

Article

Evaluation of Multi-Model Architecture Against Single-Model PPO in the MiniGrid Environment

Igor Ciganović ^{1,*}, Aleksandar Pluškoski ¹, Miloš Jovanović ^{2,3} and Jelena Vasiljević ¹¹ School of Computing, University Union, 11000 Belgrade, Serbia; apluskoski317d@raf.rs (A.P.)² Technical Test Center, Ministry of Defense, 11000 Belgrade, Serbia³ Faculty of Information Technology, Belgrade Metropolitan University, 11000 Belgrade, Serbia

* Correspondence: iciganovic217d@raf.rs

Abstract

Reward sparsity is a considerable problem in reinforcement learning when solving complex problems. In an environment where the reward is given only after successfully solving a problem in multiple steps, the signals for optimization can be too weak, preventing the agent from learning. However, this can be solved by breaking the problem into an array of simpler tasks that the agent can learn to perform. This paper presents a possible solution that uses a mixture-of-experts architecture. Separate models are used for planning, routing expert outputs, and solving singular sub-tasks. The solution was implemented in the MiniGrid environment, which allows for the design of arbitrarily complex problems. The results presented in this paper show that a multi-model agent can solve more complex problems comprising multiple steps than a single reinforcement learning agent.

Keywords: machine learning (ML); artificial intelligence (AI); deep learning; reinforcement learning (RL); neural networks; MiniGrid; large language model; mixture of experts

1. Introduction

Reinforcement learning (RL) is a subset of machine learning (ML) algorithms. As with other ML algorithms, RL uses large amounts of data to build a statistical model. The difference is in how the data is generated. Instead of a previously assembled “static” dataset, RL algorithms generate data during training. RL is mostly used for control tasks. The agent observes the environment’s state input and produces an action to execute [1]. During training, the agent interacts with the environment and receives a reward signal for each action it executes. The dataset, consisting of a state, an action, and a reward, is dynamically constructed and then used to optimize the model weights. This process repeats for every training step.

One of the major problems with this type of algorithm is the sparsity of the reward [2]. Ideally, the agent would get the reward for every action. In most problems, this is not possible. The reward can be calculated only at the end of the episode. The reward is then determined by whether the episode was successful. This poses a significant challenge when determining how each individual action contributed to the episode’s success. The problem is exponentially more complex in environments that require multiple sub-tasks [3].

The solution presented in this paper uses the MiniGrid (v3.0.0) environment [4]. MiniGrid is a lightweight, grid-based environment designed to be easily extensible. It allows the creation of problems (maps) with varying levels of complexity. It was originally introduced as part of the OpenAI Gym ecosystem, but it is now maintained as a



Academic Editors: Alexander Gegov
and Andrea Asperti

Received: 10 May 2026

Revised: 29 June 2026

Accepted: 17 July 2026

Published: 19 July 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

Gymnasium-compatible implementation. The state of the environment can be represented as a parameter matrix or an RGB bitmap. For the solution presented in this paper, the parameter matrix was used. Each value in the matrix represents a square in the grid and contains a tuple of the object's code, color, and state. As shown in Tables 1 and 2, the states are dependent on the type of object. For example, a door's state can be open, closed, or locked.

Table 1. Object codes in the MiniGrid environment.

Object	Code
unseen	0
empty	1
wall	2
floor	3
door	4
key	5
ball	6
box	7
goal	8
lava	9
agent	10

Table 2. Color codes in the MiniGrid environment.

Color	Code
red	0
green	1
blue	2
purple	3
yellow	4
gray	5

While solving complex multi-step problems is a considerable challenge, RL algorithms can effectively solve simpler problems that do not require long-term planning. The solution proposed in this paper shows that complex problems can be solved by decomposing them into simpler sub-tasks and using a combination of smaller agents, each trained to solve a single task, to address those sub-tasks. Single-task agents can be successfully trained using an RL algorithm. They are then combined into a single agent capable of solving any sub-tasks using a mixture-of-experts (MoE) strategy. In the end, a large language model (LLM) is used for high-level planning by decomposing the problem into a set of simpler sub-tasks.

Mixture of experts (MoE) is a modular learning paradigm in which multiple specialized subnetworks, referred to as experts, are combined via a gating mechanism that dynamically selects or weights them for each input [5]. The gating network can perform soft (weighted combinations) or hard (sparse expert selection) routing. The conditional computation property of the MoE is particularly attractive for large-scale models and complex tasks, where it can improve performance, generalization, and interpretability. The MoE architecture shows strong performance across diverse domains [6,7]. In the solution presented in this paper, hard routing was used. Each expert can solve a single task. Therefore, the gating network must select the correct expert, and only the selected expert's output is used as the model output.

LLMs [8] have emerged as a new paradigm in modern artificial intelligence, demonstrating great capabilities across a wide range of tasks. For example, they can perform

natural language processing, text generation, question answering, summarization, and reasoning. They are built primarily on transformer-based architectures. LLMs are initially trained on large amounts of text from a wide range of sources to build a statistical model that predicts the next word in a given text. They can then be fine-tuned for a specific task [9]. The progress of LLMs raises important questions and new challenges for interpretability, efficiency, and robustness, and poses ethical questions, prompting further research into more controlled, efficient, and reliable approaches to language modeling [10].

Sparse reward signals remain a fundamental obstacle in RL, especially when long-term planning and successful execution of various small interdependent tasks is required, before any feedback on success is received [1]. Addressing this challenge is critical to scaling RL methods to more complex problems that better reflect real-world decision-making scenarios. This paper explores one possible solution to avoid sparsity by dividing a complex task into simpler ones that can be learned individually. By coordinating agents through an MoE architecture and delegating reasoning tasks to an LLM, it achieves reasonable success in complex tasks that proved impossible for a single model of similar complexity. This demonstrates the benefits of modularity and specialization in learning systems [6,7]. This model is evaluated in the MiniGrid environment, a widely used benchmark for studying hierarchical and long-term planning in RL. This project shows that dividing a single complex RL agent into multiple simpler agents can significantly improve the ability to solve multi-step dependent tasks compared to end-to-end RL agents, providing an important direction for future research on scalable, robust RL algorithms.

In previous work [11], it was already shown that decomposing a complex problem into simpler ones and solving them can yield comparable results. Articles such as [12–14] demonstrate that LLMs have strong potential for reasoning and decomposing complex problems.

The solution presented in paper [12] focuses primarily on comparing different LLM models, and concludes that “llama-2.3” fine-tuned for Korean achieves the best results due to language-specific characteristics. In the said paper, reinforcement learning was simulated using LLM prompting. The LLM was given a detailed explanation of the scene, including the agent’s precise position, the goal, and the current reward, as well as prior learning experiences. The LLM was expected to provide simple, predefined movement instructions, which the execution module would then map to environment commands. The execution module is just a simple mapping from the textual commands to the environment inputs. The learning process is incorporated into the prompt generation. The prompt itself contains the acquired knowledge. This solution is an interesting exploration of an LLM’s capabilities, but it forgoes usability. Reinforcement learning is not leveraged in the solution. The LLM is used at every step of the episode, introducing significant computational overhead compared to a simple RL model. The acquired knowledge is severely limited by the LLM’s context window capacity. Additionally, the LLM is overburdened with very granular outputs and very precise geometric scene descriptions. The solution presented in this paper combines the LLM with an RL agent. This allows for the separation of responsibilities. The LLM is prompted only once at the start of the episode and tasked only with providing a high-level plan, while the RL agent handles pathfinding, obstacle avoidance, etc. Thus, it reduces computational cost while leveraging the capabilities of the RL algorithm.

Paper [13] proposes an RL model that incorporates an LLM into the training process. An LLM is used to define sub-goals that guide the RL agent during training. The environment is augmented to provide additional reward for achieving sub-goals. The paper identifies the high computational cost of prompting the LLM and proposes a solution by implementing a learned surrogate model to approximate its responses. In addition, the LLM is prompted only once at the beginning of an episode. This approach results in a monolithic RL agent expected to learn the entire solution, including the planning

for that solution. This limits the RL agent's headroom and significantly complicates and slows down the training process. The solution presented in this paper leverages the LLM's planning capabilities during inference, where the impact of computational cost is much less pronounced. This separates responsibilities and does not require augmenting the RL algorithm, thus leveraging the strengths of both technologies.

In article [15], the expert agents were already trained, and based on the performance achieved with a mixture-of-experts model [5–7], an experiment combining an LLM with an MoE model was proposed. This integration aims to leverage the LLM's high-level planning capabilities while preserving the specialization and efficiency of expert policies. By aligning natural language task decomposition with learned expert behaviors, the goal is to improve both sample efficiency and generalization across diverse environments.

Sparse reward signals represent a fundamental bottleneck in scaling reinforcement learning to real-world tasks. In many practical domains, agents must execute long sequences of interdependent actions before receiving any feedback. A single monolithic RL agent trained under such conditions frequently fails to converge, as the credit assignment problem becomes intractable across many steps. While prior work [12,13] has explored curriculum learning and LLM-guided reward shaping as partial remedies, these approaches either retain a monolithic policy structure or introduce prohibitive computational overhead by involving the LLM at every decision step.

The proposed architecture differs from prior hierarchical RL methods in three concrete ways. First, the LLM planner is invoked exclusively at inference time and does not participate in training. Thus, the training pipeline remains a standard RL procedure. Second, each expert is a fully independent PPO policy trained on a single sub-task in isolation, requiring no joint optimization or options-style intra-policy communication. Third, the gating network is trained in a supervised, rather than end-to-end reinforcement, manner. Together, these choices prioritize modularity and training simplicity over learned compositionality.

Prior to this architecture, established hierarchical reinforcement learning methods were explored as candidate solutions for the MiniGrid environment. These approaches did not yield a working solution within the constraints of this study. This difficulty motivated the present work, which focuses on establishing a functionally effective modular baseline rather than an exhaustive comparison with hierarchical methods. A systematic comparison against hierarchical RL baselines is therefore identified as the primary objective of future work, building on the foundation established here.

The central motivation of this work is therefore twofold. First, to demonstrate that decomposing a complex, sparse-reward task into a set of individually trainable, dense-reward sub-tasks, each solved by a dedicated expert policy, substantially alleviates the learning difficulty. Second, to show that an LLM, invoked only once per episode for high-level planning, can effectively bridge the gap between natural language task specification and low-level expert execution without imposing high runtime cost. Together, these design choices provide a modular, computationally efficient, and extensible architecture, whose advantages over a monolithic PPO baseline are empirically validated in the MiniGrid environment.

2. Related Work

RL defines decision-making as an interaction between an agent and its environment, while behavior is optimized through trial-and-error feedback [1]. Although deep RL has achieved impressive results, such as in arcade games and control tasks [16,17], its performance significantly deteriorates in tasks with sparse or delayed rewards. In environments that require a long sequence of correct actions before a reward is obtained, RL agents often struggle to find a solution, leading to slow training or complete stagnation. These limita-

tions pushed the research toward decomposing the learning goal into multiple sub-goals and dividing the monolithic policy into an array of experts.

One possible solution to this problem is curriculum learning. It improves learning efficiency by guiding the agent through the learning process, first decomposing the solution into sub-tasks and then ordering them from easier to harder, allowing the agent to gradually acquire increasingly complex behaviors [18]. These sub-tasks are usually handcrafted by the operator during the training setup. There have been attempts to automatically generate sub-tasks during the learning process [19]. Although this method can at least partially mitigate the effect of sparse rewards by introducing intermediate learning signals, it typically relies on a single policy and does not explicitly decompose decision-making into specialized components.

The mixture-of-experts architecture was introduced to split a single large neural network into an array of simpler networks. Each network is an expert in one of the sub-tasks of the complex task that the large network was trying to solve [6]. During the inference phase, the experts' outputs are routed through the gating network. The gating network can execute either soft or hard gating. In soft gating, the experts' outputs are weighted, and the weighted sum is used as the model output. On the other hand, hard gating is a selection process. Only a single expert is selected, and its output is forwarded, while all others are discarded. MoEs have shown significant efficiency in the scaling of neural networks [7,20]. This paper demonstrates the advantage of the MoE architecture over monolithic agents by using specialized experts, each trained to solve a simple task, which can serve as building blocks for more complex solutions. Thus, it eliminates the problem of sparse or delayed rewards.

With the introduction of LLMs, the logical next step was to incorporate them into RL agents. Since we as humans use natural language to plan our actions, this was the obvious place to insert an LLM [12–14,21]. The great capacity for learning and abstract reasoning in LLMs has inspired their application beyond the initial processing of natural language. The transformer architecture, used in LLMs, has proven effective for RL problems that require sequence modeling [22,23]. Particularly interesting is the attempt to use only an LLM for RL by designing prompts that instruct it to perform an RL task entirely in natural language [24]. The knowledge learned during training is retained in language form within the LLM's context window. This solution, while interesting, is prohibitively slow and computationally expensive.

The solution presented in this paper leverages an LLM to perform high-level reasoning and solution decomposition. Compared to similar solutions [12,13], our methodology removes the LLM from the training process. Instead, the LLM is used only in the inference phase. The experts are trained as pure RL models, enabling faster, more efficient training. In addition, the complexity of sub-tasks is allowed to rise organically; to that extent, the agent can learn. For example, tasks such as opening doors and removing obstacles along the route are ones the RL agent can easily learn. There is no need to strain the LLM further by designing those actions as potential sub-tasks.

3. Methodology

3.1. Environment

The solution presented in this paper uses the MiniGrid environment. It was extended to provide greater control over mission complexity. It had to follow predefined rules for object placement to avoid unsolvable scenes. The environment output (state) was extended with the textual scene description, which was later used in the LLM prompt generation. Mission complexity was primarily determined by the number of rooms and the number of objects in each room. The grid size was fixed at 11×11 , limiting the maximum number

of rooms to 4. Except for these extensions, the default environment dynamics and reward structure were preserved. The only modification was the use of a dedicated “done” action to signal the completion of a sub-task, rather than the termination being triggered by satisfaction of the global objective. An example of the execution environment is shown in Figure 1.

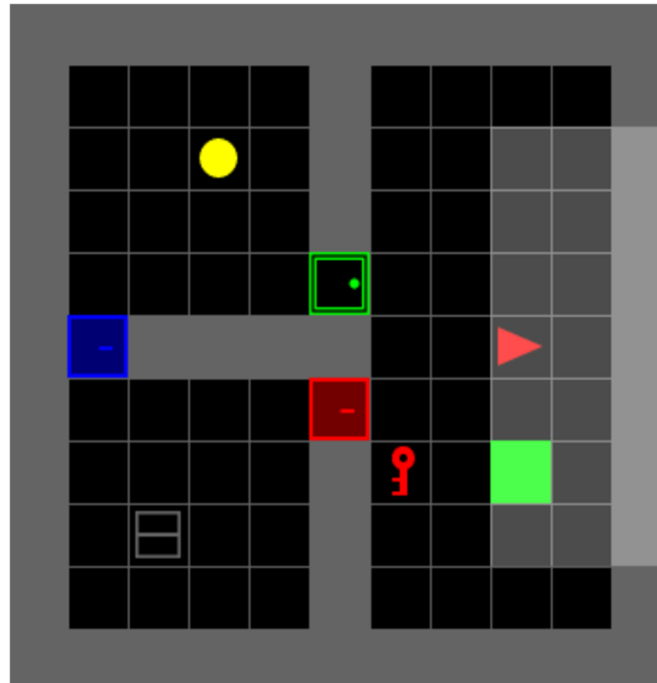


Figure 1. An example of the custom MiniGrid environment. Objects are color-coded. Objects with the same color relate to one another. The red key unlocks the red door, and the gray box can contain a gray key. The green square is the goal. The red triangle is the player. The area shaded in light gray is the player’s field of view.

The number of objects in the environment was configurable. Object placement constraints were designed to prevent obstructing the agent, the target location (green tile), or doorways. In addition, the keys were positioned so that the agent could always find a valid path to the goal.

The doors could be configured to remain either open or unlocked throughout the training process. Each door was assigned a specific color and, if locked, could be unlocked only with the same color key. The keys could be placed inside boxes, requiring the agent to locate and open the box before picking up the key. Preliminary experiments showed that the RL model, used by the experts, was unable to learn the full key–door interaction mechanism although it successfully learned to pass through doors when they were already unlocked. Consequently, all inter-room doors were kept unlocked during the training. This design choice later led to generalization issues. The agent exhibited confusion when encountering locked doors during inference. To resolve this problem, additional doors were introduced at random locations within rooms (rather than between rooms), exposing the agent to locked-door observations without requiring them for task completion. Finally, the benchmark model (RL without LLM) was trained with the environment in its original form.

To enable the LLM to reason about the mission defined by the environment, a structured textual description of the map was required. This representation was generated directly during the environment’s scene construction process; an example is shown in Table 3. The objective of the description was to provide all the information necessary to develop a strategy for completing the assigned mission. Accordingly, the representation

comprised the number of rooms, the status of the inter-room doors, the objects present in each room, and the overall task specification. To avoid overloading the LLM prompt with unnecessary data, the description was intentionally limited to essential details. This resulted in improved LLM performance.

Table 3. An example of the textual representation of the environment.

The scene contains:
Three rooms. Upper left, lower left and right.
There is an unlocked red door between the upper left and lower left rooms.
There is a locked yellow door between the upper left and right rooms.
There is a locked gray door between the lower left and right rooms.
Upper left room contains:
-goal
Lower left room contains:
Right room contains:
-robot
-gray box
-yellow box
-yellow ball
Mission: go to goal

3.2. Proximal Policy Optimization

Proximal policy optimization (PPO) [17] was used as a reinforcement learning algorithm for the experts. Proximal policy optimization was proposed as an extension of the trust region policy optimization (TRPO) algorithm [25]. The primary motivation was to reduce the computational overhead. Both approaches rely on the trust region method to constrain the divergence between successive policy updates. This divergence is typically measured using the KL divergence.

In contrast to TRPO, which requires explicit computation of the Hessian matrix and is therefore computationally intensive, PPO enforces this constraint through a clipping mechanism that provides a practical approximation. Both algorithms belong to the class of policy gradient methods, in which an estimate of the policy gradient is computed and used within a standard gradient ascent optimization framework. In general, the policy gradient estimator is in the form:

$$\hat{g} = \hat{E}_t [\nabla_{\theta} \log \pi_{\theta}(a_t | s_t) \hat{A}_t]. \quad (1)$$

where $\hat{E}_t$ is the average expectation over a finite batch of samples; π_{θ} represents the current policy parameterized by θ ; a_t is the action taken at time-step t ; s_t is the environment state at time-step t ; and $\hat{A}_t$ is the value of the advantage function at time t . Policy gradient estimates may have very large magnitudes, leading to unstable or excessively large updates to the policy parameters. Trust region techniques are therefore employed to constrain the update step. PPO achieves this constraint through a simplified clipping mechanism defined as follows:

$$L(\theta) = \hat{E}_t \left[\min \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}, \text{clip} \left(\frac{\pi_{\theta}(a_t | s_t)}{\pi_{\theta_{old}}(a_t | s_t)}, 1 - \epsilon, 1 + \epsilon \right) \right) \hat{A}_t \right]. \quad (2)$$

where ϵ is the clipping factor, a small predefined constant.

For the implementation of the algorithm, the “stable baselines 3” (v2.7.0) [26] framework was used. A custom neural network was designed to support the specific environment-state format.

3.2.1. Policy

The custom neural architecture employed in the experiments operated on three distinct inputs at the feature extraction stage, corresponding to the environment outputs summarized in Table 4. The directional input encoded the agent's orientation, the image input consisted of a matrix representation of the agent's local observation window, and the mission input was a vector derived from the tokenized textual description of the active goal. The image input consisted of a matrix representation of the agent's local observation window. Specifically, the default MiniGrid partially observable 7×7 egocentric view (array (3, 7, 7), uint8), centered on the agent and aligned with the agent's current direction. This partial observability means the agent has no access to global state information. It cannot directly observe objects outside its immediate field of view, which is particularly relevant for multi-room configurations where keys or doors may lie in unvisited rooms. Originally, the agent was trained in a fully observable environment. This proved to be extremely detrimental to learning performance due to very small differences between consecutive states.

Table 4. Single environment state.

Value	Type
direction	uint8
image	array (3, 7, 7), uint8
mission	array (32), int64

Feature extraction was performed using three separate subnetworks, each dedicated to processing one input modality. The resulting latent vectors were concatenated and fed into both the policy and value function heads. An overview of the network design is provided in Figure 2. The network produced two outputs: a policy output, representing a probability distribution over the environment's action space, and a value output, corresponding to the estimated value of the current state. Additional model details are shown in Appendix A.3.

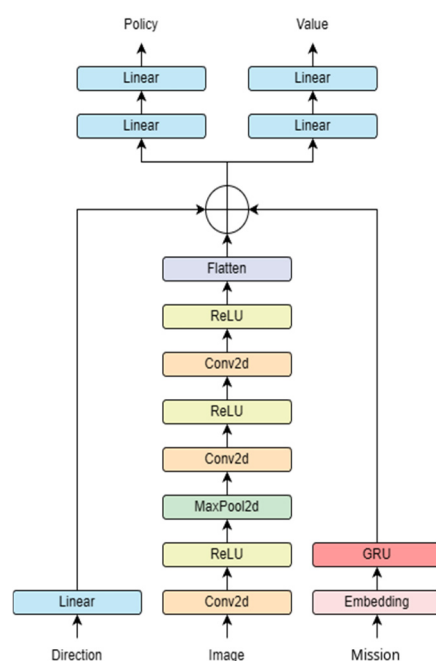


Figure 2. The neural network architecture.

3.2.2. Hyperparameter Search

Manual hyperparameter tuning can be impractical, especially in reinforcement learning, where small changes in hyperparameter values can lead to drastically different learning performance. Evolutionary population-based techniques are often used to optimize parameters. In the solution presented in this paper, the “Hydra” (v1.3.2) population-based hyperparameter sweeper [27] was used, with proposed default values [28] as an initial setup. Hyperparameter optimization was performed using the PB2 algorithm, which leverages a Bayesian surrogate model to select effective mutations for the highest-performing agents within the population. Compared to conventional population-based training, this approach offers improved sample efficiency and enables the use of smaller populations. The parameter search intervals used by the sweeper are shown in Appendix A.1.

During experimentation, training instability was observed in later phases due to an excessive number of update steps. To address this issue, the training procedure was segmented into multiple consecutive phases, with the number of epochs per update reduced. The configurations used at each training phase are shown in Appendix A.2.

3.3. Mixture of Experts

Mixture of experts is an ensemble learning technique [6]. In general, the neural network is partitioned into experts, which the gating network aggregates into a single output. In this setup, the entire network is optimized in a single training process. During training, the gating network focuses on a particular expert for a given task or task family. This creates a soft gating mechanism, in which a weighted linear combination of expert outputs is presented as the network’s output, usually in the following form:

$$f(x) = \sum_i \omega_i(x) f_i(x) \quad (3)$$

where ω_i is the weight of the i -th expert, and f_i is the output of the expert. In the solution presented in this paper, a hard gating mechanism was used, in which a single expert is selected by the gating network, and the expert’s output is propagated as the network’s output. This was achieved by representing the weights in Equation (3) as one-hot vectors. Given that the experts were pretrained on a predefined set of problems, the logical conclusion was to let a particular expert alone control the agent to solve the problem for which that expert was trained.

The gating network was trained in a supervised manner to output a one-hot vector selecting the predefined expert, while the experts were trained using the PPO algorithm to solve predefined problems. A deterministic string-matching rule would be functionally equivalent to the implemented gating network in the current fixed vocabulary setting. However, a trainable network was preferred because it operates on continuous token embeddings rather than exact string equality. Thus, it provides robustness to minor phrasing variations in LLM output and allows future extension to soft gating without architectural modification. Each of those problems was designed as a possible sub-goal for the LLM planning stage. Given this, it was necessary to design sub-goals to maximize the LLM’s reasoning capability while leveraging the PPO algorithm’s learning capacity. Additionally, every environment mission had to be solvable by decomposition into only those sub-goals. Through experimentation, the following sub-goals were identified as appropriate: “go to goal”, “go to <object>”, “toggle <object>”, “pick up <object>”.

Each expert is trained independently using the PPO algorithm described in Section 3.2. During expert training, the chosen sub-task (e.g., “pick up yellow key”) is treated as the complete episode mission. The reward formula in Equation (4) applies directly at the sub-task level, providing a dense-enough signal compared to the original sparse global

mission. Sub-task completion is detected implicitly. The expert signals completion by executing the dedicated “done” action, at which point control is returned to the gating network for the next sub-goal assignment. No external completion oracle is required.

3.4. Large Language Model

An LLM was employed for the planning stage. As described, the environment output was augmented to produce a textual representation of the scene. For the first step of the episode, this textual representation, along with the global mission, was fed into the LLM to generate a step-by-step plan consisting of predefined sub-goals. After this first step, the LLM was not engaged again until the next episode. These sub-goals are then given to the MoE agent, one at a time, as the current mission. This mission was then executed until the agent performed the “done” action, signaling that the mission was over. Table 5 shows the expected output from the LLM for the input given in Table 3.

Table 5. Response provided by the LLM.

1. toggle yellow box
2. pick up yellow key
3. toggle yellow door
4. go to goal

The LLM used in this project was “qwen3:30b”, which demonstrated the strongest performance. Model evaluation followed a straightforward procedure. The LLM was prompted using a fixed set of predefined scenes paired with the user-specified solutions. The LLM output was assessed against the expected responses. Additionally, the parameters used to run the LLM, along with the system prompt and a limited LLM benchmark are provided in Appendix B. The LLM evaluation is limited to manual verification of outputs on 100 episodes. Prompt sensitivity, the impact of alternative phrasings, hallucination failure modes, and model size effects are not analyzed and represent important avenues for future investigation.

As a benchmark, an end-to-end agent was trained using the PPO algorithm alone, without incorporating an LLM-based planning component.

4. Experimental Results

The PPO models for the experts were trained first. As stated earlier, the training was divided into multiple consecutive steps and continued until satisfactory results were obtained. We observed different numbers of required runs for different problems. The easiest proved to be “go to goal”, achieving maximum possible performance after only two runs. The training timeline is shown in Figure 3, with consecutive runs presented in different colors.

Figure 4 shows the same data for the “pick up” expert. It can be observed that the training took considerably longer. Five training runs were needed to finish the training. After the first two, the subsequent runs showed diminishing returns. Training was stopped after the fifth run as the results showed little improvement. It can also be observed that the average reward achieved after the training was considerably lower than that of the “go to goal” expert. Consecutive runs were executed with fewer epochs and a lower learning rate, as presented in Appendix A.2 and achieved stable, gradual improvement throughout training. In the first couple of steps of the later runs for the “pick up” expert, we observe a small drop in performance, but the run ultimately finished with a higher score than the previous one.

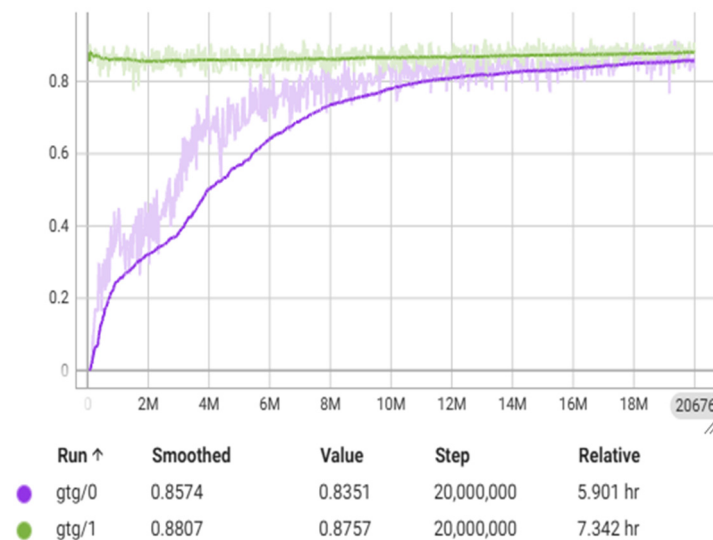


Figure 3. The training timeline for the “go to goal” expert. The y-axis shows the average reward received in the hundred episodes, and the x-axis shows the training step. The graph shows training performance converging to a stable value. Further training would not improve performance.

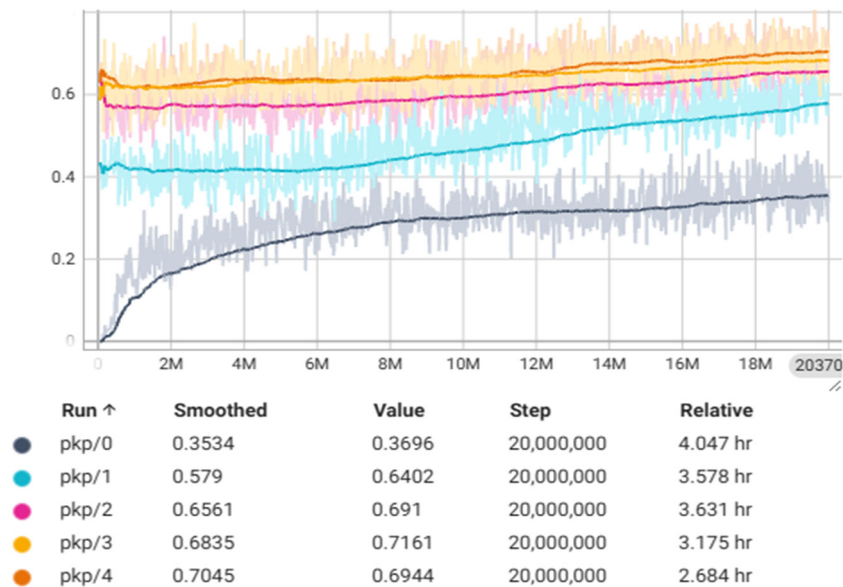


Figure 4. The training timeline for the “pick up” expert. The y-axis shows the average reward received in the hundred episodes, and the x-axis shows the training step. The graph shows training performance converging to a stable value. Further training would not improve performance.

Reinforcement learning is known to produce different results with different random seed values. To assess training stability, all four experts were trained across four randomly chosen seed values. The results can be seen in Figure 5. The highest-performing expert shows very tight convergence across all four seeds. The final smoothed value spread was less than 0.015. This indicates that training stability for simpler, well-defined sub-tasks is largely seed independent. The remaining three experts exhibit greater variance in final performance levels. This is consistent with the known sensitivity of reinforcement learning to initialization conditions under more challenging learning problems. Importantly, all four seeds across all four experts exhibit the same qualitative learning trajectory, monotonic improvement and eventual convergence. No seed failed to learn or diverge. The observed variance is therefore in the final performance level rather than in the fundamental stability of the training process.

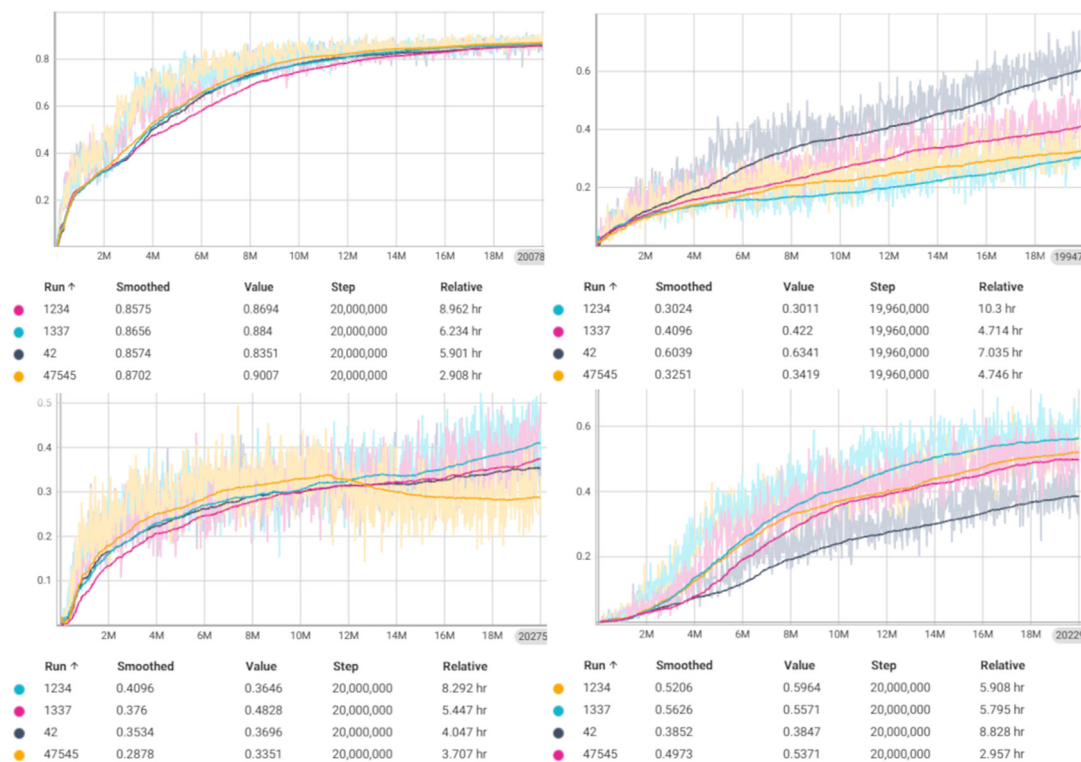


Figure 5. The training timeline for the different random seed values. The top left shows the “go to goal” expert, the top right shows the “go to object” expert, the bottom left shows the “pick up” expert, and the bottom right shows the “toggle” expert. The y-axis shows the average reward received over one hundred episodes, and the x-axis shows the training step. The graph shows the initial training run of each expert with different seed values. Runs are named by the value of the random seed.

The benchmark model used to compare results from introducing the LLM into the planning stage was also trained with the PPO algorithm. It was trained to solve the global environment mission, including planning. Figure 6 shows the training timeline. It shows improvement during training but ultimately plateaus at a relatively low level. It can be observed that subsequent runs show no improvement. This behavior was expected, since not all missions require long-term planning. Ultimately, the agent cannot solve more complex problems that require multiple steps.

The results achieved by different experts during the training are shown in Table 6. The experts were tested on all of the problems. As expected, they achieved zero reward on problems they were not trained on. The “toggle” and “pick up” experts show considerably lower performance than the other two. Consequently, this reduces the agent’s ultimate performance. When used in combination with the LLM, the agent must solve every sub-task the LLM assigns to successfully complete the global mission. The agent can fail any of those sub-tasks, resulting in episode failure.

Table 6. The average reward achieved by experts during the training was measured across all problems.

Model	Go to Goal	Go to Object	Pick Up	Toggle
Go to goal	0.86	0.0	0.0	0.0
Go to object	0.0	0.72	0.0	0.0
Pick up	0.0	0.0	0.57	0.0
Toggle	0.0	0.0	0.0	0.47

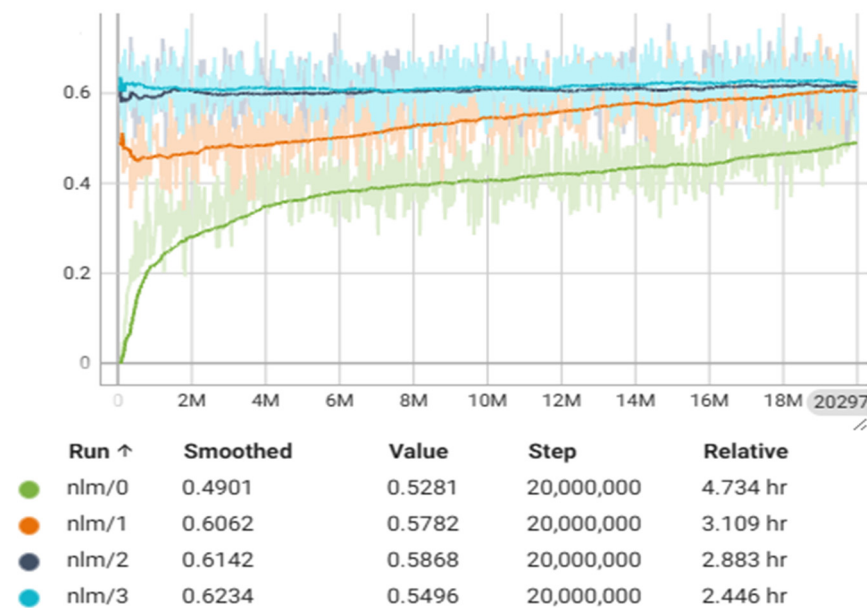


Figure 6. The training timeline for the “no language model” agent. The y-axis shows the average reward received over one hundred episodes, and the x-axis shows the training step. The graph shows training performance converging to a stable value. Further training would not improve performance.

The results are shown in Table 7. The “PPO” model is just PPO, without an LLM, trained on the single-step problems. As expected, this model performs the worst on complex, multi-step problems. “LLM + PPO” is the same model but paired with an LLM. A significant uplift in the performance can be observed. In this setup, the PPO model handles only the simple tasks on which it was trained, while the LLM decomposes the complex problem into those simple tasks. “LLM + Mixture of Experts” shows the best results. This was the expected outcome because the experts were trained on a single, simple task. This enabled better performance compared to a model trained on multiple tasks simultaneously. “No Language Model” was used as a benchmark. This model was the only one trained on complex problems. It is implemented as an end-to-end PPO model, trained to solve multi-step problems. Additionally, Table 7 shows the average reward achieved as a function of the number of sub-tasks. The number of sub-tasks was determined by the LLM decomposition of the global mission. As expected, all the models have similar results on a single sub-task. With a higher number of sub-tasks, a significant reduction in achieved reward can be observed, with models paired with an LLM leading by a wide margin.

Table 7. The average reward achieved during testing. Rewards shown for different complexity brackets are calculated on a sample of 100 episodes.

Model	Average Reward (1000 Episodes)	Average Reward (100 Episodes)	1 Sub-Task	3 Sub-Tasks	4 Sub-Tasks
PPO	0.33	0.35	0.63	0.0	0.0
LLM + PPO	0.55	0.58	0.81	0.15	0.09
LLM + Mixture of Experts	0.57	0.60	0.89	0.22	0.1
No Language Model *	0.43	0.39	0.63	0.04	0.06

* The model was the only one trained using PPO on complex problems. All others encountered only single-step problems during the training.

The first 100 episodes were recorded and manually verified. Table 8 shows the percentage of episodes that models successfully completed across different complexity levels. Table 9 shows individual rewards for the first 10 runs. We observe comparable results for episodes with relatively simple objectives. The episodes with higher-complexity objectives

show significantly better performance by the models paired with the LLM. The results for episodes with 4 or more sub-tasks are relatively less informative due to the small number of episodes in the recorded sample. As a result, episodes with more than 4 sub-tasks were omitted from the results. Of the 100 manually recorded episodes, approximately 64 involved a single sub-task, 21 involved three sub-tasks, and 14 involved four sub-tasks. The small number of four-sub-task episodes limits the statistical reliability of the results in that bracket, and those figures should be interpreted as indicative rather than conclusive.

Table 8. Percent of the episodes successfully completed based on a sample of 100 episodes.

Model	1 Sub-Task (64 Episodes)	3 Sub-Tasks (21 Episodes)	4 Sub-Tasks (14 Episodes)
PPO	68.75%	0.00%	0.00%
LLM + PPO	89.06%	23.81%	14.29%
LLM + Mixture of Experts	98.44%	28.57%	14.29%
No Language Model	68.75%	4.76%	7.14%

Table 9. The reward received by both models during the first 10 episodes.

Number of Sub-Tasks	PPO	LLM + PPO	LLM + Mixture of Experts	No Language Model
3	0	0.7422334	0.74710745	0
1	0.9486359	0.9579439	0.9469437	0.9553719163
4	0	0.7147534	0.7237845	0
4	0	0	0	0
1	0.7127631	0.7543476	0.7157412	0.7247933745
1	0.7322314	0.7423315	0.7214331	0.8289256096
3	0	0.56623106	0	0
3	0	0	0.87412578	0.8661156893
3	0	0	0	0
1	0.9436017	0.9528192	0.9518792	0.9553719163

The environments in which the LLM decomposed the solution into one sub-task were solvable by all agent configurations. These environments provided a clear path for the agent to go to the goal. No additional tasks were required. The goal tile was either in the same room or had unlocked doors along the path. The environments with three sub-tasks had a locked door on the path to the goal. The agent had to pick up the key, unlock the door, and then go to the goal. In this setup, “LLM + PPO” and “LLM + MoE” show, as expected, comparable results. This similarity propagates to the aggregate results shown in Table 7. One interesting anomaly can be observed. The “No Language Model” successfully solved one environment with three sub-tasks. This is the result of an unexpected emerging behavior. The model learned that it takes fewer actions to pick up an object on the path than to go around it. In this specific example, the object that was on the path was the key needed to unlock the door to the room where the goal tile was. The environments with four sub-tasks had the key inside the box. The example is shown in the provided video.

In the first 100 recorded episodes, LLM outputs were manually verified. No error was observed. This result has an important implication for the interpretation of the experimental results: since the planning component introduced no failures, all observed performance differences between model configurations in Table 8 are attributable entirely to the execution module. Consequently, the comparison between LLM + PPO and LLM + MoE in Table 8 constitutes a direct measure of the benefit of expert specialization, with the planner effectively held constant as a zero error component across both conditions. Not all outputs were optimal, but all were valid solutions to the global mission. Extrapolating from this, it

can be assumed that most of the observed failures were due to the execution model failing to execute one of the tasks. Since every task has some chance of failure, missions with more sub-tasks are expected to have a higher chance of failure. Given the expert success rates in Table 6, a rough upper bound on three-sub-task success can be estimated as the product of the relevant expert rates. For a pick-up + toggle + go-to-goal sequence this yields approximately $0.57 \times 0.47 \times 0.86 \approx 0.23$, which is relatively close to the observed 28.57% success rate, indicating that execution failure rather than planning failure is the dominant bottleneck.

The gating network was not tested individually. It was trained to overfit to a small set of inputs and consequently has 100% accuracy. It was left as a network due to performance reasons.

The reward is calculated using the following formula:

$$R = 1 - 0.9 \left(\frac{\text{num_steps}}{\text{max_steps}} \right) \quad (4)$$

Consequently, a score of 1.0 is practically impossible. Even the optimal solution will have a reduced reward, depending on the number of steps taken. The reward was awarded to the agent only upon successful completion of the global mission. This was true regardless of the number and success of the sub-goals.

The absolute reward values are bounded below the theoretical maximum by the step-count penalty in Equation (4) and by the individual expert performance ceilings reported in Table 6. However, the results show the relative improvement of the proposed architecture over the flat baseline model on multi-step missions, thus providing a method for potentially solving any MiniGrid scenario.

The source code for the implemented solution can be found at the address provided in the Supplementary Materials section. In addition, a video demonstrating the proposed solution is available at the YouTube link in the Supplementary Materials.

5. Discussion

The results presented in this paper show that the modular architecture offers a promising direction for sparse reward and long-horizon problems in RL. By explicitly separating planning and execution, the proposed solution enables specialized learning and reduces interference, a known problem in monolithic RL agents. However, the proposed solution raises new questions, such as how to define experts, how to connect an LLM with experts, and how to input the environment state into an LLM without an explicit textual representation.

The primary goal of this study was to compare the proposed framework against a widely adopted flat RL baseline, PPO. In order to isolate the benefits of introducing modular planning and execution components within the same general training paradigm, PPO was selected due to its broad applicability and strong performance. However, we acknowledge that other hierarchical reinforcement learning approaches, including options-based frameworks, feudal-style architectures, and subgoal-driven strategies, represent an important family of methods that share conceptual foundations with the proposed architecture. It should be noted that hierarchical reinforcement learning methods were explored as candidate solutions during the development of this work. These approaches did not produce a functioning agent in the MiniGrid environment under the sparse reward conditions used in this study, which was a key motivation for the architecture presented here. The primary goal of this paper is therefore to establish a working modular baseline that solves multi-step MiniGrid tasks. Future work will use this baseline as the foundation for a systematic comparison against established hierarchical reinforcement learning approaches.

In the current implementation, the experts are defined manually, which simplifies the analysis but may limit generalization and transferability to other environments. Future

work could include experimentation with a soft-gating MoE model. This could allow for self-defined experts.

In the current solution, the LLM prompt is hard-coded. This could also be learned [24], potentially improving planning performance. Additionally, the sub-tasks (experts) are predefined. This could be improved by using a small LLM for textual input to the agent network, rather than a simple GRU unit. This would improve the execution module's task comprehension, in turn making the connection to the planning module simpler.

Many LLM models are multi-modal. They can work with the image as input. Using this ability would render it unnecessary to modify the environment to output a textual representation of the state. In our experiments, models small enough to be self-hosted performed poorly at understanding the image input. However, using one of the more robust, commercially available models (such as GPT) could solve this.

Current experiments are limited to the MiniGrid environment, which is suitable for long-term planning studies but remains very simple. Further work should evaluate the scalability of this solution across different environments and with more complex control mechanisms. This approach could prove especially beneficial in transferring the learned behavior from a simulator to a hardware robot. One possible extension of this approach is its application in swarm robotics [29]. The experiments were conducted with a drone simulator but had not produced conclusive results as of the writing of this article. The addition of multiple drones is an interesting avenue for further research, especially the swarm orchestration module that connects LLM global planning to single-drone execution.

In this study, an LLM was used as a planning module. An interesting avenue for further research would be to introduce an oracle planning module. A symbolic planner with guaranteed accuracy could be used. This would allow for an investigation into the benefits and compromises of using a more flexible but inherently not 100% accurate planner. A formal ablation comparing the LLM planner against a deterministic rule-based planner is not pursued in this work, as the zero planning errors observed across 100 episodes indicate that both approaches would yield equivalent results within the current experimental scope. The more informative avenue for future ablation work is therefore a broader evaluation of planning reliability under more varied and complex task configurations, where LLM errors are more likely to emerge.

Future research in hierarchical models should improve the adaptability, scalability, and efficiency of learning, while preserving modularity. Advances in the automatic discovery of sub-tasks and integration with an LLM should address this approach's shortcomings. As proved in this paper, hierarchical models have the potential to significantly improve the performance of intelligent agents in complex environments with sparse rewards.

6. Conclusions

This paper presents research on the challenges posed by sparse rewards in RL and proposes a solution via a hierarchical architecture. This solution effectively addresses the problem by training RL agents in a dense-reward environment and combining them with an LLM to address sparsity.

Another benefit of this approach is the agent's modularity. New LLM models are coming out every day. Improving the planning module would be as easy as modifying a single configuration parameter, enabling the solution of even more complex tasks. Given the versatility of LLMs, we believe most problems in the MiniGrid environment can be solved with an agent like this. Additionally, transferring the agent to a different environment should require only a relatively trivial amount of training on the basic skills in that environment.

Compared to similar works [12,13], potential problems were identified. The solution in the paper [12] forgoes the RL agent entirely, allowing the LLM to control the agent directly. This puts unnecessary strain on the LLM. RL models have repeatedly demonstrated the ability to solve complex environments. By leveraging that capability, the solution presented in this paper delivers better results, regardless of the language used. In the paper [13], a different solution was implemented. In this work, an LLM was used during the training of an RL agent. This opens up additional possibilities but at a significantly greater computational cost. In accordance with the previous work [11], the complexity of both the agent and the training process was reduced. This provides much faster training times and greater accessibility due to lower hardware requirements. Additionally, by employing the LLM during inference, the RL agent is not overburdened with planning tasks. This allows the strengths of both technologies to be leveraged, with each supplementing the other's weaknesses.

Supplementary Materials: The source code for the solution presented in this paper can be found at <https://github.com/Idokorro/MiniGrid-RL>, accessed 20 March 2026, and the video demonstrating the results is available at <https://www.youtube.com/watch?v=EfYVCSxA6Gw>, accessed 20 March 2026.

Author Contributions: Conceptualization, I.C. and A.P.; methodology, I.C. and A.P.; software, I.C. and A.P.; data curation, I.C. and A.P.; investigation, I.C. and A.P.; writing—original draft preparation, I.C. and A.P.; writing—review and editing, I.C., A.P., M.J., and J.V.; supervision, M.J. and J.V. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: A publicly available LLM was used in this study. This LLM can be found at <https://ollama.com/library/qwen3> (accessed on 17 December 2025), and the MiniGrid environment can be found at <https://minigrid.farama.org/index.html> (accessed on 17 December 2025).

Acknowledgments: During the preparation of this manuscript, the authors used “ChatGPT” (GPT-5.2) for the purposes of spelling and grammar correction. The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ML	Machine learning
AI	Artificial intelligence
RL	Reinforcement learning
MoE	Mixture of experts
LLM	Large language model
PPO	Proximal policy optimization
TRPO	Trust region policy optimization
PB2	Population-based bandits
GRU	Gated recurrent unit

Appendix A

Appendix A.1

Table A1. The ranges used for the Hydra parameter search.

Parameter	Type	Range
initial_learning_rate	uniform_float	0.000001–0.01
final_learning_rate	uniform_float	0.000001–0.01
batch_size	categorical	[32, 64, 128, 256]
gamma	uniform_float	0.8–0.9997
horizon	categorical	[256, 512, 1024, 2048]
n_epochs	uniform_float	4–10
gae_lambda	uniform_float	0.9–1.0
clip_range	categorical	[0.1, 0.2, 0.3]
clip_range_vf	uniform_float	0.0–1.0
normalize_advantage	categorical	[true, false]
ent_coef	uniform_float	0.0001–0.1
vf_coef	uniform_float	0.5–1.0
max_grad_norm	uniform_float	0.1–1.0

Appendix A.2

Each training run was 20 million steps long, and in each step, 16,384 samples were processed. The number of epochs for different training runs is shown in Table A2. All training runs used a fixed random seed of 42. Training was performed on a single MSI NVIDIA RTX 4070 Ti GPU (sourced in Belgrade, Serbia). Each 20-million-step training run required between 4 and 7 h of wall-clock time. Training used 16 parallel environment instances. The LLM was served locally via Ollama (v. 0.30.0) on the same machine. Inference time per episode was between 3 and 15 min, due to the thinking phase. Training and evaluation environments were strictly separated. Experts were trained on randomly generated scenes and evaluated on a freshly sampled set of 1000 episodes with no overlap. The evaluation distribution matched the training distribution in terms of map structure and object placement rules.

Table A2. Hyperparameter values in consecutive runs.

Model	Run	Nb. of Epochs	Initial Learning Rate	Final Learning Rate
Go to goal	1	6	0.001	0.00003
Go to goal	2	4	0.0003	0.000003
Pick up	1	9	0.001	0.00003
Pick up	2	9	0.0003	0.000003
Pick up	3	9	0.0003	0.000003
Pick up	4	9	0.0003	0.000003
Pick up	5	6	0.0002	0.000003
Go to object	1	7	0.001	0.00003
Go to object	2	7	0.0003	0.000003
Go to object	3	4	0.0003	0.000003
Toggle	1	4	0.001	0.00003
Toggle	2	4	0.0003	0.000003
Toggle	3	4	0.0003	0.000003
Toggle	4	4	0.0003	0.000003
No language model	1	6	0.001	0.00003
No language model	2	6	0.001	0.00003
No language model	3	4	0.0002	0.000003

Appendix A.3

The mixture-of-experts model consisted of 4 experts and a hard gating network. Each expert had an extractor and a policy network. The extractor would process the raw input data and provide an embedding vector as input for the policy network. The policy network would be based on the extractor's input, outputting a policy distribution for the input and an expected value (actor and critic policies).

The gating network consisted of 1 embedding layer with a 32-character input and a 1024-dimensional output, a gated recurrent unit with 128 neurons, and a linear layer with 4 neurons.

Each extractor network had 3 sub-modules for 3 inputs. Textual input consisted of 2 layers for processing the current mission. These layers were an embedding layer with a 32-character input, outputting an embedding vector with 1024 dimensions (32 per character), and a gated recurrent unit with 128 neurons. The direction input module consisted of 1 linear layer with 16 neurons. The structure of the image module is shown in Table A3.

The policy network had 2 outputs: policy and value. Both had 1 linear layer with 64 neurons.

Table A3. The image sub-module structure from the extractor network.

Layer	Layer Type	Num. Kernels	Strides
1	Conv2d	16	2,2
2	ReLU	/	/
3	MaxPool2d	/	2
4	Conv2d	32	2,2
5	ReLU	/	/
6	Conv2d	64	2,2
7	ReLU	/	/
8	Flatten	/	/

Appendix B

Table A4 shows configuration parameters used for running an LLM using Ollama software, and Table A5 shows the system prompt used by the LLM.

Table A4. LLM configuration.

Parameter	Value
model	qwen3:30b
arch	qwen3moe
quantization	Q4_K_M
num_ctx	40960
repeat_last_n	−1
repeat_penalty	2.0
temperature	0.0
top_k	20
top_p	0.95
min_p	0.0

Table A5. LLM system prompt.

You are giving instructions to a robot. Available instructions are: "go to", "pick up", "toggle", "go to goal". Available objects are: "key", "door", "ball", "box". Available colors are: "red", "green", "yellow", "blue", "purple", "grey". Give a simple answer, consisting only of available instructions, colors and objects. Don't look for alternative routes. Give your answer in a step by step format. Try to have as little as possible steps in your answer. If the door is locked, a key of the same color is needed. Doors are unlocked by toggling them while holding the key. Keys can only toggle locked doors of the same color. The robot must first pick up a key before toggling the door. Boxes do not need keys to be toggled. Consider everything directly accessible unless it is in another room. Boxes can contain other objects with the same color. If the object you are looking for is not present but there is a box, try opening the box. Make sure to reference only objects present in the scene. Do not tell the robot to go into the room or to open the unlocked door. Examples: The scene contains: -green ball -yellow ball -blue key -purple door -yellow door -green key -goal Mission: go to goal' Answer: 1. go to goal The scene contains: -green ball -purple ball -red box -purple box -green box -blue key Mission: pick up purple box' Answer: 1. pick up purple box The scene contains: Two rooms. Left and right. There is a locked gray door between the rooms
--

Table A5. *Cont.*

Left room contains:
-goal
-green ball
-yellow key
Right room contains:
-robot
-gray box
-red key
-gray ball
Mission: go to goal
Answer:
1. toggle gray box
2. pick up gray key
3. toggle gray door
4. go to goal

A comparison of the tested open-source models is shown in Table A6. Benchmarking of different LLM models was not the focus of this work. The first model with 0 errors on the test set was used.

Table A6. LLM benchmark.

Model	Match (Out of 14)
Qwen3:30b *	14
llama3:8b	9
deepseek-r1:32b	11
gemma2:27b	13

* Selected model.

References

1. Sutton, S.R.; Barto, G.A. Reinforcement Learning. In *Reinforcement Learning: An Introduction*, 2nd ed.; The MIT Press: Cambridge, MA, USA, 2018; pp. 2–5.
2. Rengarajan, D.; Vaidya, G.; Sarvesh, A.; Kalathil, D.; Shakkotta, S. Reinforcement Learning with Sparse Rewards using Guidance from Offline Demonstration. In Proceedings of the International Conference on Learning Representations (ICLR), Virtual, 25–29 April 2022.
3. Agarwal, R.; Liang, C.; Schuurmans, D.; Norouzi, M. Learning to Generalize from Sparse and Underspecified Rewards. In Proceedings of the 36th International Conference on Machine Learning, Long Beach, CA, USA, 9–15 June 2019.
4. Chevalier-Boisvert, M.; Dai, B.; Towers, M.; de Lazcano, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P.S.; Terry, J. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *arXiv* **2023**, arXiv:2306.13831. [\[CrossRef\]](#)
5. Yuksel, S.E.; Wilson, J.N.; Gader, P.D. Twenty Years of Mixture of Experts. *IEEE Trans. Neural Netw. Learn. Syst.* **2012**, *23*, 1177–1193. [\[CrossRef\]](#) [\[PubMed\]](#)
6. Jacobs, A.R.; Jordan, I.M.; Nowlan, J.S.; Hinton, E.G. Adaptive Mixtures of Local Experts. *Neural Comput.* **1991**, *3*, 79–87. [\[CrossRef\]](#) [\[PubMed\]](#)
7. Shazeer, N.; Mirhoseini, A.; Maziarz, K.; Davis, A.; Le, Q.; Hinton, G.; Dean, J. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. In Proceedings of the International Conference on Learning Representations, Toulon, France, 24–26 April 2017. [\[CrossRef\]](#)
8. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, N.A.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. In Proceedings of the 31st Conference on Neural Information Processing Systems, Long Beach, CA, USA, 4–9 December 2017. [\[CrossRef\]](#)
9. Parthasarathy, B.V.; Zafar, A.; Khan, A.; Shahid, A. The Ultimate Guide to Fine-Tuning LLMs from Basics to Breakthroughs: An Exhaustive Review of Technologies, Research, Best Practices, Applied Research Challenges and Opportunities. *arXiv* **2024**, arXiv:2408.13296. [\[CrossRef\]](#)

10. Bommasani, R.; Hudson, A.D.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, S.M.; Bohg, J.; Bosselut, A.; Brunskill, E.; et al. On the Opportunities and Risks of Foundation Models. *arXiv* **2024**, arXiv:2108.07258. [\[CrossRef\]](#)
11. Ciganović, I.; Pluškoski, A.; Jovanović, M. Smart autonomous vehicle-one proposed realisation. *Robot. Manag.* **2020**, *25*, 9–14.
12. Park, B.-J.; Yong, S.-J.; Hwang, H.-S.; Moon, I.-Y. Optimizing Agent Behavior in the MiniGrid Environment Using Reinforcement Learning Based on Large Language Models. *Appl. Sci.* **2025**, *15*, 1860. [\[CrossRef\]](#)
13. Ruiz-Gonzalez, U.; Andres, A.; Del Ser, J. Large Language Models for Structured Task Decomposition in Reinforcement Learning Problems with Sparse Rewards. *Mach. Learn. Knowl. Extr.* **2025**, *7*, 126. [\[CrossRef\]](#)
14. Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. *arXiv* **2022**, arXiv:2204.01691. [\[CrossRef\]](#)
15. Pluškoski, A.; Ciganović, I.; Jovanović, M.; Vasiljević, J. Benefits of using LLM for long-term planning with distilled sub-task model compared to End to End Reinforcement Learning in the MiniGrid Simulator. *Electronics* **2026**, *15*, 1921. [\[CrossRef\]](#)
16. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. *arXiv* **2013**, arXiv:1312.5602. [\[CrossRef\]](#)
17. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347. [\[CrossRef\]](#)
18. Bengio, Y.; Louradour, J.; Collobert, R.; Weston, J. Curriculum Learning. In Proceedings of the 26th International Conference on Machine Learning, Montreal, QC, Canada, 4–18 June 2009.
19. Florensa, C.; Held, D.; Wulfmeier, M.; Zhang, M.; Abbeel, P. Reverse Curriculum Generation for Reinforcement Learning. *arXiv* **2017**, arXiv:1707.05300. [\[CrossRef\]](#)
20. Fedus, W.; Zoph, B.; Shazeer, N. Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity. *arXiv* **2021**, arXiv:2101.03961. [\[CrossRef\]](#)
21. Ewald, D.; Rogowski, F.; Suśniak, M.; Bartkowiak, P.; Blumensztajn, P. Exploring the Cognitive Capabilities of Large Language Models in Autonomous and Swarm Navigation Systems. *Electronics* **2026**, *15*, 35. [\[CrossRef\]](#)
22. Chen, X.; Lv, H.; Yin, L.; Fang, J. Multi-Agent Collaboration for 3D Human Pose Estimation and Its Potential in Passenger-Gathering Behavior Early Warning. *Electronics* **2026**, *15*, 78. [\[CrossRef\]](#)
23. Tang, C.; Liu, Y.; Wu, Y.; Han, W.; Yin, Q.; Zheng, X.; Zeng, W.; Zhang, Q. MoE-World: A Mixture-of-Experts Architecture for Multi-Task World Models. *Electronics* **2025**, *14*, 4884. [\[CrossRef\]](#)
24. Hu, L.; Huo, M.; Zhang, Y.; Yu, H.; Xing, P.E.; Stoica, I.; Rosing, T.; Jin, H.; Zhang, H. LMGAME-BENCH: How Good are LLMs at Playing Games? *arXiv* **2025**, arXiv:2505.15146. [\[CrossRef\]](#)
25. Schulman, J.; Levine, S.; Moritz, P.; Jordan, I.M.; Abbeel, P. Trust Region Policy Optimization. *arXiv* **2015**, arXiv:1502.05477. [\[CrossRef\]](#)
26. Raffin, A.; Hill, A.; Gleave, A.; Kanervisto, A.; Ernestus, M.; Dormann, N. Stable-Baselines3: Reliable Reinforcement Learning Implementations. *J. Mach. Learn. Res.* **2021**, *22*, 1–8. Available online: <http://jmlr.org/papers/v22/20-1364.html> (accessed on 9 May 2026).
27. Eimer, T.; Lindauer, M.; Raileanu, R. Hyperparameters in Reinforcement Learning and How To Tune Them. *arXiv* **2023**, arXiv:2306.01324. [\[CrossRef\]](#)
28. Andrychowicz, M.; Raichuk, A.; Stańczyk, P.; Orsini, M.; Girgin, S.; Marinier, R.; Hussenot, L.; Geist, M.; Pietquin, O.; Michalski, O.; et al. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *arXiv* **2020**, arXiv:2006.05990. [\[CrossRef\]](#)
29. Jovanović, M.; Vasiljević, J.; Lazić, D.; Mdenica, I.; Nagamalai, D. Swarm robotics in the military operations: Challenges and opportunities. In Proceedings of the 6-th International Conference on NLP & Information Retrieval (NLPI 2025), Vienna, Austria, 15–16 March 2025.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.