Language Assisted RL Agent

Oussama Kharouiche, Abdennacer Badaoui, Hatim Mrabet

CentraleSupélec
{firstname.lastname}@student-cs.fr

1 Introduction

Reinforcement Learning (RL) has emerged as a powerful framework for autonomous decision-making, demonstrating remarkable success in complex environments. However, traditional RL agents typically rely on predefined reward structures and struggle to interpret high-level human instructions, limiting their applicability in real-world interactive settings. The ability to process and act upon natural language commands remains a significant challenge, as most RL models lack intrinsic linguistic understanding.

Our project, **Language-Assisted RL Agent**, addresses this limitation by integrating Natural Language Understanding (NLU) with RL, enabling agents to interpret and execute textual instructions in structured environments. By incorporating BERT into the RL pipeline, we develop an agent capable of translating directional commands (e.g., **"go to the leftmost upper corner"**) into precise actions within a grid-based setting. This integration bridges the gap between language comprehension and decision-making, allowing for more intuitive human-agent interactions. Through this approach, we demonstrate the potential of scalable language-assisted RL, paving the way for more adaptable and interpretable autonomous systems.

2 Environment Description

Our environment consists of a structured grid-based setting where the agent must navigate toward a designated target, which is always one of the four corners of the grid. The agent starts from a random initial position and receives observations that combine spatial awareness with language comprehension—either through its position encoded alongside a text embedding or through a grid image paired with a text embedding.

The agent has a discrete action space, consisting of the following actions:

- **Up** ($\uparrow$)
- **Down** ($\downarrow$)
- **Left** ($\leftarrow$)
- **Right** ($\rightarrow$)
- **Confirm** (signals task completion)

The reward structure is designed to encourage efficient navigation. The agent receives a penalty of -1 for each step taken, discouraging unnecessary movements, while successfully confirming at the target position grants a reward of $+60$. Given this setup, the optimal policy achieves a maximum reward of 51, encouraging the agent to reach the target in the shortest possible path before confirming.

This environment serves as a controlled yet challenging testbed for evaluating the effectiveness of language-assisted decision-making in reinforcement learning.

3 Dataset Overview

Our dataset consists of a collection of textual prompts designed to guide the RL agent in navigating a structured grid environment. These prompts, generated using the Gemini API, provide natural language instructions that specify target locations within the grid, ensuring diversity in phrasing while maintaining clarity of direction. Each prompt is paired with a corresponding goal position, one of the four corners of the grid, serving as the agent's navigation objective.

The dataset aims to bridge the gap between language comprehension and decision-making by training the RL agent to interpret and execute textual commands effectively. By learning from these structured prompts, the agent can develop robust language-conditioned policies, enhancing its ability to generalize across different instructions and improve its real-world applicability.

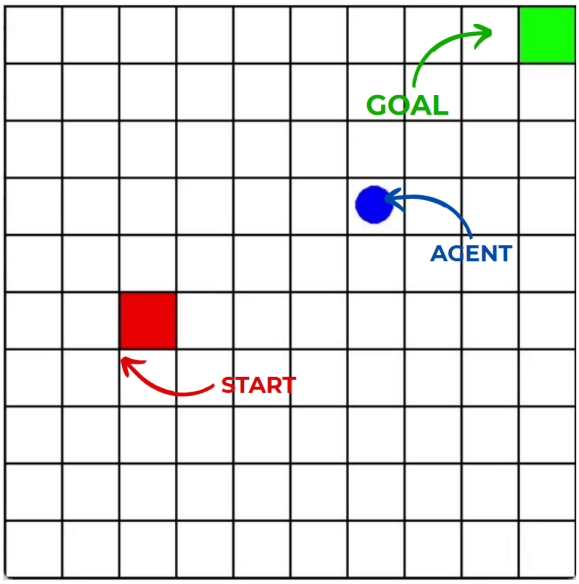


Fig. 1: Overview of the environment setup: The agent receives a language-based command and moves within the grid to reach one of the four corner targets before confirming its position.

Prompt	**Goal**
Navigate to the terminal cell, utmost on the rightmost border of the grid.	(0,9)
Move to the end of row 9 in the rightmost column.	(9,9)
Proceed to the far left of the bottom row.	(9,0)
Navigate to the first case along the top edge, leftmost location on row zero.	(0,0)

Table 1: Generated dataset used for training and testing our agents.

4 Theoretical Overview of Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is a reinforcement learning (RL) algorithm designed to improve upon the stability and efficiency of policy gradient methods. It is an on-policy algorithm that aims to strike a balance between ease of implementation, efficiency, and performance. PPO has become one of the most widely used RL algorithms due to its simple, yet effective, approach to policy optimization.

4.1 Policy Optimization and Challenges

In traditional policy gradient methods, the policy is optimized by adjusting its parameters in the direction of the gradient of the expected return, using a sample of experience. However, these methods can suffer from high variance and instability, especially when the policy changes too drastically between updates. Large policy updates can lead to performance degradation, or in some cases, catastrophic forgetting, making training unstable.

4.2 The PPO Approach

PPO addresses these challenges by introducing a novel strategy that limits the magnitude of policy updates, ensuring that the updates are not too large. This is achieved through the use of a clipped objective function. The key idea behind PPO is to ensure that each new policy does not deviate significantly from the previous one, which improves stability during training.

The objective function in PPO is based on a surrogate loss that combines the traditional policy gradient with a clipping mechanism. The clipping mechanism helps avoid large policy updates by penalizing any changes that would move the policy too far from its previous state. This prevents overfitting to particular trajectories and helps maintain stable learning.

The PPO objective is given by:

$$L^{CLIP}(\theta) = \hat{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$$

where:

- $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between the current and previous policies.
- $\hat{A}_t$ is the advantage estimate at time step t.
- ϵ is a small positive hyperparameter that determines the clipping range.

This clipped objective ensures that the policy update does not move too far from the previous policy, thus maintaining stability. If the probability ratio $r_t(\theta)$ is within the range $[1-\epsilon, 1+\epsilon]$, the objective function remains unchanged. If the ratio goes beyond this range, the objective is clipped, preventing excessive updates.

4.3 Advantages of PPO

- **Sample Efficiency:** PPO strikes a good balance between sample efficiency and computation. Unlike some other on-policy algorithms, PPO performs well with a relatively low number of environment interactions.
- **Stability:** The use of clipping in the objective function helps stabilize training by reducing the impact of large policy updates.
- **Simplicity:** PPO is relatively easy to implement compared to other advanced RL algorithms like Trust Region Policy Optimization (TRPO), as it avoids the need for second-order optimization or complex line searches.
- **Scalability:** PPO can be easily applied to a wide range of environments, making it suitable for both discrete and continuous action spaces.

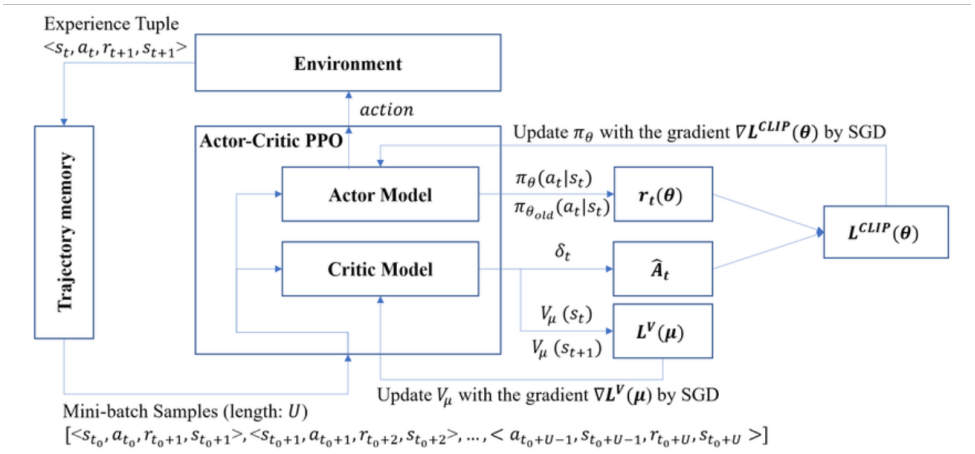


Fig. 2: Overview of the PPO Algorithm

5 Methodology

In this work, we adapted Proximal Policy Optimization (PPO) to include dual architectures for both the actor and critic networks. This modification allows us to leverage multimodal inputs—specifically combining spatial and textual information—which improves the agent's decision-making process, particularly in tasks requiring language-guided actions and visual context integration.

Our approach is designed to enhance PPO's capability by incorporating either **multi-input networks** and a **visual attention pathway**, enabling the agent to process and respond to visual or textual cues in a more robust manner.

5.1 Multi-Input Network: Agent Position and Text Embeddings

The main component of our first approach is the **multi-input network**, which processes the **agent's position (coordinates)** along with **text embeddings** generated by a pretrained BERT model. The agent's position in the environment is crucial for guiding its spatial navigation and interaction with other objects, while the text embeddings—derived from natural language instructions—are key to interpreting the context and desired actions.

This dual input structure enables the actor network to make **language-guided decisions**. By combining both the agent's physical coordinates and textual descriptions, the policy can be better informed about the specific goals or tasks the agent is expected to complete. The textual data, encoded as high-dimensional embeddings, provides semantic context that augments the agent's understanding of its environment, allowing for more accurate and context-aware decision-making.

The text embeddings are processed through the BERT model, which encodes the linguistic features of the input instruction. This textual information is then integrated into the PPO framework, where it helps inform the policy by guiding actions that are aligned with the provided language commands.

5.2 Visual Attention Pathway: Cross-Attention between Visual Features and Text Embeddings

The second experiment's key component is the **visual attention pathway**, which plays a critical role in processing the agent's perception of the environment. In this pathway, we first use a **Convolutional Neural Network (CNN)** to extract grid-based features from the visual input. These grid features represent different regions of the environment, allowing the agent to process visual cues such as objects, obstacles, or other relevant elements.

Once the visual features are extracted, we apply **cross-attention mechanisms** to dynamically fuse the visual and textual representations. The cross-attention mechanism allows the network to attend to specific parts of the visual input that are most relevant to the instruction, based on the semantic content of the text embeddings. This fusion of visual and textual information creates a **context-aware representation** that is used for both policy and value estimation.

By applying attention between the visual features and the text embeddings, the model can focus on the regions in the visual input that correspond to the current instruction or task, thereby **prioritizing areas of the environment** that are most aligned with the goal described in the language input. This dynamic attention mechanism enables the agent to adapt its behavior based on the task at hand, ensuring that it attends to the most critical visual cues that will guide its actions.

5.3 Exploring Multimodal Inputs for PPO: Text, Position, and Visual Features

By conducting these two experiments, we explore how different input modalities—text embeddings combined with the agent's position in one experiment, and visual features combined with text embeddings in the other—impact the performance of the PPO algorithm. These distinct multimodal approaches allow the agent to adapt and navigate complex environments using either linguistic or visual context, making PPO more efficient and adaptable in language-guided decision-making tasks.

5.4 Fine-Tuning BERT Embeddings for Guiding the RL Agent

In this work, we use BERT embeddings to represent textual prompts provided to the RL agent. The goal is to guide the agent by leveraging the features extracted from textual instructions, helping it navigate the environment efficiently towards the goal.

However, some instructions may differ by only a single word while representing distinct goals, causing their embeddings to be highly similar. This similarity can pose a challenge for the agent to distinguish between instructions, hindering its ability to learn effectively. To address this issue, we fine-tuned the BERT model in two ways: through **sequence classification** and **contrastive learning**.

Fine-Tuning BERT with Sequence Classification In the first fine-tuning approach, we train BERT on a **sequence classification task**. Specifically, we train the model to classify textual instructions into their corresponding goals. This technique encourages the model to cluster instructions that share the same goal in the embedding space. By doing so, we improve the intra-label similarity (i.e., the similarity of embeddings within the same goal) and reduce the cross-label similarity (i.e., the similarity of embeddings between different goals). This classification-based fine-tuning ensures that the model better distinguishes between different goals, even when the instructions are semantically similar.

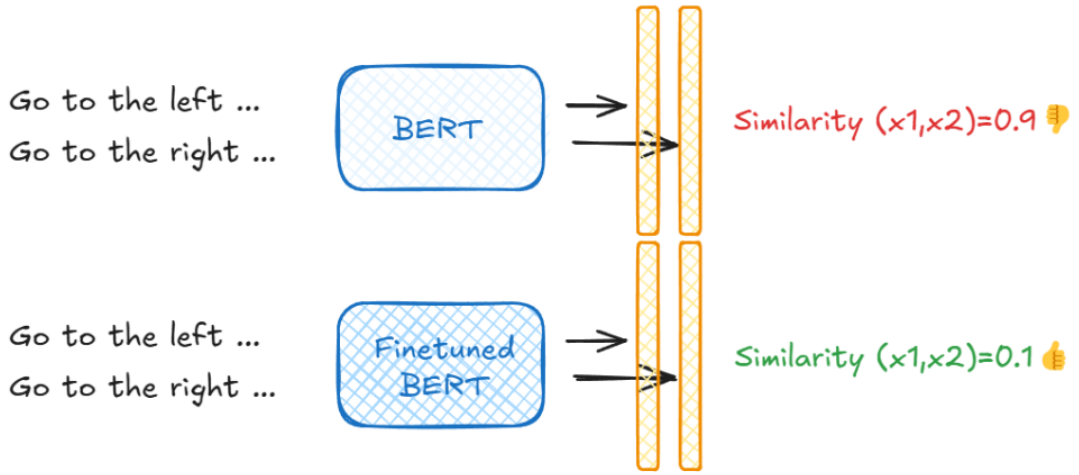


Fig. 3: BERT Finetuning for enhanced textual embeddings using sequence classification

Fine-Tuning BERT with Contrastive Learning and Triplet Loss To further resolve ambiguity in instructions that sound similar but have distinct meanings, such as "top-left" versus "bottom-left", we apply **contrastive learning** with a **triplet loss**. This method is designed to map semantically similar commands closer in the embedding space while pushing dissimilar ones apart. We use triplet loss to optimize this objective, where each training instance consists of three components: an anchor (a), a positive (p) example that is semantically similar to the anchor, and a negative (n) example that is semantically dissimilar.

The traditional triplet loss function is defined as:

$$L_{triplet} = \|\mathbf{e}_a - \mathbf{e}_p\|_2^2 + \max\left(\alpha - \|\mathbf{e}_a - \mathbf{e}_n\|_2^2, 0\right),$$

where: - $\mathbf{e}_a$, $\mathbf{e}_p$, and $\mathbf{e}_n$ are the embeddings of the anchor, positive, and negative examples, respectively, - $\|\cdot\|_2$ denotes the Euclidean distance between embeddings, - α is a margin that ensures that the negative example is pushed farther from the anchor than the positive example by at least a margin α.

However, in our case, the margin is not fixed but dynamically computed for each pair based on the BERT embeddings. The margin $\alpha_{dynamic}$ for each triplet loss is defined as a linear combination of the distance between the text embeddings of the anchor and positive examples, and the real spatial distance between the corresponding grid points, such as corners in the environment. Specifically, the dynamic margin is calculated as:

$$\alpha_{dynamic} = \lambda_1 \|\mathbf{e}_a - \mathbf{e}_p\|_2^2 + \lambda_2 d_{grid}^2,$$

where: - $\|\mathbf{e}_a - \mathbf{e}_p\|_2$ is the Euclidean distance between the embeddings of the anchor and positive examples (textual similarity), - d_{grid} is the real distance between the two corresponding grid points (e.g., corners), - λ_1 and λ_2 are hyperparameters that control the relative importance of textual similarity and spatial distance in the margin computation.

This dynamic margin adjusts the loss function to place more emphasis on the distance between text embeddings for instructions that are more semantically similar, as well as the actual spatial relationship between the grid points (e.g., the distance between the corners). By doing so, we ensure that the model not only distinguishes between semantically similar instructions but also takes into account the spatial context of the instructions, improving its ability to resolve ambiguities.

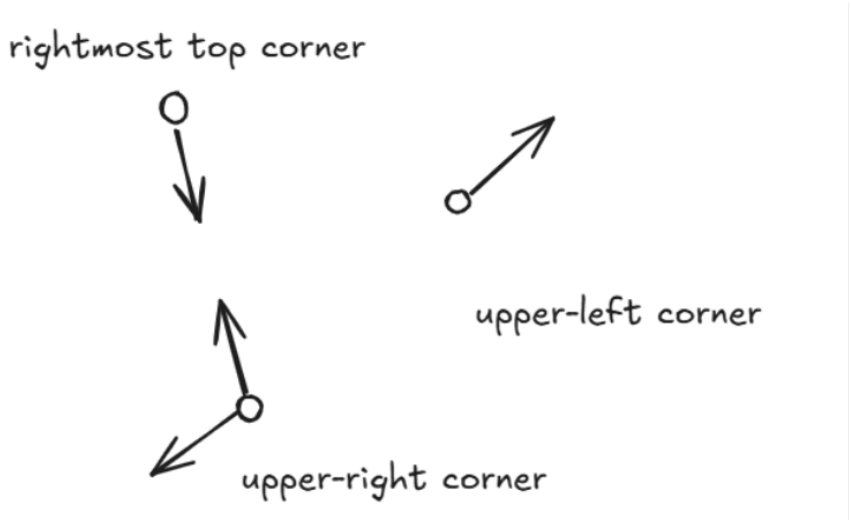


Fig. 4: BERT Finetuning for enhanced textual embeddings using contrastive learning

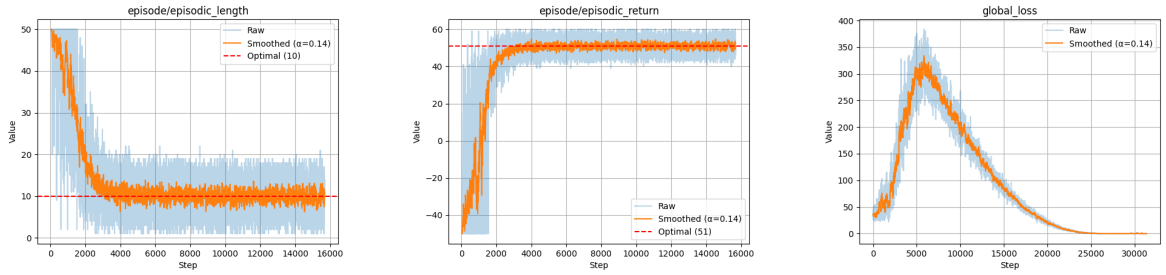
Through this approach, we fine-tune BERT embeddings to both differentiate similar commands and account for real-world spatial relationships, ensuring that the agent can effectively interpret and act on textual instructions in a variety of environments.

6 Experiemnts and Results

In order to validate the functionality of our PPO, we will experiment with a simple task without any textual guidance. The agent starts from a random position in the grid with a randomly assigned goal at one of the four corners. The agent receives as an observation its position coordinates and the coordinates of the corner goal. Below, we present the results obtained for this experiment (Fig. 5). The agent successfully reaches the optimal reward of 51 (Fig. 5b) and the optimal average path length of 10 (Fig. 5a), which indicates that it has learned an efficient policy to navigate the grid and reach the goal.

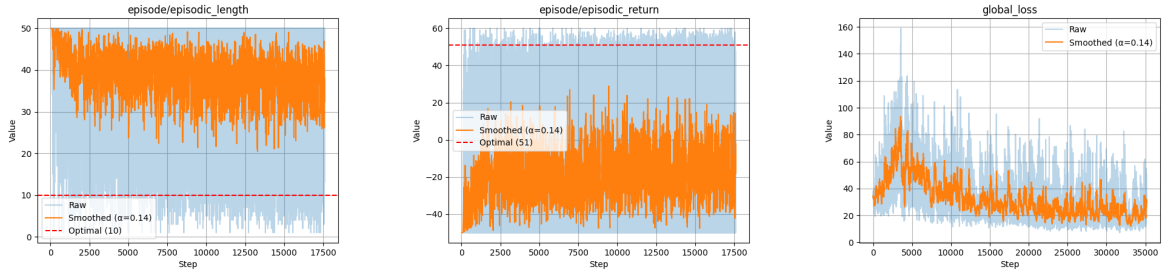
To guide our RL agent, we will use BERT embeddings as representations of the natural language commands provided by the user. As a first experiment, we will directly use these embeddings to encode textual instructions and evaluate their effectiveness in guiding the agent. Below are the results obtained in this setup :

The agent performs poorly when guided by the raw BERT embeddings. This outcome was expected, as our textual commands are often similar, sometimes differing by just a single word (e.g., "Go to the



(a) the average length over episodes (b) the average reward over episodes (c) The evolution of the loss.

Fig. 5: Results - RL agent without textual guidance



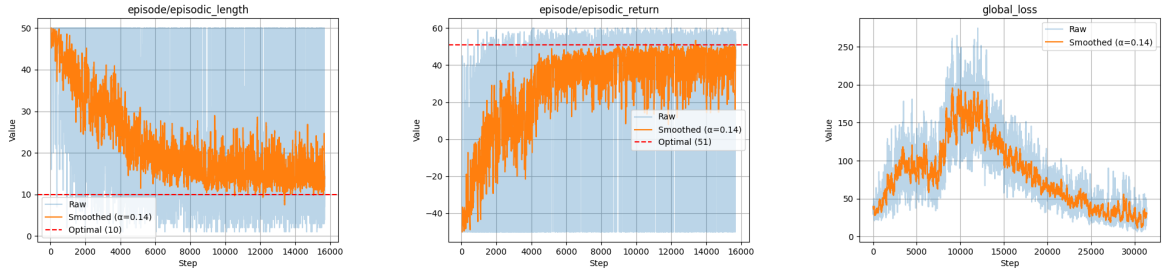
(a) the average length over episodes (b) the average reward over episodes (c) The evolution of the loss.

Fig. 6: Results - RL agent with raw BERT embeddings

left-top corner" vs. "Go to the right-top corner"). Since BERT embeddings capture semantic similarity, these commands result in very similar embeddings, making it difficult for the agent to distinguish between them.

A possible solution is to apply contrastive learning to explicitly enforce distinct representations for subtly different instructions. Additionally, fine-tuning BERT using a sequence classification objective could help the model better differentiate between similar commands by learning task-specific nuances.

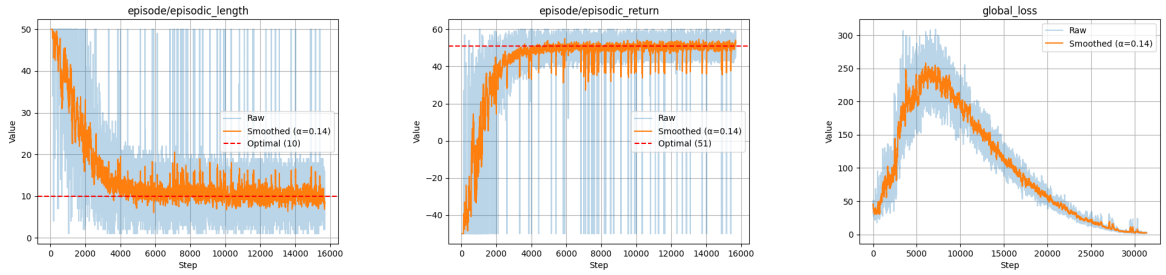
Here are the results of the contrastive learning approach (Fig. 7). While it approaches the optimal value, it still struggles with high variance, as the embeddings remain insufficiently distinguishable.



(a) the average length over episodes (b) the average reward over episodes (c) The evolution of the loss.

Fig. 7: Results - RL agent with the contrastive learning approach

Another, more interesting approach is to fine-tune the BERT model by trying to solve a sequence classification task, as we have already presented in Section 5.4. This will reduce the similarities between different classes, making it easier for our agent to distinguish between the corner goals. Below, we present the results of this approach.

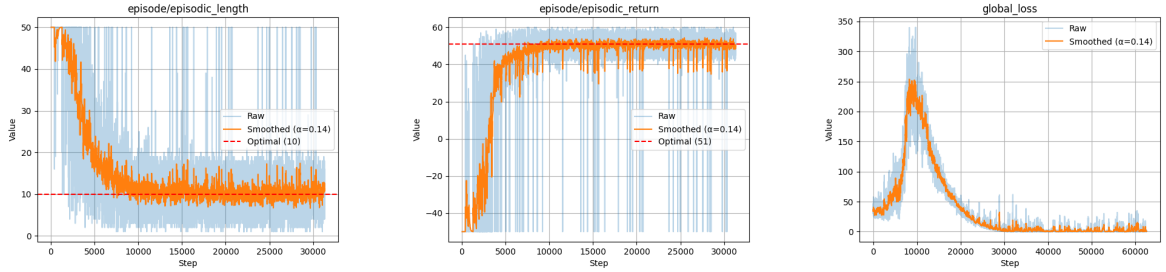


(a) the average length over episodes (b) the average reward over episodes (c) The evolution of the loss.

Fig. 8: Results - RL agent with the finetuned BERT embeddings on sequence classification task

This approach yields our best results. The RL agent rapidly converges to the optimal policy, achieving an optimal reward of 51 and an optimal path length of 10 in just 4000 steps (8). Its performance is comparable to our first experiment, where the agent was given the exact position of the goal as observation. However, in this case, the agent relies solely on textual guidance, which can be expressed in any style, as long as it clearly specifies the target corner. This demonstrates the effectiveness of our approach in interpreting natural language instructions to guide the agent's behavior, using finetuned BERT embeddings.

Cross-attention between visual features and text embeddings, as presented in Section 5.2, could yield interesting results. Below are the results obtained for this approach:



(a) the average length over episodes (b) the average reward over episodes (c) The evolution of the loss.

Fig. 9: Results - RL agent using grid attention with the finetuned BERT embeddings on sequence classification task

The agent shows similar results to the one with fine-tuned BERT embeddings, but it is slower in terms of convergence, requiring approximately 10,000 steps (9) to converge compared to the other agent, which converges in just 4,000 steps. This might be because we skipped hyperparameter tuning—stuff like adjusting the learning rate or batch size—since it's super computationally expensive. The attention mechanism over the grid was made so the model can learn to concentrate over the subgrid where the goal resides

7 Code Implementation

The full implementation of the Project is available in Github:
Language-Assisted-RL-Agent

8 Conclusion

We presented a language-assisted reinforcement learning framework that successfully integrates BERT-based natural language understanding with Proximal Policy Optimization. Our experiments demonstrate

that fine-tuned text embeddings enable effective translation of natural language instructions into navigation policies, achieving optimal performance comparable to position-guided agents. Key contributions include:

- A novel dual architecture for PPO combining spatial/textual inputs
- Dynamic cross-attention mechanisms for visual-language fusion
- Contrastive and Sequence Classification fine-tuning strategy resolving semantic ambiguities

The framework's success in our grid environment suggests broader applicability to language-guidance in RL tasks.

9 promising leads

Our framework lays the groundwork for several impactful extensions:

- **Complex Language-Guided Tasks**: Extending the environment with interactive objects (e.g., keys/doors) to enable hierarchical instructions like *"take the green key then open the blue door"*, building on the compositional approach of BabyAI [1].
- **Adversarial Reward Learning**: Integrating GAN-based architectures to learn language-aligned reward functions from demonstrations, inspired by recent advances in inverse reinforcement learning with textual goals [2].
- **Multimodal Attention Scaling**: Expanding our cross-attention mechanisms to handle sequential subgoals and object relationships through transformer-based architectures.

References

1. Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, Yoshua Bengio: BabyAI: A Platform to Study the Sample Efficiency of Grounded Language Learning.
2. Li Zhou, Kevin Small: Inverse Reinforcement Learning with Natural Language Goals.