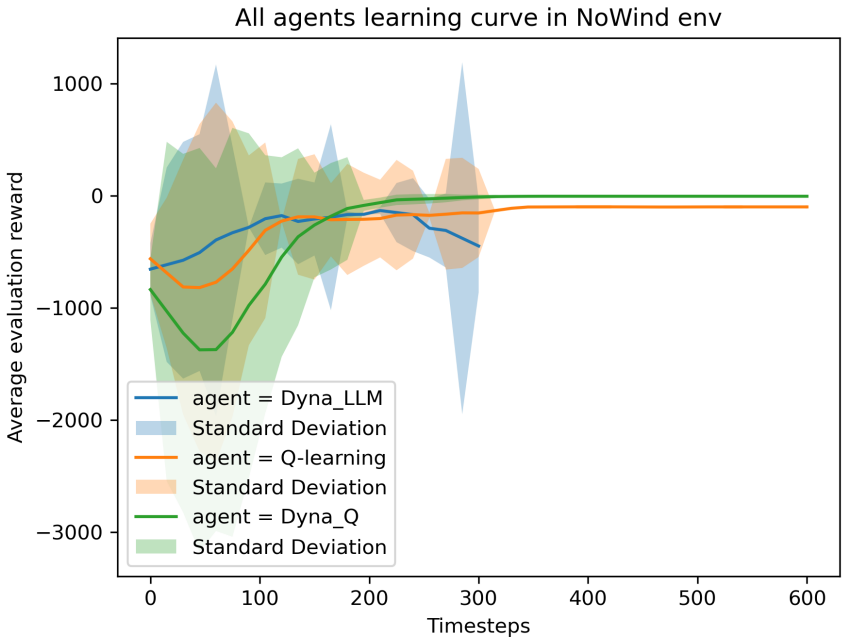

Figure 14: The average evaluation reward and standard deviation over timesteps for both environments. An evaluation is done every 15 timesteps. For the Dyna-LLM algorithm, the average evaluation reward curves are calculated for 10 repetitions on 21 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. For the Q-learning and Dyna-Q algorithms, the average evaluation reward curves are calculated for 40 repetitions on 41 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. This figure shows that the average evaluation rewards are lower for the Dyna-LLM algorithm at timestep 240 and up, compared to Q-learning and Dyna-Q. The lower average evaluation rewards of the Dyna-LLM are likely because of the LLM output is often incorrect. Furthermore, the average evaluation rewards of Dyna-LLM are lower than those of Dyna-Q after timestep 240. Due to the partially saved history, the LLM potentially gives incorrect outputs. The higher average evaluation of Dyna-LLM at the timesteps 30 until 105 is likely caused by the output of Tiny-Llama during the planning updates.

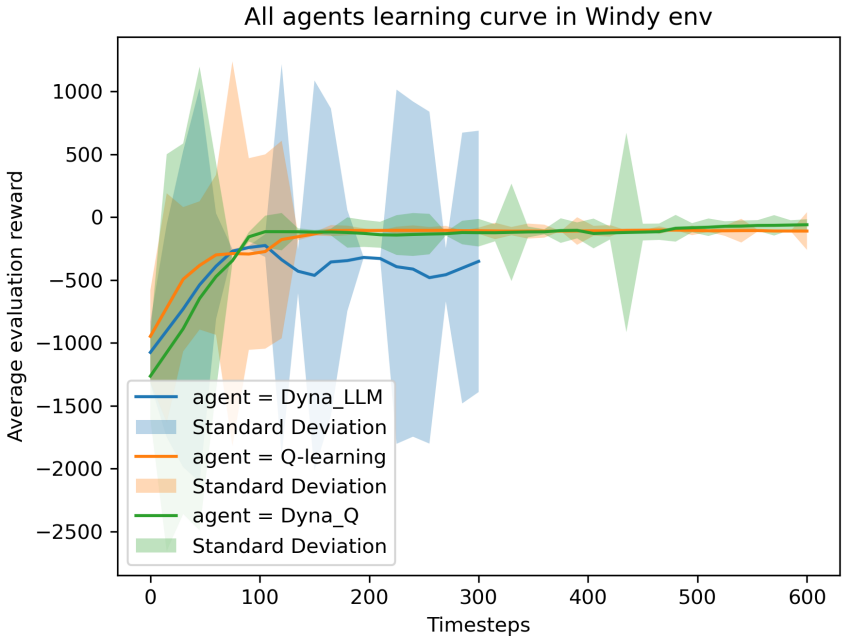

Figure 15: The average evaluation reward, minimum and maximum over timesteps for both environments. An evaluation is done every 15 timesteps. For the Dyna-LLM algorithm, the average evaluation reward curves are calculated for 10 repetitions on 21 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. For the Q-learning and Dyna-Q algorithms, the average evaluation reward curves are calculated for 40 repetitions on 41 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. This figure shows that the average evaluation rewards are lower for the Dyna-LLM algorithm at timestep 240 and up, compared to Q-learning and Dyna-Q. The lower average evaluation rewards of the Dyna-LLM are likely because of the LLM output is often incorrect. Furthermore, the average evaluation rewards of Dyna-LLM are lower than those of Dyna-Q after timestep 240. Due to the partially saved history, the LLM potentially gives incorrect outputs. The figure also shows that the Dyna-LLM algorithm has higher average evaluation rewards than Q-learning and Dyna-Q at the timestep 90. Because of the Savitzky-Golay filter that is used to smoothen the learning curve in the graph, the figure does not show that timestep 105 is also a timestep where Dyna-LLM has the highest average evaluation rewards. The differences between the average evaluation rewards of Dyna-LLM and the classical RL algorithms (Q-learning and Dyna-Q) are likely caused by the LLM output.

Figures 14 and 15 show that the average evaluation rewards are lower for the Dyna-LLM algorithm at timestep 240 and up, compared to Q-learning and Dyna-Q. The lower average evaluation rewards of the Dyna-LLM are likely because of the LLM output is often incorrect. Furthermore, the average evaluation rewards of Dyna-LLM are lower than those of Dyna-Q after timestep 240. Dyna-Q has used all the history to estimate the reward function and transition function. Thus, the Dyna-Q agent uses the entire history, while the Dyna-LLM agent does not. The maximal history length is set to 37 samples, because of the maximal prompt length of the LLM. The maximal prompt length makes the LLM rely on a smaller history, making the predictions less accurate. For example, consider a boulder is present at a certain state, but the effect of the boulder is not present in

the history part of the prompt. The LLM might give a transition as output that starts at a tile state and ends at a boulder state, which is incorrect. The LLM should give a next state that is also a tile state. Thus, due to the partially saved history, the LLM potentially gives incorrect outputs.

In Figure 14 can be seen that the Dyna-LLM algorithm has higher average evaluation rewards than Q-learning and Dyna-Q at the timesteps 30 until 105 in the windless environment. The higher average evaluation of Dyna-LLM at the timesteps 30 until 105 is likely caused by the output of Tiny-Llama during the planning updates. Thus, the higher average evaluation of Dyna-LLM at the timesteps 30 until 105 will be discussed in section 4.6 in the conclusion of the results.

Figure 15 shows that the Dyna-LLM algorithm has higher average evaluation rewards than Q-learning and Dyna-Q at timestep 90. Because of the Savitzky-Golay filter that is used to smoothen the learning curve in the graph, the figure does not show that timestep 105 is also a timestep where Dyna-LLM has the highest average evaluation rewards. The Dyna-LLM algorithm also has slightly higher average evaluation rewards than the Dyna-Q algorithm before timestep 90. As will be discussed in section4.6 in the conclusion of the results, the higher average evaluation rewards are likely caused by the LLM outputs.

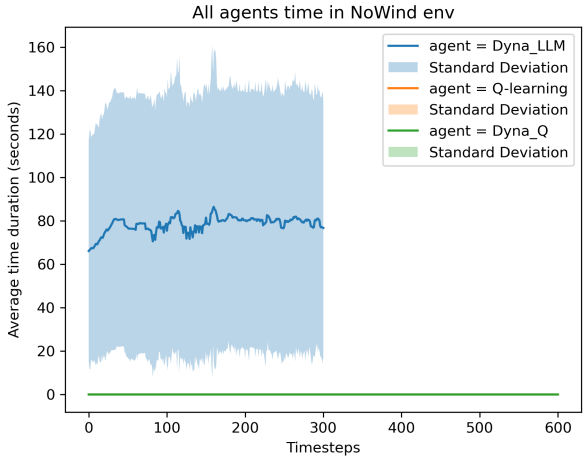

Figure 16: The average time duration and standard deviation over timesteps for both environments. For the Dyna-LLM algorithm, the average time duration reward curves are calculated for 10 repetitions on 301 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. For the Q-learning and Dyna-Q algorithms, the average time duration curves are calculated for 40 repetitions on 601 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. The result for the Q-learning algorithm is plotted where the green line is as well, therefore hiding the orange line. The significantly longer waiting times of Dyna-LLM compared to Q-learning and Dyna-Q is partially because of the Tiny-Llama is not always generating valid answers. Additionally, generating one next state and reward using an LLM is many times slower than generating the next state using the constructed model of the Dyna-Q algorithm. Because of the long time duration, sampling the environment instead of using an LLM to generate the next-state and reward might be less time consuming.

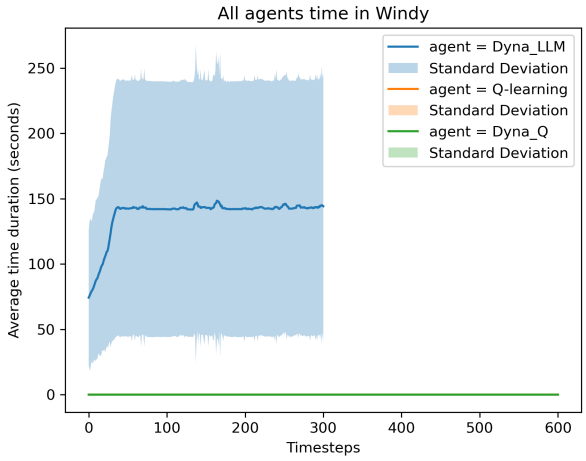

Figure 17: The average time duration and standard deviation over timesteps for both environments. For the Dyna-LLM algorithm, the average time duration reward curves are calculated for 10 repetitions on 301 timesteps, starting on timestep 0 with the last evaluation being on timestep 300. For the Q-learning and Dyna-Q algorithms, the average time duration curves are calculated for 40 repetitions on 601 timesteps, starting on timestep 0 with the last evaluation being on timestep 600. The result for the Q-learning algorithm is plotted where the green line is as well, therefore hiding the orange line. The significantly longer waiting times of Dyna-LLM compared to Q-learning and Dyna-Q is partially because of the Tiny-Llama is not always generating valid answers. Additionally, generating one next state and reward using an LLM is many times slower than generating the next state using the constructed model of the Dyna-Q algorithm. Because of the long time duration, sampling the environment instead of using an LLM to generate the next-state and reward might be less time consuming.

The the significantly longer waiting times of Dyna-LLM compared to Q-learning and Dyna-Q is partially because of the LLM not generating valid answers, making it have to do multiple loops of answer generation, which is time consuming. Additionally, generating one next state and reward using an LLM is many times slower than generating the next state using the constructed model of the Dyna-Q algorithm. This is seen in Figures 16 and 17, as Dyna-LLM is significantly slower than the other two algorithms. Since the benefit of sampling efficiency are possibly time or money, the slowness of the LLM generation makes this algorithm setup less desirable to use in a robotics setting. Choosing to sample the environment instead of using the learned model of the world of this particular Dyna-LLM algorithm using Tiny-Llama might have more benefits, since sampling the environment is more reliable than generation and might be faster in the real world instead of generating the next state and reward using the LLM.

4.5 LLM output comparison

Given the unexpected low results of the Dyna-LLM agent, to explore if the prompt or Tiny-Llama is the reason for the low results of the Dyna-LLM, the output of the Qwen3[8] LLM and Tiny-Llama[9] are compared. After further investigation of the prompt, I have found that the bounds of the environment are incorrect. This can be seen in the environment rules of the prompt. The bounds should be state from 0 to 24 instead of from state 1 to 25. The corrected prompt will be called the new prompt. The prompt that was used during the main experiment being called the old prompt. During the main experiment, state 25 was given as next state by Tiny-Llama, which is not a valid state in the environment. It appears that this happened because of the mistake in the old prompt.

4.5.1 Methodology

As was mentioned in the introduction section (1.2) the research questions are:

RQ1: Can Tiny-Llama improve sample efficiency as the Transition and Reward function of model-based reinforcement learning in a grid world problem?

RQ2: Are large LLMs better than small LLMs in model-based reinforcement learning?

RQ3: Does Tiny-Llama understand the environment mechanics, given the prompts used in this experiment?

The ability to generate the consequences of an state-action pair without it being in the history part of the prompt will be called the generalizability of the LLM, with the quality of the output of the LLM denoting how correct or complete the generated answer is. **RQ2** will be answered by comparing the generalizability and quality of the outputs of Tiny-Llama and Qwen3. **RQ3** will be answered using the resulting generalizability and quality of Tiny-Llama's outputs. Because of the mistake in the prompt, the generalizability and quality of the Tiny-Llama and Qwen3 LLM with the old prompt as well as the new prompt are compared.

There are 12 prompts used for this mini experiment. The prompts differ to show certain strengths or weaknesses that the LLMs may have, giving possible explanations to the results of the main experiment. In Table 3 the combinations of factors that determine a type of question is shown. These different types of questions help in understanding the key differences between the prompts used in this mini experiment. As was discussed in section 3.4, the question is the last part of the prompt that lets the LLM give the next state and reward, by giving the beginning of the transition in the environment. The question is of the form: State: """+str(s)+""", Action: Agent moves """+`action`+""".", which can be seen in Prompt 1, at line 31. In Table 4 the differences between the prompts are given.

[8] https://huggingface.co/Qwen/Qwen3-235B-A22B

[9] https://huggingface.co/TinyLlama/TinyLlama-1.1B-Chat-v1.0

Type of question	Current state is in history	State-action combination is in history
1	True	True
2	True	False
3	False	False

Table 3: The different types of questions in the prompt. Note that the current state is in the history part of the prompt if the state-action combination is in the history. Because the state-action combination is a pair that consists of the current state and the action that the agent performs on that state.

Prompt number	Type of prompt	Environment	Type of question
3	old	no wind	1
4	old	no wind	2
5	old	no wind	3
6	old	windy	1
7	old	windy	2
8	old	windy	3
9	new	no wind	1
10	new	no wind	2
11	new	no wind	3
12	new	windy	1
13	new	windy	2
14	new	windy	3

Table 4: Key differences of the prompts. Showing that the differences of the prompts are caused by three factors: type of prompt, environment and the type of question which can be seen in Table 3.

Table 4 shows the key differences of the prompts. The differences of the prompts are caused by three factors: type of prompt, environment and the type of question. By using these different prompts, different aspects of the LLMs will be tested. Highlighting strengths or weaknesses in different situations that could be present during the planning updates of the Dyna-LLM algorithm during the main experiment.

The type of prompt is old or new, with old being the environment rules that were used during the main experiment and new being the environment rules after they have been corrected.

The environment factor determines what is in the history part of the prompt. The history is the same for the same type of environment, but differs between environment types. So all windless and windy environments and have the same history, respectively. Prompt 6.5 and prompt 12.5 are similar to prompt 6 and 12 respectively, with the only difference being the history of the prompt to further investigate how the Tiny-Llama determines the answers to the questions.

Furthermore, the type of question determines what part of the question is known by the agent using only the prompt. Otherwise said, the type of question determines if the asked current state and state-action pair (the action that the agent is tasked to perform from the current state) is present

in the history. Thus, there are three different questions per environment or history. As can be seen in Table 3, the first question asks the LLM to generate the next state and reward of a state-action combination that appears in the history. The second question: for a current state that appears in the history with an action that does not appear together as state-action combination in the history. Thus, this second kind of question uses an unknown state-action combination for a known current state. The third kind of question uses a current state and state-action combination that is not in the history. To clarify, prompt 3 and 9 ($3 + 6 = 9$), 4 and 10 ($4 + 6 = 10$), 5 and 11 ($5 + 6 = 11$) and so on are the same respectively, with the only difference being the corrected environment rules, which is the difference between the old type of prompt and the new type of prompt.

The complete prompts and resulting LLM outputs can be found on the GitHub[10] in the LLM_prompts_and_outputs.txt file.

Furthermore, to generate the output of Tiny-Llama, python code is used, specifically the LLM_output_test.py. Qwen3 is tested by giving the prompt to the LLM online. There are the steps that are performed in order to get results:

1. Use this link: https://huggingface.co/chat/settings/Qwen/Qwen3-235B-A22B
2. Click on: New chat.
3. Copy the corresponding prompt and paste it in the prompt box (where it says: ask anything).
4. The prompt box will be empty, but the pasted prompt can be seen as pasted content above the prompt box.
5. To generate an answer, type: continue, in the prompt box and hit enter.
6. Afterwards you should see the output of the LLM, if it gives an error, try the steps again.

During this test, when it asks for a hugging face account, log in or create an account and confirm your email address.

4.5.2 Results

It should be noted that the LLM_output_validation_errors.txt of the main experiment shows that the amount of errors for the windy environment are about 25% of the total errors.

The results from Tiny-Llama and Qwen3 can be seen in Table 5. Using information of Table 3 together with Table 5, show that Tiny-Llama has a low quality answer (an incomplete answer) for Prompt 5. Given that there is nothing like the question in the history (because it is a question of type 3), it appears that answer is incomplete because the LLM can not repeat a part of the history. With this prompt Tiny-Llama can not generate the next state when it was not seen in the history part of this prompt. Qwen3 LLM can generate the next state correctly. It appears that the Tiny-Llama model has limited reasoning skills and that Qwen3 is able to at least give a predicted consequence based on the environment rules when no inconsistencies have been seen in the history.

[10] https://github.com/Alex3124536745869708/thesis

The output of Qwen3 for the prompts 8 and 14 are complete but not of the correct syntax. The location is not asked in the prompt but is given by Qwen3, showing that Qwen3 can generate more freely compared to Tiny-Llama, which only repeats part of the prompt to answer the questions.

Tiny-Llama only answers the correctly for questions of type 1. To clarify, questions of type 1 have the state-action pair and consequences (next-state and reward) present in the history of the prompt. Qwen3 answers all questions correctly given the environment rules. The only prompt Qwen3 answered incorrectly is Prompt 4. This incorrect answer is likely because of the environment rules that are present in the prompt. Those rules dictate that the agent cannot go right if it is against the border of the grid world. Given the environment rules Qwen3 answered correctly following the knowledge given by the prompt.

The outputs of the prompts: 6, 6.5, 12 and 12.5, are correctly answered by Tiny-Llama and Qwen3. These prompts show that Tiny-Llama repeats the correct answer option that occurred the most in history. The answers of Qwen3 are correct as well, showing that it uses the history of the prompt in combination with the environment rules to determine the next-state and action.

Tiny-Llama does not give a correct possible next-states after correcting the environment rules (new prompt type) for prompts of question type 2 and 3. Instead, it repeats the most common matching history of the prompt like with the old prompt type. Qwen3 gives correct answers and appears to be using the environment rules in combination with the history of the prompt. Qwen3 appears to be using the environment rules in combination with the history of the prompt, because Prompt 4 was correctly answered given the environment rules (but incorrect given the environment) and Prompt 10 (the same prompt as Prompt 4 but with different environment rules) was correctly answered.

Prompt number	QT	QQ	CT	CQ
3	+	+	yes	yes
4	+	+	no	not for env., yes given available knowledge
5	-	+	no	yes
6	+	+	yes	yes
6.5	+	+	yes	yes
7	+	+	no	yes
8	+	+ (/-)	no	yes
9	+	+	yes	yes
10	+	+	no	yes
11	+	+	no	yes
12	+	+	yes	yes
12.5	+	+	yes	yes
13	+	+	no	yes
14	+	+ (/-)	no	yes

Table 5: Results for all prompts. QT means: quality Tiny-Llama, QQ means: quality Qwen3, CT means: correctly answered by Tiny-Llama, CQ means: correctly answered by Qwen3. + means: a complete answer, - means: an incomplete answer, + (/-) means: the answer was complete but not of the right syntax. Tiny-Llama has an incomplete answer for prompt 5 and a complete answer for the rest of the prompts. Qwen3 has complete answers for all prompts but extra information for prompts 8 and 14. Tiny-Llama only has correct answers for prompts with question type 1 (prompts: 3, 6, 6.5, 9, 12 and 12.5). Qwen3 has correct answers for all prompts given the provided knowledge in the prompt. However, Prompt 4 is not correct given the environment. These results show that Tiny-Llama repeats part of the history to answer the question and likely fails to answer correctly when that is not an option. Qwen3 appears to use reasoning, combined with the environment rules and history of the prompt to determine the next-state and reward.

4.5.3 Conclusion

For Tiny-Llama, the prompts that have the same outcomes are all prompts that differ only in the type of prompt (old or new), except the prompts with question type 3. So prompt 3 has the same outcome as prompt 9, prompt 4 as 10 repeating that for all prompts, except for the prompt pairs: {5 and 11} and {8 and 14}. In total the only prompts that are correctly answered by Tiny-Llama are the prompts with question type 1. The other question types do not result in consistent correct responses. While it may happen that Tiny-Llama answers a question correctly from a prompt other than question type 1, it appears that that would not happen consistently, since Tiny-Llama uses the history to determine the next state, not because of the ability of Tiny-Llama to generate the next state accurately using reasoning. Only type 1 questions are correctly answered. Thus, the results show that Tiny-Llama in combination with both old and new prompts does not understand the environment mechanics and only repeats or shuffles and uses parts that it has seen in the history part of the prompt. With the new and old prompt, it appears that Tiny-Llama can not reason given the environment rules and can not understand the impact of the wind on the transitions in the environment. Qwen3 answers all the questions correctly given the environment rules in the prompt. It appears that Qwen3 uses the environment rules to reason what would happen and prioritises the

history.

Potential impact on the main experiment results Using the old prompt type makes Tiny-Llama less likely to give a next state in the environment without wind than when using the new prompt type. Namely, prompt 5 is the only prompt that does not give a next state using Tiny-Llama. The other question type 3 prompts do give a next state. However, all the answers to the question types 2 and 3 were wrong in this mini experiment. Being unable to update the Q-table with incorrect information in the main experiment using Dyna-LLM could increase the performance of the Dyna-LLM algorithm, compared to if the agent were able to update the Q-table with incorrect information. The question type that results in an Q-table update because of the LLM output validation function of the Dyna-LLM agent could only be of type 1 or 2. This increases the chance that a question type 1 is asked, as a type 3 question results in generating a random new question. This improves the overall accuracy of the Tiny-Llama generated next states. Therefore, the use of the old prompt instead of the new prompt for Tiny-Llama would result in less Q-table updates that would conflict with the convergence of the Q-table, as less updates are done that represent wrong environment mechanics or transition function.

To summarise, while the answer quality of the old prompt was only worse than the new prompt in prompt 5, the inability of generating a next state might have helped the Dyna-LLM agent in the main experiment. In conclusion, it would seem logical that using the old prompt for the environment without wind in the main experiment would result in a relatively better sampling efficiency, compared to using the new prompt. It appears that using the new prompt instead of the old prompt does not make a difference in the windy environment performance of the Dyna-LLM algorithm.

It should be noted that the LLM_output_validation_errors.txt shows that the amount of errors for the windy environment are about 25% of the total errors. While no syntax errors were observed in this mini experiment, the syntax errors were present in the main experiment. The syntax errors for the windy environment were present for questions of type 2 and 3 but less likely to occur as in the windless environment. Thus, the syntax errors in the windy environment, like in the windless environment, likely result in an improved sample efficiency.

It appears that using Qwen3 instead of Tiny-Llama would improve the cumulative reward performance of the Dyna-LLM algorithm (making the resulting mean cumulative reward of the valuation of the Dyna-LLM Q-table higher). Given that Qwen3 answered all the questions correctly. However, Qwen3 never used the influence of the wind in its answer in a windy environment history, unless question type 1 was used, where the answer is directly in the history. A possible reason may be that Qwen3 probably does not reason using the wind and only uses the environment rules in the prompt unless the history contradicts the initial answer of Qwen3. Qwen3 also updates the Q-table as if boulders and cliffs are not there when there is no proof of them in the history. Tiny Llama has this problem as well. Thus, using Qwen3 and Tiny-Llama with a limited history is suboptimal. The limited history can result in incorrect updates that do not represent the transition function and reward function of the environment, which reduces cumulative reward performance.

4.6 Conclusion

LLM output comparison Tiny-Llama in combination with both old and new prompts does not understand the environment mechanics and only repeats or shuffles and uses parts that it has seen in the history part of the prompt. The inability to reason using the environment mechanics likely results in consistently correct question type 1 answers for the old and new prompt. Questions of type 2 and 3 are not answered correctly for both prompt types. Additionally, using the environment without wind in combination with the old prompt, makes the Tiny-Llama less able to give answers to questions of type 3. In that context, Tiny-Llama will likely perform better with the old prompt type (that has mistakes) than with the new prompt type.

From LLM_output_validation_errors.txt of the main experiment we saw that Tiny-Llama also gives syntax errors in the output for the windless environment. Given that the answers to questions of type 2 and 3 are often incorrect, the syntax errors of the LLM answers could improve the average evaluation reward of the Dyna-LLM algorithm.

Qwen3 answers all the questions correctly given the knowledge that it has. However, that does not guaranty that all answers to questions of type 2 and 3 are correct given the environment, since not all sampled knowledge is able to be saved in the prompt because of the prompt length limit. It appears that using a limited history reduces the validation rewards, compared to using an unlimited history which is not possible given the LLM's maximum prompt lengths. Because not only type 1 but also type 2 and 3 questions are asked, the Dyna-LLM algorithm using Qwen3 instead of using Tiny-Llama will probably get higher mean cumulative rewards given by the validation of the Q-table.

Q-learning results Q-learning gets higher average evaluation results before timestep 330 in the windy environment, compared to the windless environment. A possible reason may be that the tile that crosses the cliff states has a low value, since all actions on that state may result in a low reward. A low value for the cliff-crossing-tile-state makes the agent explore mainly the path around the shortest path, resulting in a safer exploration in the windy environment, compared to the windless environment. It appears that the safer exploration results in higher average evaluation result before timestep 330 in the windy environment compared to the windless environment.

Furthermore, after timestep 330, the average evaluation reward is mostly higher for the windless environment.

Q-learning mapped the optimal path for the environment without wind but not for the environment with wind by getting maximum values of -4 but not a value close to -17 for the windless and windy environments respectively. A possible reason could be that the greater amount of state-action pairs to explore when exploring the cliff-avoiding-path in combination with the wind that blows the agent back results in more timesteps needed to map the optimal path in the Q-table.

Additionally, the Q-learning algorithm uses a time duration of approximately $6 * 10^{-5}$ seconds to perform the action and update the Q-table.

Dyna-Q results Dyna-Q mostly got higher average evaluations in a windy environment before timestep 160. After timestep 160, the average evaluation reward is mostly higher for the environment without wind.

A possible reason for the higher early rewards in the windy environment is that all the state-action pairs for the first cliff-crossing-state (state 17) have a low value in the Q-table, making the agent often steer away from the dangerous cliff-crossing-tile-states. Like with Q-learning, the safer exploration results in less negative results for the agent in the windy state after some timesteps in the beginning, compared to the agent in the windless environment.

Dyna-Q maps the optimal path on average in the windless environment, but not in the windy environment. It appears that this is caused by the greater amount of states for the cliff avoiding path in combination with the wind blowing the agent back, like with Q-learning. The amount of timesteps (601 in total) is likely not enough to converge the Q-table in all repetitions.

Dyna-Q algorithm uses a time duration of approximately $8 * 10^{-4}$ seconds to perform the action and update the Q-table including the planning steps.

Dyna-LLM results The LLM_output_validation_errors.txt file does not contain the error that the output text before the question is different than the input text. The lack of the error means that only answers with valid syntax are used to update the Q-table.

As was discussed in section 4.3, Dyna-LLM performs better at the first timesteps for the environment without wind, unlike the other algorithms that perform better in the windy environment at the first timesteps. This is likely because Tiny-Llama, which is used as the LLM in the Dyna-LLM algorithm, has a lower probability to answer questions of type 3 in the environment without wind as was discussed in the conclusion of the LLM output comparison (section 4.6). Because the questions of type 3 often yield invalid answers, they are less often used to update the Q-table. This increases the relative amount of type 1 and 2 questions that are used to update the Q-table. Since questions of type 1 are often correct (because the Tiny-Llama has an example of the right answer in the history part of the prompt) the average evaluation reward is relatively high for the environment without wind for the Dyna-LLM algorithm, compared to the windy environment.

The average evaluation reward is never -4 or approximately -17. It appears that the incorrect outputs of the Tiny-Llama result in updating the Q-table using incorrect transitions and rewards. The incorrectly updated Q-table likely results in suboptimal behaviour of the greedy agent (the agent that uses a greedy policy) during the evaluations. This suboptimal behaviour is the probable reason the the average evaluation reward is never -4 or approximately -17 for the windless and windy environment respectively.

The increasing and roughly becoming constant of the average duration time is likely caused by the history length of the prompt. The prompt grows in length per timestep because a sample is added to the history in each timestep. The history has a maximum length of 37 samples, likely making the average time duration roughly constant after timestep 37.

It appears that the difference in the roughly constant average time duration is primarily caused by the length of the shortest routes. If on average an episode is shorter in the windless environment, compared to the windy environment, then more episodes are completed in the same amount of timesteps. Thus, the difference in episode lengths could result in the difference of average time duration.

Q-learning vs Dyna-Q vs Dyna-LLM results The average evaluation rewards for the Dyna-LLM are lower for the timesteps 240 and up, compared to the Q-learning and Dyna-Q algorithms (Figures 14 and 15). Thus, the sample efficiency of Dyna-LLM is lower than the sample efficiency of both Q-learning and Dyna-Q. The lower performance is likely caused by the low accuracy of Tiny-Llama when answering questions of type 2 and 3, as was discussed in the LLM output comparison conclusion in section 4.6. Updating the Q-table can lead to unwanted behaviour of the agent, since the agent selects an action to perform by using the information in the Q-table. The evaluation only chooses the best possible action to perform at a given state, making the incorrect LLM outputs have a negative effect on the average evaluation rewards. In the case of questions of type 2 and 3, the LLM could answer incorrectly as was discussed in the conclusion in section 4.6. The incorrect answers can be caused by the limited sample history, making the LLM output potentially incorrect next-stated and rewards. The incorrect transitions are then used to update the Q-table which can lead to suboptimal decision making.

As was discussed in section 4.4 and can be seen in Figure 14, the higher average evaluation rewards of Dyna-LLM at the timesteps 30 until 105 is likely caused by the output of Tiny-Llama during the planning updates. The higher average evaluation rewards are likely caused because Tiny-Llama often does not validly answer questions of type 3. It appears that Tiny-Llama gets more questions of type 1, as was discussed in the conclusion of the Dyna-LLM results. The history prompt only has the 37 last transitions saved. More questions of type 1 (questions with answers that are completely in the history) in combination with only having the last 37 questions in the history means that the Dyna-LLM algorithm behaves similar to the Prioritised sweeping algorithm [MA93]. The Dyna-LLM effectively updates the one of the last 37 state-action pairs when it is asked a type 1 question. When more than 37 state-action pairs are sampled and something important has happened e.g., in the event that the goal is found for the first time, Dyna-LLM has more chance than Dyna-Q to update a state-action pair that leads to the goal. The Prioritised Sweeping agent calculates a high priority for the state-action pairs that are connected to the goal state, resulting in updating those state-action pairs in the Q-table. It appears that Dyna-LLM receives higher average evaluation rewards than Dyna-Q in the beginning, because Dyna-LLM updates the Q-table for state-action pairs that have on average more priority than the state-action pairs that Dyna-Q has selected to update. Given [MA93], updating state-action pairs that have high priority instead of state-action pairs that have low priority results in faster convergence of the Q-table relative to Dyna-Q and Q-learning, increasing sampling efficiency.

The Dyna-LLM algorithm also has higher average evaluation rewards than the Dyna-Q algorithm before timestep 90 as can be seen in Figure 15. As was discussed in the LLM output comparison conclusion (section 4.6), invalid answers of the LLM (the LLM output is not of the right syntax) increase the chance of a question of type 1 instead of a question of type 2 or 3 where the error occurs. This improves the accuracy of the used LLM output, since questions of type 1 are

more often correct than questions of type 2 and 3. The accuracy increase of the used LLM outputs is less than in the windy environment because an invalid output is given less often. The increase of the accuracy, means more state-action pairs that are updated are maximally 37 timesteps old. It appears that simulating these relatively new state-action pairs results in higher average evaluation rewards than the Dyna-Q in the windy environment. As was said in the previous paragraph, Moore and Atkeson [MA93] show us that updating the Q-table for state-action pairs that have a relatively higher priority results in faster convergence of the Q-table relative to Dyna-Q and Q-learning, increasing sampling efficiency.

The the significantly longer waiting times of Dyna-LLM compared to the other two algorithms is partially because of the LLM not generating valid answers, making it have to do multiple loops of answer generation, which is time consuming. Additionally, generating one next state and reward using an LLM is many times slower than generating the next state using the constructed model of the Dyna-Q algorithm. TH

5 Conclusions and Further Research

In this section, the problem statement and research questions are answered. The limitations are discussed and a direction for future work is given.

5.1 Problem statement

The problem statement and research questions were previously mentioned in the introduction (section 1.2) and are repeated in this section for convenience. The problem statement is:

PS: Can LLM improve model-based reinforcement learning?

In order to answer the problem statement the following research questions (RQ's) are used:

RQ1: Can Tiny-Llama improve sample efficiency as the Transition and Reward function of model-based reinforcement learning in a grid world problem?

RQ2: Are large LLMs better than small LLMs in model-based reinforcement learning?

RQ3: Does Tiny-Llama understand the environment mechanics, given the prompts used in this experiment?

In conclusion, the average evaluation rewards for the Dyna-LLM are lower for the timesteps 240 and up compared to the Q-learning and Dyna-Q algorithms. Given this result, it appears that Tiny-Llama does not improve sample efficiency as the Transition and Reward function of model-based reinforcement learning in a grid world problem, answering **RQ1**. As was mentioned in section 4.5, questions of a certain type are prompts with at the end a question that lets the LLM generate the next-state and reward pair given a current-state action pair. In the case of questions of type 1, the current-state action pair with the corresponding outcome is present in the history of the prompt. In the case for question type 2 and 3, the outcome corresponding to the question is not present in the history in the prompt. The sample efficiency of Dyna-LLM is lower than the sample efficiency of both Q-learning and Dyna-Q. The lower sample efficiency is likely caused by the low accuracy of Tiny-Llama when answering questions of type 2 and 3.

Tiny-Llama only accurately answers questions of type 1 and Qwen3 accurately answers questions of types 1, 2 and 3. Because Qwen3 answered all the questions correctly (high generalizability and quality of the output) given the knowledge that it has, the validation rewards of Dyna-LLM will likely improve when using Qwen3 instead of Tiny-Llama. The likely improved performance of Dyna-LLM when using Qwen3 instead of Tiny-Llama lets us conclude that large LLMs are likely better than small LLMs in model-based reinforcement learning, answering **RQ2**.

It appears that Tiny-Llama uses the history to determine the next-state, since it incorrectly answers all the questions of type 2 and 3 in the mini experiment (section 4.5). Tiny-Llama in combination with both old and new prompts does not understand the environment mechanics and only repeats or shuffles and uses parts that it has seen in the history part of the prompt, answering **RQ3**.

In the related work, the LLM is used as action suggester and they validate the reward by sampling the environment before updating the Q-table. It appears that their approach is more LLM error resistant than my approach (Dyna-LLM). With LLM error resistance, the ability to converge to the optimal solution despite incorrect LLM outputs is meant. When using the LLM as action suggester, the LLM output is validated using the environment. By validating the output, a wrong LLM output does not update the Q-table with incorrect information. In the case of a wrong LLM output, Dyna-LLM does update the Q-table incorrectly. The possibility of updating the Q-table incorrectly makes the average evaluation rewards highly dependent on the capabilities of the LLM. Given that large LLMs are likely better than small LLMs in model-based reinforcement learning (**RQ2** answer) and that Tiny-Llama does not understand the environment mechanics (**RQ3** answer) it is logical that Dyna-LLM using Tiny-Llama has a lower sample efficiency than Dyna-Q and Q-learning (**RQ1** answer). Because Dyna-LLM does not improve the sample efficiency compared to Dyna-Q or Q-learning, the **PS** can not be answered. While others have shown that LLM can improve RL, we did not show that LLM can improve Model-based RL using Dyna-LLM, answering the **PS**.

5.2 Limitations

We are aware that our research may have five limitations. The first is the amount of timesteps used in the main experiment. 601 timesteps for Q-learning and Dyna-Q and 301 timesteps for Dyna-LLM are used because mainly the LLM inference time made the main experiment duration a week long.

The second is the amount of repetitions for Dyna-LLM. Not more than 10 repetitions were used for the Dyna-LLM algorithm because of the long LLM inference time duration.

The third limitation is the amount of prompts used in the mini experiment. Using more prompts in the mini experiment per prompt type would make the findings more significant. Using more prompts would take more time that was unavailable during this thesis.

The fourth limitation is that the Qwen3 outputs are generated using an online chat function. Therefore, a seed is not set, possibly compromising the reproducibility of the results of the mini experiment.

The fifth limitation is that the LLM used in the main experiment: Tiny-Llama, is fine-tuned on conversation. Therefore, fine-tuning the LLM on RL outputs and reasoning with them could improve the performance of the LLM. This potential increase of performance might lead to different results, potentially answering the **PS**.

Finally, As was discussed in the previous section, the Dyna-LLM algorithm directly updates the Q-table. Because of the direct Q-table updates without using the environment to validate the LLM output, the performance of the Dyna-LLM algorithm is dependent on the LLM capabilities. This dependence on the LLM makes the Dyna-LLM algorithm error prone. Because of this dependence on the LLM, the performance of the Dyna-LLM algorithm is more likely to be lower, compared to not having this dependence. This means that the Dyna-LLM algorithm is likely to not be able to answer **PS**.

The first three limitations reveal the difficulty of collecting data on LLMs because of the inference time of LLMs. Collecting more data would improve the reproducibility and significance of the findings of both the main experiment and the mini experiment. Always using a seed would improve the reproducibility of the mini experiment. Fine-tuning the LLM or using a different LLM, as well as using a different algorithm than Dyna-LLM, would positionally improve the performance of said algorithm, and more likely answer the **PS**.

5.3 Future work

The limitations because of LLM inference time, can be resolved in future research by allocating more time before doing research to be available when doing research. These limitations can also be reduced by doing experiments in parallel on multiple devices.

Additionally, this thesis revealed several insights that can be used to improve future approaches of using LLMs in model-based reinforcement learning.

Firstly, Tiny-Llama's inference time (the time it takes to give an output) increases when increasing the prompt input length. When designing the prompt for the LLM, making the prompt as short as possible reduces inference time per prompt.

Secondly, the limited history length reduces LLM output accuracy compared to the environment. The limited history makes it so that not all outcomes of the sampled state-action pairs are saved. The discarding of samples likely reduces sample efficiency, compared to not discarding samples.

Thirdly, updating state-action pairs with high priority as is done in Prioritised Sweeping likely improves sample efficiency compared to randomly updating state-action pairs as is done in Q-learning. This was seen in the results as Dyna-LLM behaved more like Prioritised sweeping and received better early results than Dyna-Q.

Fourthly, using a model that has more parameters (and has better reasoning) than Tiny-Llama likely improves the evaluation results of the model per timestep.

Lastly, using the LLM to directly alter the Q-table without checking if the output corresponds to the environment mechanics can make the agent not able to converge to the optimal solution.

In conclusion, future work should aim to mitigate the weaknesses mentioned in the insights. For example, a different algorithm can be constructed using these insights, possibly improving the performance relative to the Dyna-LLM algorithm. Additionally, improving the LLM instead of the Dyna-LLM algorithm can possibly improve the performance relative to the Dyna-LLM algorithm as well. For example, by fine-tuning the LLM on RL data and reasoning.

References

[ABB+22] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022.

[BBK+24] Siddhant Bhambri, Amrita Bhattacharjee, Durgesh Kalwar, Lin Guan, Huan Liu, and Subbarao Kambhampati. Extracting heuristics from large language models for reward shaping in reinforcement learning, 2024.

[Bel57] Richard Bellman. A markovian decision process. *Journal of Mathematics and Mechanics*, 6(5):679–684, 1957.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[CYD+24] Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with scaled ai feedback, 2024.

[CZC+24] Yuji Cao, Huan Zhao, Yuheng Cheng, Ting Shu, Yue Chen, Guolong Liu, Gaoqi Liang, Junhua Zhao, Jinyue Yan, and Yun Li. Survey on large language model-enhanced reinforcement learning: Concept, taxonomy, and methods. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21, 2024.

[DYZ25] Thang Duong, Minglai Yang, and Chicheng Zhang. Improving the data-efficiency of reinforcement learning by warm-starting with llm, 2025.

[JSL+24] Peng Jiang, Christian Sonne, Wangliang Li, Fengqi You, and Siming You. Preventing the immense increase in the life-cycle energy and carbon footprints of llm-powered intelligent chatbots. *Engineering*, 40:202–210, 2024.

[KSR+23] Thommen George Karimpanal, Laknath Buddhika Semage, Santu Rana, Hung Le, Truyen Tran, Sunil Gupta, and Svetha Venkatesh. Lagr-seq: Language-guided reinforcement learning with sample-efficient querying, 2023.

[Lee24] Y.-S. Lee. Analysis of small large language models (llms). *International Journal of Advanced Smart Convergence*, 13(4):155–160, 2024.

[LHZ+24] Zhihan Liu, Hao Hu, Shenao Zhang, Hongyi Guo, Shuqi Ke, Boyi Liu, and Zhaoran Wang. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency, 2024.

[MA93] Andrew W. Moore and Christopher G. Atkeson. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, Oct 1993.

[MCH+20] Christopher D. Manning, Kevin Clark, John Hewitt, Urvashi Khandelwal, and Omer Levy. Emergent linguistic structure in artificial neural networks trained by self-supervision. *Proceedings of the National Academy of Sciences*, 117(48):30046–30054, 2020.

[MG24] Joy Mahapatra and Utpal Garain. Impact of model size on fine-tuned llm performance in data-to-text generation: A state-of-the-art investigation, 2024.

[NHR99] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[Rao00] Rajesh Rao. Reinforcement learning: An introduction; r.s. sutton, a.g. barto (eds.). *Neural Networks*, 13:133–135, 01 2000.

[SB20] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction, second edition*. 2020.

[UXL+22] Ike Uchendu, Ted Xiao, Yao Lu, Banghua Zhu, Mengyuan Yan, Joséphine Simon, Matt Bennice, Chuyuan Kelly Fu, Cong Ma, Jiantao Jiao, Sergey Levine, and Karol Hausman. Jump-start reinforcement learning. In *NeurIPS 2021 Robot Learning Workshop, RSS 2022 Scaling Robot Learning Workshop*, 2022.

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[WWS+23] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[ZCL+24] Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and adapting environments via llms for training embodied agents, 2024.

Appendix

Here, the complete LLM input prompt of the main experiment can be found. The line numbering is not part of the prompt.

Prompt 1. (Complete) LLM input prompt

1. Role:
2. You are an AI that simulates an environment for model-based reinforcement learning.
3.
4. Environment Rules:
5. The agent is in a 5×5 grid maze with states numbered from 1 to 25, structured as follows:
6. Top-left corner: State 1.
7. Top-right corner: State 5.
8. Bottom-left corner: State 21.
9. Bottom-right corner: State 25.
10. The agent moves up, down, left, or right, but cannot move outside the grid.
11. Each move results in a reward of -1 unless otherwise specified.
12. Use past interactions if available.
13.
14. If the action is new, apply logical reasoning based on the grid structure:
15. Moving left decreases the state number by 1, unless at the left boundary.
16. Moving right increases the state number by 1, unless at the right boundary.
17. Moving up decreases the state by 5, unless at the top boundary.
18. Moving down increases the state by 5, unless at the bottom boundary.
19. Circumstance at the beginning of the line means that YOU should reason with the effect the wind could have on the outcome of the action.
20.
21. You must return the following after an action:
22. state, location, action, reward, next_state, next_location.

23.

24. Past Interactions:

25. Context: The action from a certain state might result in a different next_state than the following action because of environment circumstanses.

26. In case this happens, write: "Circumstance" in the output.

27. """+`history_copy`+"""

28. Now Continue:

29. Follow past interaction patterns if available. If the interaction is new, use the movement rules to determine the correct outcome.

30.

31. State: """+str(s)+""", Action: Agent moves """+`action`+""".