Large Language Models and Reinforcement Learning for Sequential Decision Making

Bissmella Bahaduri
MVA, ENS Paris-Saclay
`bissmellabahaduri@gmail.com`

Lab Supervisors: Olivier Sigaud, Nicolas Thome, Laure Soulier, Mohamed Salim Aissi
{olivier.sigaud, nicolas.thome, laure.soulier, mohamed-salim.aissi}@sorbonne-universite.fr
ISIR, Sorbonne University

University Supervisor: Marc Lelarge

Internship Period: 01/03/2025 - 30/09/2025

Contents

1 Acknowledgments

I heartily thank Professor Olivier Sigaud for his support, supervision, and patience during this research internship. I also thank Professors Nicolas Thome, Laure Soulier, and PhD candidate Mohamed Salim Aissi for their kind guidance and supervision. Further, I would like to thank Professor Marc Lelarge for both his supervision of my internship and his great course on LLMs. I also wish to thank the team in charge of the MVA Masters at ENS Paris-Saclay for the organization of the internships, both on the pedagogic and on the administrative side.

2 Introduction

General context: This internship was conducted within the MLIA (Machine Learning – Deep Learning and Information Access) team at ISIR (Institute of System Intelligence and Robotics) lab, as part of the PILLAR project. A central goal of the internship is to explore how recent advances in foundation models, particularly language and vision language models, can be combined with reinforcement learning to solve complex sequential decision-making problems, including those involving visual inputs, partial observability, or open-ended goals.

Recent times have seen rapid progress in both reinforcement learning (RL) and large language models (LLMs), each revolutionizing artificial intelligence. Reinforcement learning (Sutton et al., 1998) has proven highly effective for sequential decision-making tasks, especially in environments where agents must learn optimal policies through trial-and-error interaction. On the other hand, LLMs, and more recently vision-language models (VLMs), have demonstrated impressive capabilities in reasoning, planning, and multi-modal understanding across diverse domains. While RL provides a principled framework for decision making, it often suffers from poor sample efficiency, limited generalization, and brittle exploration strategies. LLMs, on the other hand, possess rich world knowledge and can perform complex reasoning, but lack the grounding and interactivity required to act in dynamic environments. Bridging these two paradigms opens the door to agents that can not only act and learn but also reason, explain, and adapt.

Recently there has been many different reinforcement algorithms (Shao et al., 2024; Ouyang et al., 2022; Zheng et al., 2025) proposed for post-training language models. Many of these post-training methods are based on group based value estimation. These group-based RL algorithms have proven to be efficient for single-turn problem solving and question answering.

However, in multi-turn, and interactive settings the situation is far more difficult. The success depends on large number of steps where at each of those steps the LLM/VLM agent has to reason and choose between different possible answers. As an example, in the ALFWorld (Shridhar et al., 2020) environment, the agent needs to take up to 50 steps, and each step involves hundreds of tokens. A major issue that makes things even more difficult is that there is no feedback on each step, but the agent might know the effect of its action in a much later step in the trajectory. This situation named as reward sparsity complicates step-based and token based credit assignment.

In addition to the mentioned challenges, exploration for LLM-based agents remains another difficulty. Though the LLMs provide priors that they have learned during pre-training and post-training, these priors are not grounded in the real world. Moreover, the large size of LLMs makes it computationally expensive to interact with an environment to gather trajectories and data for training. Sampling from the LLMs also remains a big challenge due to the auto-regressive nature of these models.

Recently, VIPER (Aissi et al., 2025) introduces merging the perception capability of VLM and the reasoning capability of an LLM to solve interactive tasks in the ALFWorld (Shridhar et al., 2020) environment, by training the LLM while keeping the VLM frozen. Previously, RL4VLM (Zhai et al., 2024) has introduced a framework for finetuning VLMs in interactive environments.

Finally, nearly all of the existing methods for training LLMs in interactive environments heavily depend on an initial phase of supervised fine-tuning based on expert data (Zhai et al., 2024; Aissi et al., 2025) which is collected either by a human or another more powerful LLM such as ChatGPT. This makes the training of LLMs expensive, and if in a problem the expert does not exist or the teacher LLM does not have expertise, then it becomes completely a failure.

Our work is mainly exploratory in this domain with efforts to deal with the problems mentioned, with the hope of finding a solution. Exploration as one of the main issues in RL is a big portion of our work. In the same way that exploration is a challenge in RL, it is a challenge in research as well, and the *exploration in exploration* becomes a two-fold challenge. This report tries to raise and respond to two main questions:

- Can pretrained models explore well?
- How can we enable VLM/LLM agents to explore better in interactive environments?

In order to address all these, this report aims to first study and analyze the current existing literature, analyze VIPER (Aissi et al., 2025) which is based on leveraging both LLM and VLM as a case study, and explore new methods to enhance exploration. Our objectives are:

- To survey recent advances in RL, LLMs, and VLMs, with a focus on their integration.
- To analyze specific algorithms that attempt this integration, such as VIPER (Aissi et al., 2025), and RL4VLM (Zhai et al., 2024).
- To provide an in-depth analysis of VIPER, particularly regarding the balance between visual grounding and language-based reasoning.
- To study how exploration strategies can be guided by curiosity or model priors in LLM-based/VLM-based agents.
- To explore and enhance the exploration in VLM reinforcement learning based training strategies.

The main contributions of this work are:

- A structured literature review on the use of RL with LLMs/VLMs in sequential decision making.
- Insights from a case study analyzing the reasoning vs vision trade-off in VIPER.
- An investigation into exploration strategies that leverage curiosity.
- A study of exploration based on model priors.

3 Literature review

This section provides a comprehensive review and analysis of the existing literature on sequential decision making, reinforcement learning, and the role of large language and vision-language models in these contexts.

3.1 Sequential decision making

Alongside perception, decision making is also a major capability of intelligent agents. Sequential decision making refers to problems where an agent must make a series of decisions over time to achieve a goal, often under uncertainty and in dynamic environments (Puterman, 2014). The agent's decisions influence not only immediate rewards but also future states and available actions. This paradigm is central to many domains such as robotics (Kober et al., 2013), game playing (Mnih et al., 2015), user interface control, and autonomous driving.

Formally, such problems are often modeled as Markov Decision Processes (MDPs), defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ the action space, $\mathcal{P}$ the transition probability function, $\mathcal{R}$ the reward function, and γ the discount factor. In partially observable environments, the model is extended to Partially Observable MDPs (POMDPs), where the agent receives observations that provide only partial information about the true state.

Effective sequential decision making requires a combination of planning, learning, and exploration. In recent years, there has been a growing interest in augmenting agents with high-level cognitive abilities, such as reasoning or abstraction, especially for generalization and sample efficiency in complex domains (Lake et al., 2017; Guez et al., 2019).

3.2 LLMs and VLMs

Large Language Models (LLMs), such as GPT series (Radford et al., 2018, 2019; Ouyang et al., 2022), PaLM (Chowdhery et al., 2023), LLaMA series (Touvron et al., 2023) and Mistral (Jiang et al., 2024)to name a few, have achieved remarkable success in various natural language processing tasks due to their ability to learn powerful representations from large-scale text corpora. These models are capable of zero-shot and few-shot generalization (Brown et al., 2020), in-context learning (Dong et al., 2022), reasoning (Wei et al., 2022; Kojima et al., 2022) over structured and unstructured inputs, and planning (Huang et al., 2022). However, there has also been some controversy regarding the planning capability of language models (Valmeekam et al., 2023).

Vision-Language Models (VLMs) (Liu et al., 2023; Alayrac et al., 2022; Zhu et al., 2023) extend LLMs to handle visual inputs, integrating modalities through image-text foundational models (Radford et al., 2021; Jia et al., 2021; Li et al., 2022) and pretraining on image-text pairs. These models have been used for visual question answering (Antol et al., 2015), image captioning (Vinyals et al., 2015), and even multi-turn interactive tasks (Liu et al., 2023). They provide a promising interface between high-level reasoning and grounded perception, making them attractive candidates for decision-making systems in real-world environments.

Recent efforts have explored using LLMs and VLMs not just for passive tasks (single turn generation), but for *interactive* tasks, guiding action selection (Ahn et al., 2022), generating plans and interpreting environment feedback(Yao et al., 2023). This bridges the gap between language-driven reasoning and low-level decision making.

3.3 Reinforcement learning

Reinforcement learning (RL) (Sutton et al., 1998) is a learning paradigm where agents learn to take actions in an environment to maximize cumulative rewards. Classical RL methods include value-based approaches such as Q-learning (Watkins and Dayan, 1992) and DQN (Mnih et al., 2015), policy gradients based methods such as REINFORCE (Williams, 1992) and PPO (Schulman et al., 2017), and actor-critic frameworks. These methods have been widely applied to domains like robotics (Levine et al., 2016), games (Silver et al., 2016, 2018; Schrittwieser et al., 2020), and autonomous systems (Kiran et al., 2021).

However, despite their successes, standard RL methods face several challenges:

- **Sample inefficiency**: Agents typically require millions of interactions to learn effective policies. LLMs can help by the priors it has gained during pre-training.
- **Exploration**: In complex or sparse-reward environments, random exploration often fails to discover useful behavior. The exploration can be informed by the LLM knowledge.
- **Generalization**: Learned policies often overfit to specific environments and struggle with transfer. LLM/VLM-based policies might generalize better due to broad knowledge.

To address these issues, recent work has explored integrating external priors, hierarchical learning, meta-learning, and the use of pretrained models to inform policy learning. The integration of LLMs and VLMs into RL workflows is one such promising direction, potentially enabling agents to ground decisions in rich semantic knowledge, reason about long-term consequences, and generalize across diverse tasks. A comprehensive review of reinforcement learning algorithms is beyond the scope of this report and we just mention the ones important to language models finetuning and training.

3.3.1 Proximal policy optimization

Using samples generated from the old policy $\pi_{\theta_{old}}$, PPO (Schulman et al., 2017) constrains the policy update by clipping the policy gradient, and a KL regularization term to stabilize the training. Specifically PPO employs the following objective:

$$\mathcal{L}_{\text{PPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\ y\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \left[\frac{1}{|y|} \sum_{t=1}^{|y|} \min\Big(w_t(\theta) A_t,\ \text{clip}(w_t(\theta),\ 1-\epsilon,\ 1+\epsilon)\hat{A}_t \Big) - \beta \cdot \text{KL}\left[\pi_{\theta_{\text{old}}}(\cdot|x_t) \parallel \pi_\theta(\cdot|x_t)\right] \right] \tag{1}$$

where $w_t(\theta) = \frac{\pi_\theta(y_t|x, y_{<t})}{\pi_{\theta_{\text{old}}}(y_t|x, y_{<t})}$, and the advantage $\hat{A}_t$ is estimated using a value model. A main challenge in PPO has been its reliance on the value model that can be noisy and inaccurate from one side, and requires more memory from the other side to store the value model. This challenge becomes even more pronounced in the scenarios of sparse reward environments.

3.3.2 Group relative policy optimization

GRPO (Shao et al., 2024) solves the problem of value estimation in PPO by estimating advantage of one response relative to a group of responses to a query. The main idea in GRPO is to estimate the value relative to a group instead of relying on a critic model. Different from PPO, GRPO optimizes for the following objective:

$$\mathcal{L}_{\text{GRPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\ \{y_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\left(w_{i,t}(\theta)A_{i,t},\ \text{clip}(w_{i,t}(\theta),\ 1-\epsilon,\ 1+\epsilon)A_{i,t}\right)\right] \tag{2}$$

where G (group size) is the number of generated responses in a group for a given query. The KL divergence is omitted in many implementations and we avoid denoting it here for brevity. The policy ratio $w_{i,t}$ and advantage $A_{i,t}$ are as follow:

$$w_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}|x_{i,<t})}{\pi_{\theta_{old}}(y_{i,t}|x_{i,<t})}$$

$$A_{i,t} = A_i = \frac{r(x, y_i) - 1/G\sum_{i=1}^{G} r(x, y_i)}{\text{std}(\{r(x, y_i)\}_{i=1}^{G})}$$

All the tokens in y_i share the same advantage as A_i.

3.4 Language models and sequential decision making

Large Language Models (LLMs) have emerged as powerful tools for reasoning, planning, and interacting with complex environments. In the context of sequential decision making, LLMs can either serve as the policy itself or act as a high-level planner guiding low-level control modules. The literature can be broadly divided into two major lines of work.

One line of research focuses on *finetuning* LLMs to improve their performance on sequential decision-making tasks. These methods adapt the pretrained model to the target environment using reinforcement learning or supervised imitation from expert trajectories. For instance, the GLAM framework (Carta et al., 2023) (Grounded Language Model) applies reinforcement learning with environment feedback to align LLM outputs with task-specific goals. This adaptation involves:

- Collecting environment interaction data (states, actions, and rewards).
- Incorporating environment reward.
- Estimating advantages and updating model parameters using algorithms such as PPO.

Such methods generally yield strong task-specific performance but require significant computational resources, careful reward design, and a substantial amount of interaction data from the environment.

Another line of work leverages LLMs *without modifying their parameters*, instead relying on prompt engineering, in-context learning, and structured interaction patterns. In these approaches, the pre-trained LLM is treated as a frozen reasoning engine, while control is delegated to an external loop that translates environment observations into textual prompts. Examples include:

- **SayCan** (Ahn et al., 2022), which integrates a frozen LLM with trained an affordance functions to select actions grounded in robotic capabilities.
- **ReAct** (Yao et al., 2023), which interleaves reasoning traces and actions in the LLM's output to enable sequential interactive problem-solving.
- Tool-augmented prompting systems like (Schick et al., 2023), where the LLM queries APIs, knowledge bases, or other models as part of its decision-making process.

These methods are often more sample-efficient since they avoid costly parameter updates and can adapt quickly to new tasks by changing the prompt, but they may underperform finetuned models on specialized domains due to the mismatch between the pretraining data distribution and the task environment. In addition, these methods rely on prompting and prompt-engineering which becomes very cumbersome in some situations. Usually, in the case of in-context learning, the few-shot examples that work for one problem do not work for a slightly different problem (Verma et al., 2024). Finally, there is a relatively low limit on the performance that can be achieved by these methods and it saturates fast.

3.4.1 VIPER

VIPER (Aissi et al., 2025) proposes a modular framework for *multimodal instruction-based planning* that integrates a frozen Vision-Language Model (VLM) for visual *perception* with a Large Language Model (LLM) for *reasoning* and action selection. Concretely, the VLM first converts image observations into textual scene descriptions, which serve as an intermediate, human-interpretable representation. An LLM is then conditioned on the goal instruction and the textualized observation to produce actions. The LLM module is further adapted via behavioral cloning and reinforcement learning, while the perception module remains frozen. On ALFWorld (Shridhar et al., 2020), VIPER significantly improves success rates over prior visual instruction planners and narrows the gap to text-oracle policies, and its text-based intermediate representation supports fine-grained, post-hoc analysis of perception vs. reasoning (Aissi et al., 2025).

We distinguish *visual understanding* from *reasoning* as follows. Visual understanding (perception) entails mapping pixels or video frames to semantically meaningful entities (objects, attributes, relations, spatial layouts) that are *grounded* in the scene. Reasoning, in contrast, operates primarily over symbolic/linguistic representations, deriving plans, multi-step deductions, counterfactuals, and task decompositions, potentially conditioned on world knowledge and goals. VIPER operationalizes this split by (i) delegating perception to a VLM that produces textual descriptions and (ii) delegating planning/decision making to an LLM policy that consumes those descriptions alongside the instruction (Aissi et al., 2025). This separation clarifies error sources: failures can be attributed to misperception (incorrect or incomplete textualization) or misreasoning (poor plan or action choice given a correct description).

Earlier approaches typically emphasized either LLM-based planning with weak/implicit perception, or VLM-based perception without a dedicated planning module:

- **LLM-centric (no finetuning / frozen LLM).** Methods such as SayCan (Ahn et al., 2022) and ReAct (Yao et al., 2023) keep the LLM frozen and leverage prompting, affordance scores, or tool-use to plan actions. SayCan grounds LLM-generated action proposals in the robot's skill affordances (feasibility and value), while ReAct interleaves chain-of-thought with environment actions for web or textual environments (e.g., ALFWorld). These systems showcase strong *reasoning and planning* but rely on external heuristics/functions or task-specific tools to bridge perception.
- **VLM-centric (perception-first).** A complementary line develops stronger multimodal encoders/decoders for visual understanding (e.g., CLIP-derived or instruction-tuned VLMs), sometimes with light prompting or tool routing (e.g., MM-ReAct (Yang et al., 2023)). Another line of work (Zhai et al., 2024) finetunes a VLM specifically for sequential decision making in interactive environments.
- **LLM-VLM merged** close to VIPER, EMMA (Yang et al., 2024), first trains an LLM in the text-only environment, and then distill the knowledge of LLM into VLM for image-based environment.

VIPER explicitly marries these strands: a *frozen* VLM handles perception; an *LLM policy* handles planning and is adapted with BC/RL for sequential decision making (Aissi et al., 2025).

Recent studies suggest that VLMs' apparent reasoning often stems from pattern recognition and dataset priors; genuine multi-step visuolinguistic reasoning remains challenging. Empirical work has examined chain-of-thought (CoT) for VLMs, finding that naive CoT prompting can underperform or even *harm* results without targeted data or objectives, whereas targeted preference/post-training can improve multimodal CoT quality (Zhang et al., 2024b,a). These findings motivate VIPER's design: use VLMs primarily for robust visual grounding, and concentrate planning in the LLM policy, which can be finetuned with RL to handle sequential dependencies.

In contrast to prior systems that either (a) treat the LLM as a frozen planner with ad-hoc perception bridges (SayCan, ReAct) or (b) rely on VLMs with limited sequential planning, VIPER formalizes a *two-stage, explainable* pipeline with clear interfaces: visual understanding (VLM $\rightarrow$ text) and language-based reasoning/planning (LLM $\rightarrow$ actions), and it trains the latter for long-horizon decision making.

Using $\mathcal{V}$ to denote the discrete and finite token space and $\mathcal{V}^m$ input text space, where m is the maximum token length of text input. The input state space is: $\mathcal{S} = \mathcal{O} \times \mathcal{V}^m$ where $\mathcal{O}$ is RGB image space. A frozen VLM denoted by VLM is used to map from the image space to tokens space with some maximum token length n as $\mathcal{V}^n = VLM(\mathcal{O})$ and the textualized state is $\mathcal{S}' = \mathcal{V}^n \times \mathcal{V}^m$. Then an LLM policy can be written as $\pi_\theta : \mathcal{V}^n \times \mathcal{V}^m \to \mathcal{V}^{out}$ where $\mathcal{V}^{out}$ is the output action.

VIPER adopts a multi-stage training pipeline that combines supervised imitation and reinforcement learning:

1. **Perception module**: The VLM used for visual grounding is kept *frozen* throughout training, ensuring stable and consistent scene textualization. This module is typically pretrained on large-scale image-text pairs and is not updated to avoid catastrophic forgetting of general visual concepts.
2. **Reasoning/policy module**: The LLM-based planner is first initialized from a pretrained instruction-following language model and then trained in two phases:
   - *Behavioral cloning (BC)* from expert demonstrations, where the model learns to map VLM-generated textual descriptions and instructions to action sequences.
   - *Reinforcement learning (RL)* fine-tuning, optimizing task success rate using PPO. This phase enables adaptation to environment-specific dynamics and long-horizon dependencies.

This staged training approach allows the reasoning module to leverage strong priors from pretrained LLMs while progressively aligning to the target environment through direct interaction.

VIPER's decision-making draws inspiration from the GLAM (Carta et al., 2023) paradigm, where policy elicitation is formulated as an *instruction-to-action* generation problem. In this setup:

- The LLM receives a structured prompt containing the task instruction, the current (textualized) state description from the VLM, and the history of past actions.
- The model autoregressively generates the next action tokens by teacher forcing, akin to language modeling as continuation of the prompt. This is repeated for all possible actions in the current state.
- The probability of possible actions is then calculated by summing the probabilities of the tokens for each possible action, and their maximum is selected as the policy action.

By converting images to easily understandable, interpretable and debuggable textual tokens the LLM can decide on an action, while also benefiting from LLMs' capacity for structured reasoning and flexible plan adaptation.

3.4.2 RL4VLM

Reinforcement Learning for Vision-Language Models (RL4VLM) (Zhai et al., 2024) is another recent framework designed to enhance the reasoning capabilities of multimodal models by aligning their outputs with desired behaviors through a combination of supervised learning and reinforcement learning. RL4VLM focuses on making VLMs not only predict responses grounded in visual inputs but also *generate explicit reasoning traces*, thereby improving interpretability and task performance.

A core contribution of RL4VLM is the integration of explicit *thoughts* into the VLM model's responses. Previously, CoT reasoning has shown promising result in LLMs but it was less explored for VLM multi-step interactive decision making. Instead of producing a direct answer to a visual-linguistic prompt, the RL4VLM model first generates an intermediate reasoning CoT trace that explains how it arrives at the final decision. As mentioned, this design is motivated by the observation that LLMs benefit significantly from CoT prompting in purely textual reasoning tasks, and extending this to multimodal inputs can help VLMs perform more structured reasoning. By training models to output both *thoughts* and *answers*, RL4VLM improves generalization to long-horizon reasoning and complex visual grounding.

We use $\mathcal{V}$ to denote the discrete and finite token space and $\mathcal{V}^m$ and $\mathcal{V}^n$ denotes the input and output text space where m and n are the maximum token length of input and output sequences respectively. The input state space is: $\mathcal{S} = \mathcal{O} \times \mathcal{V}^m$ where $\mathcal{O}$ is RGB images space. Then a VLM policy can be written as $\pi_\theta : \mathcal{O} \times \mathcal{V}^m \to \mathcal{V}^n$ and $v^{out} = \pi_\theta(o, v^{\text{in}})$ and it will be parsed to action. The probability of VLM outputting v^{out} becomes $\pi_\theta(v^{\text{out}}|\pi_\theta(o, v^{\text{in}})) \in [0, 1]$ given input consisting of image o and prompt v^{in}.

action parsing function: Because we want the probability of action to take $a \in \mathcal{A}$ we need to extract it from the $\pi_\theta(v^{\text{out}}|\pi_\theta(o, v^{\text{in}}))$. The paper provides an action parsing function f specifically to deal with times when the VLM's output v^{out} is not in the action space $\mathcal{A}$. The f is defined as follow:

$$f(v^{\text{out}}) = \begin{cases} a, & \text{if "action" } : a \in v^{\text{out}}, \\ \text{Unif}(\mathcal{A}), & \text{otherwise.} \end{cases}$$

The output of the VLM is in the following format:

"thoughts": "I am solving task T, given the current state I should do this a ", "action": "a"

Estimating action probabilities of VLM policies: The goal is to calculate $\log \pi_\theta(a_t, v_t^{\text{in}})$ but what the VLM gives is the $\log \pi_\theta(v_t^{\text{out}}|o_t, v_t^{\text{in}}) = \log \frac{P(o_t, v_t^{\text{in}}, v_t^{\text{out}})}{P(o_t, v_t^{\text{in}})}$

$$= \log[\frac{P(o_t, v_t^{in}, v_{[:1]})}{P(o_t, v_t^{in})} \cdot \frac{P(o_t, v_t^{in}, v_{[:2]})}{P(o_t, v_t^{in}, v_{[:1]})} \cdots \frac{P(o_t, v_t^{in}, v_{[:n]})}{P(o_t, v_t^{in}, v_{[:n-1]})}]$$

$$= \sum_{i=1}^{n} \log(\frac{P(o_t, v_t^{in}, v_{[:i]})}{P(o_t, v_t^{in}, v_{[:i-1]})}).$$

However, the above way of calculating the probability of actions is naive as it counts the *thought* section of the LLM response as the *action* as well and will be greatly influenced by the CoT tokens. In order to mitigate this problem, the authors suggest adding a factor $\lambda \in [0, 1]$ to deal with it:

$$\log \pi_\theta(a_t|o_t, v_t^{in}) \leftarrow \lambda \log \pi_\theta(v_t^{tht}|a_t, v_t^{in}) + \log \pi_\theta(v_t^{act}|o_t, v_t^{in}, v_t^{tht}).$$

The authors recommend $\lambda \in [0.2, 0.5]$.

The RL4VLM training pipeline consists of three stages:

1. **Data collection**: First trajectories are collected using GPT-4 CoT reasoning and answers.
2. **Supervised fine-tuning (SFT)**: The model is then trained on collected dataset where responses include both explicit reasoning steps and final answers. This stage initializes the model's ability to produce coherent CoT reasoning grounded in visual and textual inputs and in correct format.
3. **Reinforcement learning (RL)**: After SFT, the model is further refined using reinforcement learning, with environment rewards and the selected action only. The model parameters are updated using PPO, optimizing for average return.

This hybrid training strategy allows RL4VLM to leverage both the stability of supervised learning and the adaptability of reinforcement learning. The result is a vision-language model that can *think before acting*, yielding more accurate and explainable responses in multimodal decision-making tasks.

3.4.3 Policy extraction

A crucial design choice in language-model-based decision making is *how to extract a policy from an LLM or VLM*. Two main strategies have emerged in recent works: (i) GLAM-style (also used by ReACT) policy extraction, and (ii) RL4VLM-style policy extraction.

In the GLAM framework (Carta et al., 2023) similar to ReACT framework, the policy is elicited directly by formulating action selection as a next-token prediction task. Given a textual prompt that encodes the environment state x (e.g., observations, history, and goal instruction), the model is asked: *"give your next action"* The LLM then samples or scores possible action tokens from its vocabulary.

Formally, the GLAM-style policy can be written as:

$$\pi_\theta(a \mid x) = \frac{\exp\big(f_\theta(x, a)\big)}{\sum_{a' \in \mathcal{A}} \exp\big(f_\theta(x, a')\big)} \tag{3}$$

and

$$f_\theta(x, a) = \prod_{i=0}^{|a|} f_\theta(a_i | x, a_{<i})$$

where:

- x is the prompt encoding the environment state,
- $a \in \mathcal{A}$ is a candidate action token,
- $f_\theta(x, a)$ is the logit or score assigned by the LLM to action token a following context and previous action tokens x.

This formulation treats the LLM as a policy network directly over discrete action tokens, analogous to standard autoregressive decoding.

By contrast, RL4VLM (Zhai et al., 2024) uses free prompting and asks the model explicitly about the next action. This allows the addition of *thoughts* (reasoning traces) between the prompt and the final action, enabling a two-step policy extraction process. Using a prompt, first a free generation is sampled and then mapped to the possible actions using an action parsing function. The probability of the parsed action is the sum of probabilities of each action token generated conditioned on the prompt and previous action tokens generated.

1. The model is prompted with current state, history, and goal asking for the next action.
2. The sampled generation is parsed to one of possible actions using a parsing function.
3. The probability of the parsed action is the sum of conditional probabilities of generated tokens.

The corresponding policy is defined as:

$$\pi_\theta(a \mid x) = \sum_z \pi_\theta(z \mid x)\, \pi_\theta(a \mid x, z) \tag{4}$$

where:

- z is a latent reasoning trace (textual chain-of-thought),
- $\pi_\theta(z \mid x)$ is the probability of generating reasoning z given state x,
- $\pi_\theta(a \mid x, z)$ is the probability of selecting action a given both x and z.

While this approach allows the policy to leverage the full expressive power of language modeling (including reasoning chains, explanations, and flexible action descriptions), it introduces a fundamental challenge: the effective exploration space is no longer $|\mathcal{A}|$, but instead

$$|\mathcal{Y}| \approx V^n, \tag{5}$$

where V is the vocabulary size and n is the length of the generated sequence. Thus, the search space grows combinatorially with token length, making exploration far more complex. This trade-off highlights the central tension in RL4VLM-style policy elicitation: improved expressivity and reasoning capacity come at the cost of dramatically larger search spaces. However, while the search space is vast, and due to the strong priors of LLMs, the exploration remains limited. We dig deeper into this issue in next sections.

Thus, GLAM-style policy extraction views the LLM as a direct policy over actions via next-token prediction, while RL4VLM-style policy extraction treats this as direct prompting. GLAM-style policy extraction increases the computational complexity by prompting the model for each of the possible action, and on the other hand the RL4VLM-style policy extraction increases the exploration space.

4 Analysis of VIPER

In this section we dive deep into VIPER and do a failure analysis.

Aspect	GLAM-style policy extraction	RL4VLM-style policy extraction
Formulation	Action is predicted directly as the best continuation of the prompt.	Action is predicted after generating an explicit reasoning trace (chain-of-thought).
Policy definition	$\pi_\theta(a \mid x) = \frac{\exp(f_\theta(x,a))}{\sum_{a'} \exp(f_\theta(x,a'))}$	$\pi_\theta(a \mid x) = \sum_z \pi_\theta(z \mid x)\, \pi_\theta(a \mid x, z)$
Intermediate reasoning	None (direct mapping from prompt to action).	Explicit reasoning sequence z is generated before the action.
Interpretability	Limited: only final action probabilities are available.	Higher: reasoning traces expose decision-making process.
Training	Typically supervised or RL fine-tuning on prompt $\rightarrow$ action pairs.	Supervised fine-tuning on prompt $\rightarrow$ (thought, action) pairs, followed by RL refinement.
Advantages	Simpler; efficient; well-aligned with language modeling setup.	More interpretable; supports complex reasoning and long-horizon decision making.
Limitations	Less explainable; may struggle on tasks requiring structured reasoning.	Computationally heavier; quality of reasoning traces is critical.

Table 1: Comparison of GLAM-style vs. RL4VLM-style policy extraction methods.

4.1 ALFWORLD environment

To evaluate language-based sequential decision making, many works, including VIPER and ReAct, make use of the **ALFWorld** environment (Shridhar et al., 2020). ALFWorld is a text-based simulator derived from the *ALFRED* benchmark (Shridhar et al., 2020), which was originally designed for embodied agents operating in 3D visual environments (AI2-THOR). By abstracting the perception layer into textual descriptions, ALFWorld provides a controllable, language-centric setting for research on interactive reasoning, planning, and action execution.

ALFWorld bridges *natural language instructions* with *embodied actions*. At each step, the agent observes a textual description of the scene (objects, locations, relations) and must issue commands (e.g., `Open fridge`, `Go to microwave`) in order to achieve the instructed goal. This design allows research on decision making without requiring high-dimensional visual processing, while remaining aligned with realistic embodied tasks.

Tasks in ALFWorld mirror common household activities. They are categorized into six types:

- **Pick & Place**: move an object from one receptacle to another (e.g., *"Put the apple in the fridge"*).
- **Examine**: navigate to and examine a specific object (e.g., *"Find the book and look at it"*).
- **Clean**: locate a dirty object and clean it (e.g., *"Clean the mug in the sink"*).
- **Heat**: find an object and heat it (e.g., *"Heat the potato in the microwave"*).
- **Cool**: find an object and cool it (e.g., *"Cool the soda in the fridge"*).
- **Put Two**: place two specified objects in a common receptacle (e.g., *"Put the apple and the banana in the bowl"*).

An example episode begins with the instruction *"Put a clean plate in the microwave."* The agent must:

1. Navigate to a cabinet, open it, and pick up a plate.
2. Go to the sink and clean the plate.
3. Finally, navigate to the microwave and place the plate inside.

Success requires combining navigation, object interaction, and reasoning about the temporal structure of tasks.

ALFWorld has become a standard benchmark for evaluating LLM- and VLM-based decision making because:

- It supports long-horizon, instruction-driven tasks.
- It disentangles reasoning/planning from low-level visual perception.
- It allows for reproducible experiments across hundreds of procedurally generated environments.

Thus, ALFWorld provides an effective testbed for studying how LLMs and VLMs can act as policies, planners, or reasoning modules in sequential decision making.

4.2 Reasoning vs vision

Based on the capabilities of LLM and VLM where the earlier is good at textual understanding and reasoning and the later is good at visual recognition and textual description of images, VIPER intends to use the textual description as an intermediary tool for better decision making.

In order to improve such combined system it is necessary to understand the failure of each component and separate each failure case and attribute the failure either to the visual understanding or the reasoning capability of the LLM.

Since the goal in VIPER is explainable and separated perception and reasoning through the use of a frozen VLM and fine-tuned LLM respectively, we explore to answer the following core questions.

- 1. How often the perception module (frozen VLM) fails in VIPER?
- 2. How often the reasoning module (fine-tuned LLM) fails in VIPER?
- 3. What percentage of the failure cases in VIPER can be attributed to failure in perception ONLY?
- 4. What percentage of the failure cases in VIPER can be attributed to failure to reasoning ONLY?
- 5. In failed trajectories, how often does the VLM fail (independently from the LLM, as in that case we cannot conclude for the VLM)?

An additional question that gets answered as byproduct of analysis: What percentage of the failure cases in VIPER can be attributed to both perception and reasoning?

Among the raised questions, questions 3 and 4 are of more importance and trickier to answer. Questions 1 and 2 are independent of failure and success and can be answered easily.

Doing such a work manually is difficult and time consuming. Two methods of automatic pipeline failure analysis and manual failure analysis based on subset sampling is conducted and each method is explained and the corresponding result is provided in below.

4.3 An automatic pipeline

We first explain an automatic pipeline for failure analysis of VIPER.

4.3.1 The overall strategy

The general strategy in the automatic pipeline is to find when the VLM fails, and if we can determine and quantify that then we can easily decide if a trajectory's success and failure can be attributed to perception and/or reasoning. This makes sense based on the architecture as well, as the reasoning is based on perception and comes after it.

We assume that we can determine VLM-description failure with high accuracy.

If VLM has failed based on our detection, then we face two cases: either the trajectory is a failure or a success.

Case 1: failed trajectory: this case is expected as the model (LLM) does not know the state or has a wrong perception. It would be safe to say that the reasoning has also gone wrong in this case, otherwise if the LLM could reason correctly then it would have recovered from VLM failure and led to success. Then the failure can be attributed to both perception failure followed by reasoning failure. This answers to question 5.

Case 2: successful trajectory: In this case one can argue that the LLM was robust enough to recover from the VLM failure so the success can be attributed to the LLM robustness.

If the VLM has succeeded based on our detection, again we face two cases:

Case 1: failed trajectory - This case can be attributed to reasoning failure because perception was correct then the reasoning must have failed. We can conclude that this failure is due to reasoning failure. This answers to question 4.

Case 2: Successful trajectory - this case is again straightforward. VLM was correct so the reasoning cannot be wrong as the wrong reasoning does not lead to success.

Based on the above cases, question #3 remains unanswered. However, based on the argument that reasoning is built on top of perception, then the failed perception alone can not lead to failure. We analytically found the answer to question 3 and it is 0.

4.3.2 Methodology

How to detect VLM failure? The whole analysis is based on detecting VLM failure. In addition, this is a technical and tricky problem to deal with and hard to quantify precisely. We approximate it through a process by using some tools. The process and tools used are detailed step by step with an example.

But first we define trajectory and step as they are the basic units analyzed.

Trajectory: is a sequence of finite steps towards a specific goal that has either failed or succeeded. Length of failed trajectory is equal to maximum length and the length of successful trajectories are equal or less than the maximum length. A trajectory involves a set of trajectory objects. For example in a trajectory with the goal statement "put a cup on table", the trajectory objects are cup and table. We extract trajectory objects from the trajectory goal statement using LLAMA 1B prompting.

Step: a pair of state and the next action inside a specific trajectory. A step includes an image, visible objects in image, text description (ground truth), VLM description, objects mentioned in VLM-description. For a step we define a set of critical objects as the intersection of visible objects in an image and its trajectory objects. For example, if a cup is visible in image then it is critical if it is a trajectory object as well.

Method: A step-by-step description of the method is provided in below:

- Step 1 (Extract trajectory objects): objects mentioned in the trajectory goal are extracted by prompting an LLM. This is done once per trajectory.
- Step 2 (Extract visible objects in the step's image of scene): object visible in the current image is ground truth given by environment per step. This is done once per step.
- Step 3 (Extract step's critical objects): critical objects for a step are the intersection of step's visible objects and trajectory objects. This is done once per step.
- Step 4 (Extract VLM objects): objects mentioned in VLM description are extracted by prompting an LLM. This is done once per step.
- Step 5 (calculate matching score - ratio of critical objects mentioned by VLM): this ratio is calculated by cosine similarity between the encodings of critical objects and objects mentioned in VLM description and normalized to get the ratio. We call this ratio the step matching score. The encoder used is lightweight and well justified for this use case. (more information about the encoder: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2). This is done once per step.

4.3.3 Results and analysis

The results based on this analysis is provided in Table 2. A threshold of 0.6 is used for deciding between high and low VLM score. Based on this matrix we can identify the following four cases akin to the ones mentioned in Section 4.3.1:

Table 3: Benchmarking cosine similariy with different encoders. Metric: accuracy for a given threshold, prediction: element with highest score >= threshold

Encoder/Threshold	0.4	0.45	0.5	0.55	0.6
BERT	0.44	0.44	0.55	0.55	0.55
MiniBERT	0.44	0.44	0.44	0.44	0.44
T5	0.44	0.44	0.44	0.44	0.44
RoBERTa	0.44	0.44	0.55	0.55	0.55
MiniLM	0.44	0.55	0.55	0.55	0.44

Table 2: Trajectory success vs VLM success matrix.

Status	High VLM score	Low VLM score
Success	34.55	30
Failure	23.67	11.77

Case 1: A successful trajectory with high VLM score is an ideal case where the vision module has done a good job and the reasoning has also succeeded.

Case 2: A successful trajectory with a low VLM score shows the situations where the LLM has been robust and despite that the vision module has not been able to identify the object, the LLM still has chosen the correct action.

Case 3: A failed trajectory with high VLM score denotes the reasoning failure, where VLM has identified the object but the LLM has not been able to decide correctly.

Case 4: A failed trajectory with low VLM score is a failure as expected where the VLM fails and then the reasoning fails as well.

The described automatic method relies heavily on cosine similarity, so it is important to quantify the ability of cosine similarity for the terminologies and object names used in ALFWorld. It is also important for finding a good encoder. Since there are not too many objects in the AlfWorld environment, we select 8 testing samples including an object and a list of synonyms where only 1 of them is correct then we benchmark multiple encoders with different threshold to see if the correct synonym can be correctly identified by the cosine similarity and a specific encoder. The result for this benchmarking is presented in Table 3.

As it can be seen in the table, all of the encoders have the same performance and the performance of all the encoders are low. This result also demonstrates the necessity of a manual analysis of the performance of the model and dataset.

4.4 Manual analysis

Based on the insights from the automatic analysis pipeline, and for better understanding of the data, a manual sampling based analysis was performed.

Based on the analysis of the dataset, the following key steps in the process were identified:

- Object visible: The object corresponding to the task is visible in the image.
- VLM mentions: The VLM mentions the object after object visibility.
- Object picked: The LLM decides to pick the object.
- Object transformed: The LLM decides to transform the object as requested in the task description (such as heat, cool, and clean).
- Object transported: The LLM decides to transport the transformed object to the designated place in the task description.

From the available task types, 4 task types of Pick, Clean, Heat, and Pick2 were selected, and out of their corresponding trajectories, 6 trajectories were selected, including 3 successful and 3 failures per task type. In total, 102 trajectories were studied and analyzed, covering 25% of the total trajectories.

From the 5 identified steps necessary for a successful trajectory mentioned above, the first 2 determine the success and failure of the VLM and vision module, and the later 3 determine the success and failure of the reasoning module. A tree based manual algorithm can be designed as follows to determine if in a trajectory the vision has failed, or reasoning has failed, or both have failed.

VLM fails if it cannot mention the object before it is picked up by LLM and LLM fails if it does not move towards achieving the goal. If LLM is taking correct action but action is not done due to environment issues, we count it as success.

The result of this manual analysis is provided in the Table 4.

Table 4: Manual analysis result.

Status	VLM Failure	LLM failure	Both succeeded	total
Successful trajectory	38.46	0	61.53	100
Failure trajectory	27.27	54.54	18.18	100

A more detailed result based on each step of a normal trajectory is also presented in Table 5.

Table 5: Manual analysis result for each step.

Status	VLM sees	VLM mentions	Object picked	Object transformed	Object transported
Successful trajectory	69.2	46.15	100	100	100
Failure trajectory	58.3	33.3	33.3	0	8.3

4.5 Ablation experiment

In order to further analyze the role of each of the reasoning and vision components, it is necessary to understand if one of the modules does not exist, how much it will affect the performance of the system. More specifically, the reasoning module is the main component of the system, but the vision module can be replaced or removed by a fixed minimal state description. In order to quantify the effect of the vision module an experiment was conducted to remove the vision module, and provide the reasoning module with a fixed non-informative state description. The result of this experiment is provided in Table 6.

Table 6: Manual analysis result for each step.

Task/metric	Success rate		Episode length		Size of testset
	VLM True	VLM False	VLM True	VLM False	
Pick	0.74	0.77	9.34	9.33	35
Examine	0.61	0.76	13.8	9.0	13
Clean	0.74	0.66	13.05	11.05	27
Heat	0.87	0.81	9.57	13.46	16
Cool	0.64	0.60	11.9	14.0	25
Pick2	0.45	0.45	13.8	16.9	24
Average	0.67	0.67	11.91	12.29	

As can be seen in this table, removing the vision module does not affect the results hugely, which shows that the LLM mostly ignores what the VLM says as a description of the state. Further, because the description of the state is only coming through the VLM, it means that the LLM ignores the image description and decides mostly based on the set of possible actions.

5 Exploration

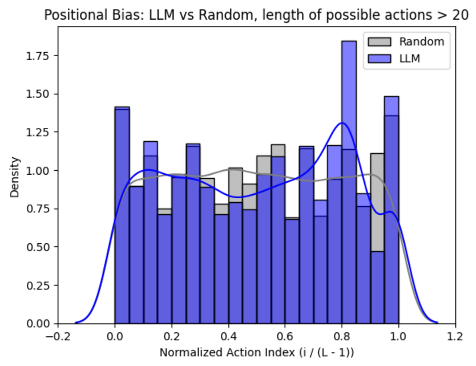
Exploration is one of the most important issues in RL, and in the following section, we dive deep into exploration in RL for LLM/VLMs.

5.1 Language models priors

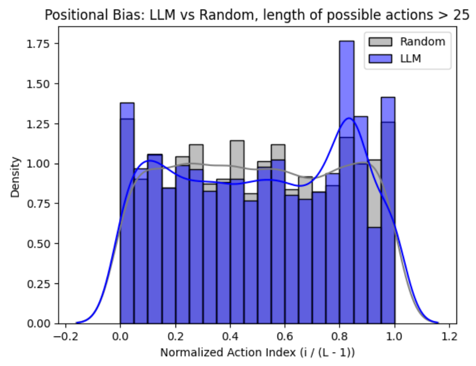
One of the unique challenges in applying large language models (LLMs) to sequential decision-making environments is that they are not trained from scratch in the target domain. Instead, they bring with them strong *priors* from pretraining on massive static text corpora. While these priors provide useful linguistic and commonsense knowledge, they also introduce systematic biases that affect exploration during reinforcement learning.

When deployed in interactive settings such as ALFWorld, LLMs do not begin with a uniform distribution over possible actions. Instead, their action preferences are heavily shaped by pretraining. For example, when presented with multiple-choice questions (MCQs), previous studies (Pezeshkpour and Hruschka, 2023) have shown that LLMs disproportionately select certain answer options (e.g., "A" or the first option), regardless of semantic content. These biases reveal that the model's probability distribution is strongly influenced by positional or frequency priors inherited from pretraining.

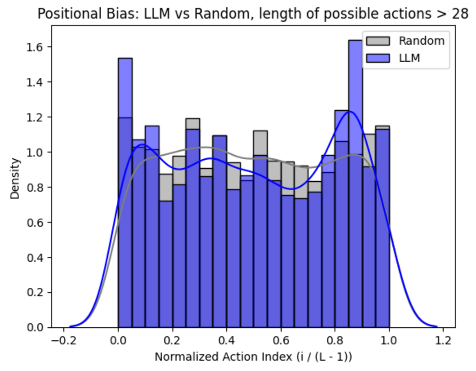
We have observed similar effects when prompting LLaMA in the ALFWorld environment. Specifically, given a list of available textual actions, the model exhibits a preference for actions at the *beginning* or *end* of the list, rather than distributing probability mass more evenly across all valid actions. This bias persists even when the correct action is positioned elsewhere, demonstrating that the exploration process is significantly constrained by pretrained token-level distributions. The figure 1 shows the frequency distribution of actions selected by LLaMA 1B averaged over multiple steps, where the positions of possible actions are randomly permuted. In order to show that this happens across different models and sizes, we conducted the same experiment with Mistral 7B. The result is shown in Figure 2.



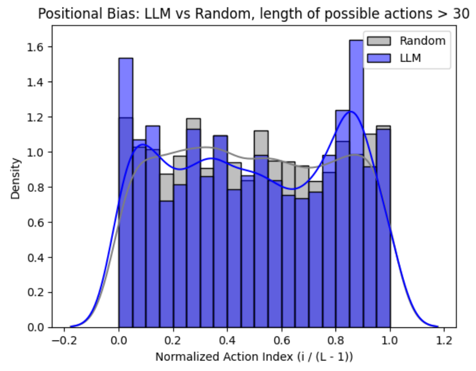
(a) Position bias in possible actions with more than 20 actions.



(b) Position bias in possible actions with more than 25 actions.



(c) Position bias in possible actions with more than 28 actions.



(d) Position bias in possible actions with more than 30 actions.

Figure 1: Positional bias analysis for Llama 1B. As the number of possible actions increase the bias also increases and the model gets less access to the actions trapped in the middle.

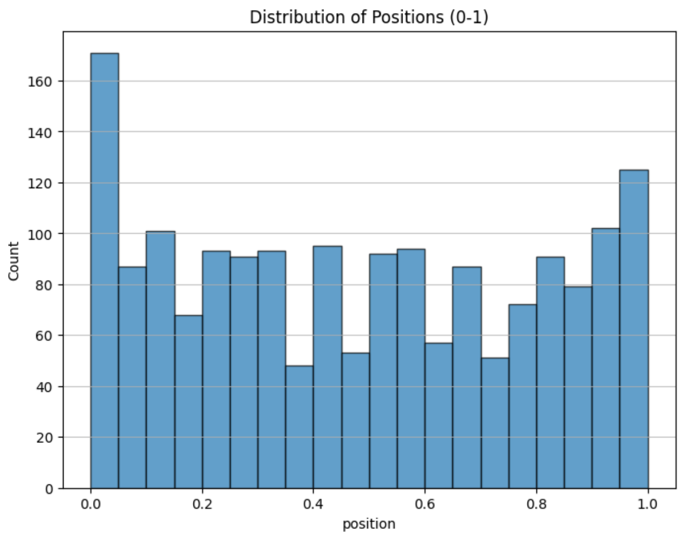


Figure 2: Positional bias analysis for Mistral 7B. Result is consistant with Llama 1B.

The LLM's bias does not remain limited to just position of the possible actions, but is also needed to be studied in actions content. In order to study this, we conduct another study in the ALFWorld environment to see where in a house environment the LLM model searches for specific objects. The result of this study is provided in Figure 3 and as it can be seen the LLM tends to search very specific places for an object. As an example, the LLM searches for the object book mostly in the sidetable, but it does not look for it in the shelf, desk, or countertop at all. While it is good to know that a book is highly likely to be on sidetable but, if it is on the desk, the model fails to find it.

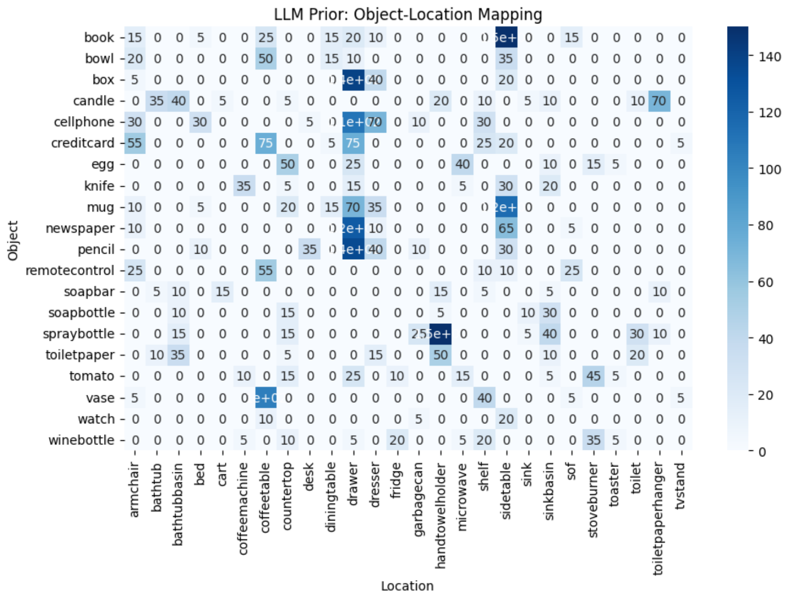


Figure 3: Content bias analysis for Llama 1B showing where the model looks for an object.

These priors have a direct impact during reinforcement learning fine-tuning with algorithms such as PPO. Instead of exploring the action space effectively, the model tends to repeatedly sample from its biased distribution. This reduces exploration diversity, slows down the discovery of successful trajectories, and can even lead to suboptimal convergence. In other words, the pretrained prior creates an implicit *exploration bottleneck*: while PPO aims to balance exploration and exploitation via policy updates, the initial bias of the LLM strongly skews the exploration dynamics.

While language model priors encode valuable knowledge, they also hinder unbiased exploration in new interactive environments. Addressing these issues requires designing training regimes that counteract or correct for pretrained biases, for example, through modified sampling strategies, entropy regularization, or explicit counter-biasing during RL fine-tuning.

5.2 Curiosity based exploration

Most prior work on curiosity-driven exploration in RL has focused on encouraging the agent to visit novel *states* in the environment (Pathak et al., 2017; Burda et al., 2018). These approaches typically reward the agent for reaching states that are difficult to predict with its internal forward model, thereby promoting exploration beyond familiar regions of the state space.

In the context of language model agents, however, the primary challenge lies not only in reaching novel states but also in diversifying the *actions* that the model selects. To this end, we consider an *action-wise curiosity* mechanism, which encourages the model to take new actions and to produce novel *patterns of actions*, rather than repeating previously preferred ones.

We identify two key ways in which LLM priors constrain exploration:

1. **Action repetition.** The model tends to repeatedly choose the same action in the same state, ignoring alternative actions that may lead to new states. This limits the diversity of exploration at the local level.
2. **Sequential pattern repetition.** Even when individual action frequencies are balanced, the model often reproduces the same patterns of sequential actions across episodes. This phenomenon mirrors *habit formation* in cognitive science (Graybiel, 2008), where repeated sequences become entrenched behaviors.

The first factor can be quantified by examining the frequency distribution of actions. If the action distribution is highly peaked (i.e., the model strongly prefers a few actions), exploration is limited. Conversely, a more uniform action frequency indicates greater exploration potential. Encouraging entropy in the action distribution is therefore one way to promote action-level curiosity.

The second factor highlights that action frequency alone is insufficient for effective exploration. Consider a maze navigation example: the sequence

$$[\text{left}, \text{right}, \text{forward}, \text{left}, \text{backward}]$$

may lead to a completely different state than the sequence

$$[\text{right}, \text{forward}, \text{left}, \text{backward}, \text{left}],$$

even though the frequency of each action is identical. This demonstrates that exploration must also account for the *ordering and structure* of actions, not just their aggregate counts. Promoting diversity in action sequences, therefore, plays a crucial role in overcoming habitual patterns and expanding the exploration space.

Curiosity in LLM-based agents should be extended beyond state novelty to explicitly target both *action frequency diversity* and *sequence diversity*. Addressing these two factors is essential for breaking out of pretrained action biases and enabling effective exploration during RL fine-tuning.

5.3 Intrinsic rewards

Intrinsic rewards are additional reward signals that are not derived from the external environment objective, but are instead designed to encourage behaviors such as exploration, novelty-seeking, or information gain (Oudeyer et al., 2007; Pathak et al., 2017; Burda et al., 2018). They act as internal motivation for the agent, complementing extrinsic rewards from the environment.

A large body of prior work has shown that intrinsic rewards are effective in promoting exploration. Examples include prediction-error based curiosity (Pathak et al., 2017), where the agent is rewarded for reaching states that are difficult for its forward dynamics model to predict, or information-gain based approaches (Houthooft et al., 2016), where rewards are tied to reductions in uncertainty about the environment. These methods encourage agents to leave familiar states and visit unseen parts of the environment.

Building on our action-wise perspective, we propose two complementary forms of intrinsic rewards tailored to language-model agents:

1. **Action frequency reward:** To counteract the tendency of LLMs to repeat the same actions in a given state, we define an intrinsic reward inversely proportional to the square root of frequency with which an action has been taken. This encourages the model to try actions it has rarely or never attempted before, thereby improving action-level exploration.
2. **Action sequence novelty reward:** Beyond single actions, we incentivize the agent to generate novel sequences of actions. This addresses the issue of habitual action patterns, where the agent repeatedly executes the same sequence across episodes. By rewarding the discovery of new action patterns, the model is encouraged to diversify its exploration strategies.

The action frequency reward for an action a can be defined as follows:

$$\mathcal{R}_{\text{act}}(a) = \alpha \frac{1}{\sqrt{N(a)}}$$

where $N(a)$ is the number of times action a is taken until now.

While rewarding action frequency is relatively straightforward, detecting and quantifying the novelty of action sequences is significantly more challenging. The core difficulty lies in defining when a sequence is “new”:

- The action space is combinatorial, so the number of possible sequences grows exponentially with sequence length.
- Small variations in a sequence (e.g., swapping the order of two actions) may or may not correspond to a genuinely novel trajectory in the environment.
- Storing and comparing all previously seen action sequences becomes computationally infeasible for long horizons.

Thus, designing effective measures of sequence-level novelty requires balancing tractability with expressiveness. Potential solutions may involve approximate similarity metrics (e.g., edit distance or subsequence matching), compressed representations of trajectories, or probabilistic novelty estimation. However, each of these approaches introduces its own trade-offs between accuracy and computational efficiency.

5.4 Temporal predictor

As discussed in the previous section, detecting whether an action sequence is “novel” is a challenging problem due to the combinatorial explosion of possible trajectories. To address this, we propose the use of an auxiliary model, which we call the *temporal predictor*, that provides a learned measure of sequence predictability. The central idea is that action patterns that are highly predictable (according to this model) are likely to have been encountered frequently, whereas unpredictable sequences correspond to novel patterns.

The temporal predictor is trained on data collected from the PPO replay buffer. Its task is to predict the next action a_{t+1} given the sequence of past actions and the goal description:

$$\hat{a}_{t+1} = f_\phi(a_{1:t}, g),$$

where f_ϕ denotes the temporal predictor parameterized by ϕ, $a_{1:t}$ is the history of actions up to time t, and g is the task goal. The predictor can be instantiated as a lightweight recurrent model or transformer that models action dependencies over time. In our experiments, we use the transformer based model T5 (Raffel et al., 2020).

The temporal predictor is trained using a supervised prediction loss, such as cross-entropy between the predicted action distribution and the actual next action observed in the PPO buffer:

$$\mathcal{L}_{\text{TP}}(\phi) = -\sum_t \log p_\phi(a_{t+1} \mid a_{1:t}, g).$$

Algorithm 1 PPO with Intrinsic Rewards for Action-wise Exploration

```
1: Input: Policy $\pi_\theta$, temporal predictor $f_\phi$, replay buffer $\mathcal{D}$, coefficients $\alpha, \beta, \gamma$
2: for each PPO iteration do
3:     Collect trajectories $\tau = \{(s_t, a_t, r_t^{\text{ext}})\}_{t=1}^{T}$ using $\pi_\theta$
4:     for each $(s_t, a_t)$ in $\tau$ do
5:         Update action count $N(a_t) \leftarrow N(a_t) + 1$
6:         Compute action frequency reward:
               $$\mathcal{R}_t^{\text{act}} = \alpha \cdot \frac{1}{\sqrt{N(a_t)}}$$
7:         Compute temporal predictor loss:
               $$\mathcal{L}_{\text{TP}}(\phi) = -\log p_\phi(a_{t+1} \mid a_{1:t}, g)$$
8:         Define pattern novelty reward:
               $$\mathcal{R}_t^{\text{pat}} = \beta \cdot \mathcal{L}_{\text{TP}}(\phi)$$
9:         Combine intrinsic rewards:
               $$\mathcal{R}_t^{\text{novelty}} = \mathcal{R}_t^{\text{act}} + \mathcal{R}_t^{\text{pat}}$$
10:        Define final reward:
               $$\mathcal{R}_t = \mathcal{R}_t^{\text{ext}} + \gamma \cdot \mathcal{R}_t^{\text{novelty}}$$
11:    end for
12:    Store $\tau$ with $\mathcal{R}_t$ in buffer $\mathcal{D}$
13:    Update policy $\pi_\theta$ via PPO objective with rewards $\mathcal{R}_t$
14:    Update temporal predictor $f_\phi$ using supervised loss $\mathcal{L}_{\text{TP}}$
15: end for
```

We propose to use the prediction loss of the temporal predictor as an intrinsic reward signal for the PPO agent. Intuitively, if the temporal predictor finds an action sequence easy to predict, this implies that the sequence is already well-represented in the agent's trajectory data, and thus not novel. Conversely, if the prediction error is high, the agent is executing an unfamiliar or unexpected action pattern. The intrinsic reward is therefore defined as:

$$\mathcal{R}_t^{\text{pat}} = \alpha \mathcal{L}_f(\phi; a_{1:t}, g).$$

Where α is the coefficient, and in our experiments, we set α to 0.0096 so that it won't overwhelm the true environment reward. The novelty reward $\mathcal{R}_t^{\text{novelty}}$ can be defined as follows:

$$\mathcal{R}_t^{\text{novelty}} = \mathcal{R}_t^{\text{act}} + \mathcal{R}_t^{\text{pat}}$$

Combining it with the extrinsic task reward $\mathcal{R}_t^{\text{ext}}$ during PPO training we get the final reward:

$$\mathcal{R}_t = \mathcal{R}_t^{\text{ext}} + \gamma \mathcal{R}_t^{\text{novelty}},$$

where γ is a scaling factor that controls the relative importance of exploration versus exploitation.

The temporal predictor provides a practical and scalable solution to quantify action-sequence novelty without requiring explicit enumeration or storage of trajectories. By rewarding the agent for surprising the predictor, we encourage exploration of diverse action patterns that may lead to the discovery of better policies. The algorithm 1 shows the detailed process.

5.5 Experiments, Results, and Analysis

We evaluate our proposed intrinsic rewards on the `pick` task of the ALFWorld environment using the LLaMA-1B model as the backbone policy. Three training configurations were compared:

1. **Config 1, PPO baseline:** PPO trained with only extrinsic rewards.

2. **Config 2, PPO + action novelty:** PPO augmented with the action frequency intrinsic reward $\mathcal{R}^{\text{act}}$.
3. **Config 3, PPO + action novelty + pattern novelty:** PPO augmented with both $\mathcal{R}^{\text{act}}$ and the temporal predictor-based novelty reward $\mathcal{R}^{\text{pat}}$.

Each configuration was trained under 4 random seeds, and evaluated by success rate achieved during training. Mean performance and variance across seeds are reported.

The results of the three configurations are summarized in Figure 4. While the curves exhibit different tendencies, the confidence intervals across the three methods overlap substantially, making it difficult to draw statistically strong conclusions. Nonetheless, several trends are noteworthy:

- The baseline PPO agent tends to exploit early, often converging to repeating a subset of actions.
- PPO with only the action novelty reward (Config 2) explores more aggressively in the early stages because the success rate is significantly lower, but this often delays convergence and leads to lower mean performance.
- PPO with both action and pattern novelty (Config 3) achieves slightly higher average success than the other two, but the effect is not statistically significant due to overlapping variances.

These inconclusive results highlight the inherent difficulty of evaluating exploration strategies in RL. While novelty-based rewards do alter the exploration dynamics, promoting broader action coverage and more diverse action sequences, the performance improvements are not yet robust. The overlap in variances suggests that larger-scale experiments may be needed to fully validate the benefits of action-wise and pattern-wise exploration.

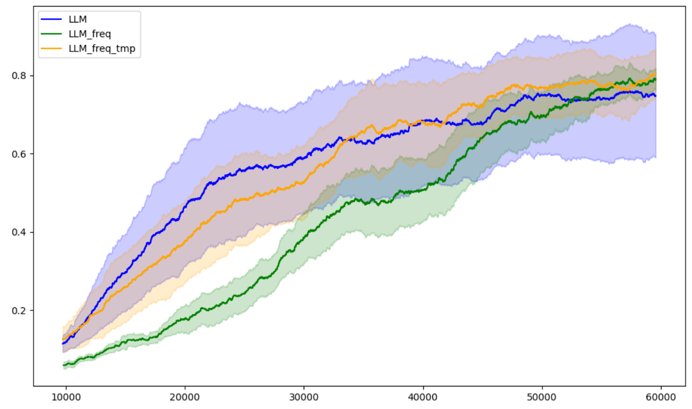


Figure 4: Results comparing three configurations based on curiosity reward. **LLM**: Config 1, **LLM_freq**: Config 2, **LLM_freq_tmp**: Config 3

6 Exploration based on model priors

In the previous section, we discussed how intrinsic rewards can be used to encourage exploration in the PPO setting. However, we could not find any evidence that the proposed curiosity-based reward significantly outperforms the simple PPO algorithm. For further exploration, we needed to test and study the proposed curiosity rewards in a different and simpler environment and with RL4VLM-style policy elicitation. Moving to RL4VLM-style policy elicitation, we encounter a much more fundamental issue that complicates exploration even more.

In RL4VLM-style policy extraction, the VLM is prompted with the current state, history, and goal, and then the model generates the next action in free-form text. The generated text is parsed by an action-parsing function and mapped to an environment action. However, this approach often leads to the following failure mode:

1. The VLM generates an action string.

2. The environment rejects the action because it is invalid or cannot be executed in the current state.
3. Since the environment state and the prompt remain unchanged, the VLM regenerates the exact same invalid action.
4. This loop continues until the episode times out or the maximum step budget is exhausted.

This phenomenon illustrates the model's tendency to "get stuck" in invalid or repetitive actions, severely limiting its ability to explore.

The core challenge of RL4VLM-style elicitation is that the model generates actions in the token space of its vocabulary. This introduces two key difficulties:

- **Exploration is indirect:** Instead of sampling from a well-defined set of possible environment actions, the model samples from the much larger space of all possible token sequences.
- **Exploding search space:** The number of possible action strings grows exponentially with sequence length, approximately as $|\mathcal{V}|^n$ where $|\mathcal{V}|$ is the vocabulary size and n is the sequence length.

One might attempt to encourage diversity in generation by increasing the sampling temperature. However, this creates its own set of problems:

- Higher temperature increases randomness, but also pushes the model toward producing irrelevant or malformed outputs that cannot be parsed into valid actions.
- Lower temperature preserves format consistency but makes the model even more deterministic, reinforcing the tendency to repeat the same invalid action.

Thus, temperature scaling alone is insufficient to address the exploration bottleneck.

As a result of these issues, RL4VLM-style elicitation prevents the agent from ever reaching novel states. Since exploration in RL fundamentally depends on being able to try new actions, this setting imposes severe limitations compared to GLAM-style policy extraction, where actions are sampled directly from a discrete action set.

6.1 Action-scoring as a bridge to model priors

As discussed in the previous section, the key limitation of RL4VLM-style policy elicitation is that it requires sampling directly from the token space, which is both inefficient and prone to invalid actions. By contrast, GLAM-style elicitation, where the model is prompted to select the best action among a set of candidates by teacher forcing, is more practical: it naturally yields a probability distribution over the discrete action set, enabling sampling and exploration. However, this raises a deeper question: how can we ensure that we are actually leveraging the priors embedded in the language model?

The priors of large language models are not straightforwardly expressed in the log-probabilities of next-token prediction. Interpreting the probability of a specific token as a meaningful prior is problematic, as this signal is highly sensitive to tokenization artifacts and generation biases. Instead, the true priors of the model are encoded in the token space itself, in the structured patterns of how the model evaluates and reasons about possible actions in a given state.

To access these priors, we propose prompting the model to *score* each candidate action with respect to its usefulness in the current state and the given goal. This formulation has several advantages:

- It constrains the model to reason within the discrete action set of the environment, avoiding the exponential token search problem.
- It allows direct sampling from a probability distribution constructed from the model's action scores.
- It makes the model's preferences more interpretable, as scores are assigned at the level of actions rather than arbitrary tokens.

We further enhance this method by asking the model to first generate a short CoT reasoning before outputting action scores. This encourages the model to explicitly weigh the pros and cons of each candidate.

Let $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$ denote the set of available environment actions at state s. We prompt the model with the state, and goal, and request both (i) a reasoning trace and (ii) scores for each action. The model produces a set of scalar values $\{s_1, s_2, \ldots, s_K\}$, one for each action with $s_i \in [0, S]$ for some arbitrary maximum score S. These are then normalized into a probability distribution:

$$\pi_{\text{score}}(a_i \mid s) = \frac{\exp(s_i)}{\sum_{j=1}^{K} \exp(s_j)}.$$

This defines a *behavioral policy* π_{score} that can be sampled during training to promote exploration consistent with model priors. In parallel, we can maintain a *training policy* π_{act} obtained via a simpler "direct action" prompt, in which the model outputs a single best action. The two policies can then be combined within an RL framework, where π_{score} provides exploration guidance and π_{act} is optimized through PPO updates.

Although both the behavioral policy π_{score} and the training policy π_{act} are instantiated from the same underlying LLM/VLM with shared parameters θ, they are elicited via distinct prompting strategies:

- **Policy prompt A (action scoring):** The model is asked to assign a scalar score to each possible action $a_i \in \mathcal{A}$, where scores lie in a discrete range (e.g., 1 to 5). The output is then converted into a normalized distribution over actions:

  $$\pi_{\text{score}}(a_i \mid s, g; \theta) = \frac{\exp(\tau \cdot \text{score}_\theta(a_i \mid \text{Prompt B}(s, g)))}{\sum_{j=1}^{K} \exp(\tau \cdot \text{score}_\theta(a_j \mid \text{Prompt B}(s, g)))},$$

  where τ is a temperature parameter that controls the sharpness of the distribution.
- **Policy prompt B (direct action):** The model is asked to select a single best action from the admissible set $\mathcal{A} = \{a_1, a_2, \ldots, a_K\}$, producing an action-conditioned JSON output. This defines the *training policy*:

  $$\pi_{\text{act}}(a \mid s, g; \theta) = \Pr_\theta(a \mid \text{Prompt A}(s, g)),$$

  where s denotes the current state and g the task goal.

Action scoring prompt (policy π_{score}):

Input:

```
You are an expert 2D game player in a grid-based environment. The
    environment is .. [ENVIRONMENT DESCRIPTION].
You are observing the image of the current state, and your goal is ..
    [GOAL].
Please evaluate each action based on the current observation and assign
    a score between 1 (very bad) and 5 (very good).
Return your answer as a valid JSON object in the following format:
{
  "thoughts": "Describe briefly the current state of the player as seen
    in the image.",
  "action_scores": {
    "Turn left": score_1,
    "Turn right": score_2,
    "Move forward": score_3,
    "Pick up": score_4,
    "Toggle": score_5
  }
}
```

Response:

```
{
  "thoughts": "I am currently at the center of the grid, facing the left
    side.
              The green goal square is located to the right of me.",
  "action_scores": {
    "Turn left": 1,
    "Turn right": 4,
    "Move forward": 1,
    "Pick up": 2,
    "Toggle": 1
  }
}
```

Direct action prompt (policy π_{act}):

Input:

```
You are an expert 2D game player in a grid-based environment. The
    environment is .. [ENVIRONMENT DESCRIPTION].
The goal is .. [GOAL].
At each step you can choose one of the actions
['Turn left', 'Turn right', 'Move forward', 'Pick up', 'Toggle'].
Return your answer as a valid JSON object in the following format:
{
  "thoughts": "Describe briefly the current state of the player as seen
    in the image.",
  "Action": "Your chosen action"
}
```

Response:

```
{
  "thoughts": "I am facing a closed door with the goal room visible
    behind it.",
  "Action": "Toggle"
}
```

In this setup, π_{score} serves only as a *behavioral policy* for sampling actions during training, since it allows structured exploration guided by model priors (the model's relative preferences across actions). Meanwhile, π_{act} is treated as the only *training policy* to be optimized via PPO, with gradient updates applied to θ based on task rewards. The use of a shared parameterization might ensure that improvements from PPO updates propagate to both prompting styles, but exploring this remains beyond the scope of this work.

This dual-prompt approach balances two desiderata:

1. *Exploration consistent with model priors:* By sampling from π_{score}, the agent explores actions that the model itself considers plausible or promising, rather than sampling arbitrary token continuations in the large vocabulary space.

2. *Exploitation via direct action selection:* By training π_{act} on the best single-action outputs, the agent learns to commit to effective behaviors for task completion.

Thus, the behavioral policy π_{score} can be seen as inducing a soft search distribution in action space, while the training policy π_{act} provides the deterministic decision-making interface used for actual control.

By moving from free-form action generation to action scoring, we align exploration more closely with the model's intrinsic priors while maintaining the efficiency and interpretability of discrete action sampling. This approach also provides a natural bridge between language model reasoning and RL policy optimization.

6.2 Experiments

In this section, we explain the experiments conducted using the π_{act} and π_{score} policies.

6.2.1 Experimental setup

We evaluate our approach in the `MiniGrid-DoorKey-6x6` environment, a commonly used benchmark for testing reasoning and exploration in grid-based settings. The environment consists of a 6x6 grid world where the agent must navigate, unlock a door, and reach the goal location. The action space contains five discrete actions:

$$\mathcal{A} = \{\texttt{turn left}, \texttt{turn right}, \texttt{move forward}, \texttt{pick up}, \texttt{toggle}\}.$$

The agent receives a shaped reward of

$$R = 1 - 0.9 \cdot \frac{\texttt{step\_count}}{\texttt{max\_steps}},$$

with a maximum horizon of 250 steps. Episodes terminate either when the agent successfully reaches the goal or when the time limit is reached. Observations are provided in the form of rendered RGB images of the grid.

For the policy backbone, we employ **Qwen2.5-VL-3B**, a vision-language model capable of processing both the visual observations and textual prompts. A representative environment image is shown in Figure 5.

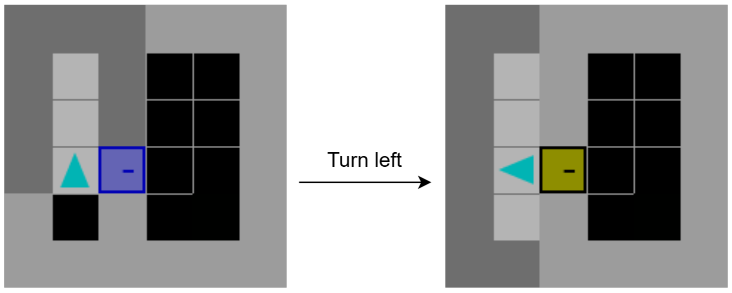


Figure 5: An example state, action, and next state from MiniGrid doorkey environment

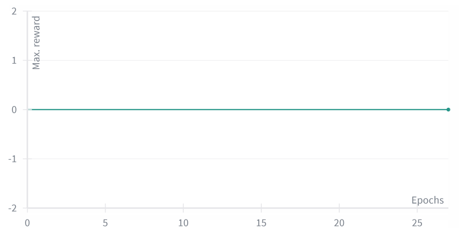
6.2.2 Simple PPO

We first test the feasibility of using the dual-policy setup described in the previous section, where π_{score} serves as the behavioral policy and π_{act} is treated as the training policy. In each rollout step, an action is sampled from $\pi_{\text{score}}(a \mid s, g)$, and the sampled action is concatenated to the `thoughts` field of the π_{score} output. This augmented representation is then used as the input context for π_{act}, which selects the final action and is updated via PPO.

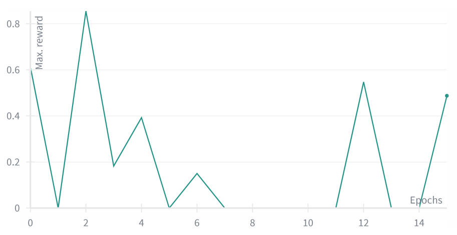
When using π_{act} directly as the behavioral policy during rollouts, the model quickly gets stuck in repetitive action loops, as discussed earlier. In contrast, with π_{score} guiding exploration, we were able to collect successful trajectories where the agent completes the task. The maximum reward achieved per iteration for pure PPO and using the proposed π_{score} as behavior policy is shown in the Figure 6.

This demonstrates that action-scoring prompts can indeed serve as a structured exploration mechanism that alleviates the limitations of direct action prompting.

An unexpected observation was that after a few PPO updates, the model began to switch the language of the `thoughts` section from English to Chinese (see Figure 7). We hypothesize that this arises from a mismatch between the two policies: while π_{score} outputs thought tokens jointly with action scores, the sampled action from π_{score} appended to the reasoning context generated by π_{act} is not fully aligned with each other. This mismatch and sometimes contradiction may encourage the model to drift into alternative decoding modes, such as changing the reasoning language, in order to reconcile the conflicting signals.

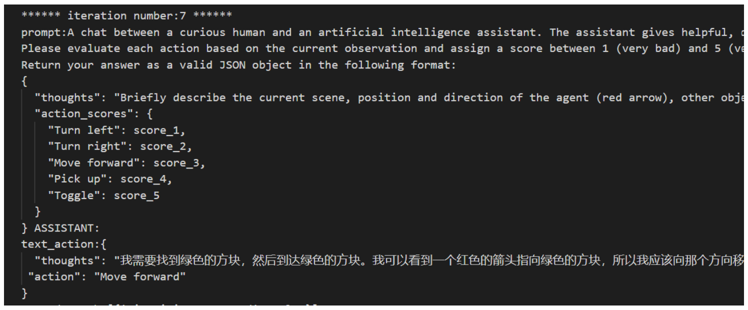


(a) The maximum reward achieved per iteration by pure PPO (π_{act} only)



(b) The maximum reward achieved per iteration by dual policy PPO (π_{score} as behavior policy, π_{act} as train policy).

Figure 6: Comparison between the maximum reward achieved during PPO iteration between pure PPO and dual policy PPO.



```
****** iteration number:7 ******
prompt:A chat between a curious human and an artificial intelligence assistant. The assistant gives helpful, c
Please evaluate each action based on the current observation and assign a score between 1 (very bad) and 5 (ve
Return your answer as a valid JSON object in the following format:
{
  "thoughts": "Briefly describe the current scene, position and direction of the agent (red arrow), other obje
  "action_scores": {
    "Turn left": score_1,
    "Turn right": score_2,
    "Move forward": score_3,
    "Pick up": score_4,
    "Toggle": score_5
  }
} ASSISTANT:
text_action:{
  "thoughts": "我需要找到绿色的方块，然后到达绿色的方块。我可以看到一个红色的箭头指向绿色的方块，所以我应该向那个方向移
 "action": "Move forward"
}
```

Figure 7: Unexpected behavior of language switching after a few PPO iterations using π_{score} as behavioral policy and π_{act} as train policy.

Despite the language drift, these preliminary experiments suggest that the dual-policy prompting scheme provides a viable mechanism for collecting successful trajectories, which would not be possible with direct action prompting alone. However, ensuring consistency between the behavioral and training policies remains an open challenge, particularly when augmenting reasoning traces with sampled actions.

6.2.3 Decayed Exploration

To address the issues observed in the simple PPO experiments, particularly the language switching problem, we experimented with a decayed exploration strategy. In this setup, the behavioral policy gradually shifts from the action-scoring policy π_{score} toward the direct action policy π_{act} over the course of training. This allows early-stage exploration guided by model priors, while progressively stabilizing the rollout distribution using the action prompt. The procedure is summarized in Algorithm 2.

This decayed approach successfully mitigated the language-switching problem encountered when π_{score} was used exclusively. However, a new failure mode emerged: the action field of the model output frequently deviated from the expected JSON format, making downstream parsing unreliable (see Figure 8. We attempted several modifications to further stabilize the training and outputs:

- Removing CoT tokens from prompts.
- Switching to the off-policy Advantage-Weighted Regression (AWR) (Peng et al., 2019) algorithm.

Neither intervention resolved the format inconsistencies.

Analysis of the generated responses suggested that the model's understanding of the environment was limited. Often, the model repeated text from the prompt rather than generating grounded reasoning about the current state. This indicates that environmental knowledge cannot easily emerge from pure reinforcement learning alone and would likely require a significant amount of computation and

Algorithm 2 Decayed Exploration with Dual Policies

```
1: Input: Dual policies $\pi_{\text{score}}, \pi_{\text{act}}$ (same model queried with different prompts); decay schedule
   $\epsilon(t) \in [0,1]$; steps per iteration $N$; replay buffer $\mathcal{D}$
2: for each PPO iteration $t = 1, \ldots, T$ do
3:     Set exploration probability $\epsilon \leftarrow \epsilon(t)$          (e.g. linear/exponential decay)
4:     Initialize empty rollout $\tau \leftarrow []$
5:     for step $i = 1, \ldots, N$ do
6:         Observe state $s$ and goal $g$
7:         Draw $b \sim \mathrm{Bernoulli}(\epsilon)$
8:         if $b = 1$ then                         ▷ use scoring policy for this step
9:             Query $\pi_{\text{score}}(\cdot \mid s, g)$ to obtain thoughts and action scores
10:            Convert scores to distribution and sample action:
                   $a \sim \pi_{\text{score}}(\cdot \mid s, g)$
11:        else                                    ▷ use direct action policy for this step
12:            Query $\pi_{\text{act}}(\cdot \mid s, g)$ (direct-action prompt) and sample:
                   $a \sim \pi_{\text{act}}(\cdot \mid s, g)$
13:        end if
14:        Execute $a$, observe extrinsic reward $r^{\text{ext}}$ and next state $s'$
15:        Append transition $(s, a, r^{\text{ext}}, s', g)$ to $\tau$
16:        $s \leftarrow s'$
17:    end for
18:    Compute returns $\{\hat{G}_i\}_{i=1}^{N}$ and advantages $\{\hat{A}_i\}_{i=1}^{N}$ from $\tau$ (e.g., GAE)
19:    Store $\tau$ in buffer $\mathcal{D}$
20:    Update policy parameters $\theta$ via PPO using transitions $(s, a, \hat{A})$ from $\tau$
21: end for
```

interaction to develop. Attempts to increase compute by generating multiple response samples in a GRPO-style approach also failed to improve performance.

Learning a successful policy via PPO depends heavily on an accurate value function, which itself requires extensive interaction with the environment. The combination of limited environmental understanding and the difficulty of value function estimation renders the learning problem particularly challenging for large vision-language models.

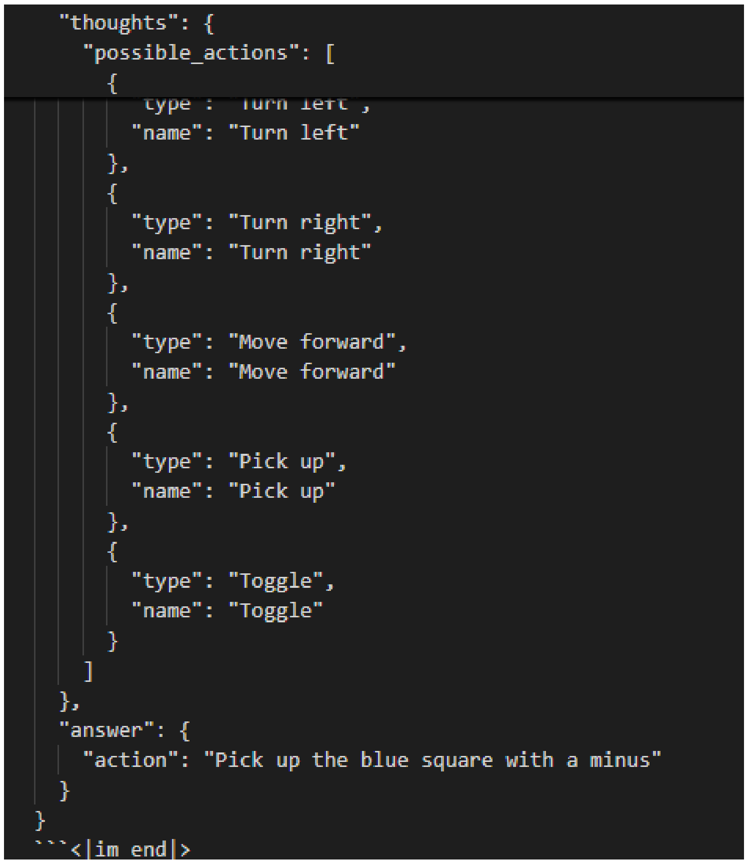
7 Leveraging a Local Utility Function for Action Evaluation

Nearly all existing RL approaches for LLMs and VLMs rely on a primary phase of supervised fine-tuning (SFT) on expert trajectories. For example:

- VIPER initializes its policy using expert data, and much of its performance comes from this SFT.
- RL4VLM performs SFT on trajectory rollouts generated by GPT-4.
- RLVMR (Zhang et al., 2025) similarly relies on SFT on 200 GPT-4-generated trajectories, along with an expert tagging model trained on GPT-4 data.

In our work, we attempted to break out of this dependence on expert knowledge, which is expensive to obtain and may not exist in many domains. However, as discussed in the previous section, our efforts were largely unsuccessful. Learning new environmental knowledge purely via RL appears to require immense amounts of interaction and trial-and-error, and the poor exploration capabilities of current LLMs and VLMs make this nearly impossible. While larger models might possess better priors for image understanding, the computational resources required are impractical for our setup.

Smaller models are often trained through distillation from larger models on a limited dataset to achieve competitive performance on benchmarks. This likely explains why they exhibit weaker



~~~
    "thoughts": {
      "possible_actions": [
        {
          "type": "Turn left",
          "name": "Turn left"
        },
        {
          "type": "Turn right",
          "name": "Turn right"
        },
        {
          "type": "Move forward",
          "name": "Move forward"
        },
        {
          "type": "Pick up",
          "name": "Pick up"
        },
        {
          "type": "Toggle",
          "name": "Toggle"
        }
      ]
    },
    "answer": {
      "action": "Pick up the blue square with a minus"
    }
  }
  ```<|im_end|>
~~~

Figure 8: The output gets out of format while decaying the exploratory π_{score} as behavioral policy.

reasoning and environment understanding capabilities, further constraining exploration and policy learning.

A reliable value function is essential for PPO and similar RL algorithms. However, learning a value model that depends solely on the current state can be insufficient, particularly in environments with complex state transitions and long horizons.

To alleviate this limitation, we propose a *local utility function* that predicts the usefulness of an action a_t with respect to its immediate effect in the environment. Formally, we define:

$$u(a_t) = f_\theta(s_t, a_t, s_{t+1}, g),$$

where s_t is the current state, s_{t+1} is the resulting state after executing a_t, g is the goal, and f_θ is a model that estimates how much a_t contributes to achieving g. Intuitively, $u(a_t)$ indicates whether the taken action was helpful or unhelpful toward goal completion.

Since states are represented as images, we suggest using a vision-language model (VLM) as the utility predictor. The utility can be operationalized as a next-token prediction task, leveraging the reasoning and knowledge capabilities of the VLM.

A similar concept has been explored in robotics by GVL (Generative Value Learning) (Ma et al., 2024), where shuffled video frames are fed to a VLM to quantify task progress. Our approach differs in two key ways:

1. We focus on *local utility* for individual actions rather than global task evaluation over entire video sequences.
2. Our VLM compares the *current state* s_t and *next state* s_{t+1} to assign a usefulness score to the action a_t, explicitly considering the goal g.

The VLM is prompted to reason about the effect of a taken action in the context of the current goal. It outputs a scalar utility reflecting whether a_t brought the agent closer to achieving g. This utility signal can then be integrated into RL as an auxiliary reward, potentially improving policy learning in the absence of expert trajectories.

7.1 Evaluating VLMs as Utility Functions

To assess whether smaller vision-language models (VLMs) such as QWEN2.5VL-3B can serve as effective utility functions, we conducted preliminary tests using collected trajectories. Each test involved providing the model with the *current state* s_t and the *next state* s_{t+1} after an action a_t, along with the goal g, and asking it to predict the utility $u(a_t)$.

As expected from previous observations, the smaller 3B model exhibits limited visual understanding. When presented with before-and-after images of an action, the model often fails to correctly assess the usefulness of a_t. A representative result is shown in Figure 9, highlighting the poor predictive performance.

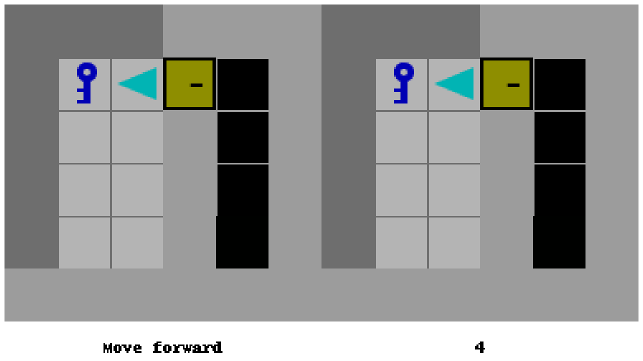


Figure 9: Zero-shot utility predicted by Qwen2.5VL-3B. The image on the left shows the previous state, the action taken is "move forward", the image on the right is the next state. The VLM gives a high score of 4 out of 5.

Repeating the same test using QWEN2.5VL-32B yields substantially better utility predictions (see Figure 10). While the larger model demonstrates improved reasoning and visual comprehension, working directly with it for RL is computationally prohibitive.

To leverage the 32B model's superior reasoning without incurring the computational cost, we propose a distillation approach:

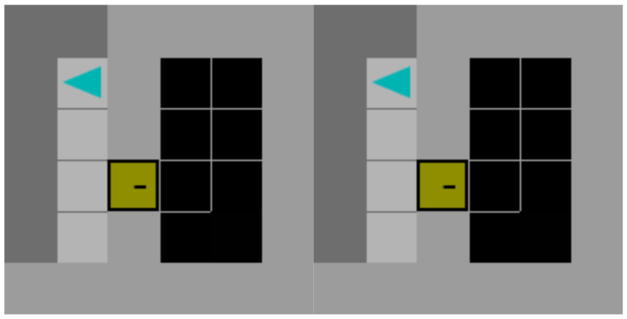
1. Collect trajectories using the 3B model and the action-scoring policy π_{score}, comprising approximately 9000 steps.
2. Label each tuple (s_t, a_t, s_{t+1}, g) with the utility $u(a_t)$ predicted by the 32B model. Since this is pure inference, it is computationally efficient.
3. Fine-tune the 3B model on these labeled steps.

This approach differs fundamentally from conventional supervised fine-tuning (SFT) used in other methods:

- No external expert model such as GPT-4 or human-labeled data is required.
- Trajectories are generated using the 3B model itself, meaning the only additional knowledge comes from the 32B model's improved reasoning and visual understanding.

7.2 Integrating the Utility Function into RL

Having obtained a utility function $u_\theta(s_t, a_t, s_{t+1}, g)$ via distillation, we explore two ways to incorporate it into RL.



Action: Pick up
Thoughts: The player's position and orientation remain unchanged between the two images. The yellow square with a minus sign, representing the closed door, is still present in the same location relative to the player. There are no visible changes indicating that the key was picked up or any other action occurred.
Score: 0

Figure 10: Zero-shot utility predicted by Qwen2.5VL-32B. The image on the left shows the previous state, the action taken is "pick up", the image on the right is the next state. The VLM gives a low score of 0 out of 5.

1. **Replacement of the value function:** Treat the utility function as a surrogate for the standard value model, directly predicting expected future returns conditioned on (s_t, a_t, g). This allows the policy to leverage action-level utility signals instead of relying solely on the potentially undertrained value function.
2. **Reward densification:** Keep the PPO value model unchanged, but augment the reward with the utility signal:
$$\mathcal{R}_t^{\text{total}} = \mathcal{R}_t^{\text{extrinsic}} + \lambda\, u_\theta(s_t, a_t, s_{t+1}, g),$$
where λ is a scaling coefficient. This provides denser, more informative feedback to the agent while retaining the original value estimation.

Both strategies aim to exploit the VLM's reasoning and visual understanding to guide policy learning: the first by replacing the value estimate with action-specific utility, and the second by densifying the reward signal to accelerate learning in sparse-reward environments.

7.3 Replacing the Value Function with the Utility Function

In conventional PPO, the advantage function at time t is defined as
$$A_t = G_t - V(s_t),$$
where G_t is the discounted sum of rewards (or the GAE-based return) and $V(s_t)$ is the value function trained via an auxiliary loss.

We propose to replace or scale the value function with the action-level utility, yielding a modified advantage:
$$\tilde{A}_t = G_t \cdot u_\theta(s_t, a_t, s_{t+1}, g),$$
where $u_\theta(s_t, a_t, s_{t+1}, g)$ is the normalized utility predicted by the VLM. Intuitively, this scales the return by the estimated usefulness of the taken action, providing denser feedback in sparse-reward environments.

In sparse-reward tasks, most trajectories initially fail, leading to $G_t \approx 0$. We explore two modalities to address this:

1. **Minimal reward for failed trajectories:** Assign a small baseline reward (e.g., $G_t = 0.1$) even to trajectories that do not reach the goal, ensuring the advantage signal is non-zero and allowing the utility function to influence learning across all trajectories.

2. **Success-only training:** Keep failed trajectories with $G_t = 0$, so that $\tilde{A}_t = 0$, and update the policy only on successful trajectories. This focuses learning on trajectories with meaningful outcomes but may reduce sample efficiency.

In standard PPO, the policy gradient is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_t\Big[\nabla_\theta \log \pi_\theta(a_t \mid s_t)\, A_t\Big],$$

where $A_t = G_t - V(s_t)$ is the advantage function.

When replacing $V(s_t)$ with a utility function $u_\theta(s_t, a_t, s_{t+1}, g)$ or scaling G_t by u_θ, the modified advantage becomes:

$$\tilde{A}_t = G_t \cdot u_\theta(s_t, a_t, s_{t+1}, g).$$

A key question is whether this transformation preserves the policy gradient direction, i.e., whether the optimal policy π^* remains unchanged. If $u_\theta(s_t, a_t, s_{t+1}, g)$ is a monotonic function of the true advantage A_t, then scaling by u_θ preserves the ranking of actions, and the policy gradient is only reweighted but not reversed. Formally, if

$$A_t > A'_t \implies u_\theta(s_t, a_t, s_{t+1}, g)\, G_t > u_\theta(s_t, a'_t, s'_{t+1}, g)\, G'_t,$$

then the gradient ascent still favors the same actions.

However, in sparse-reward environments where $G_t \approx 0$ for failed trajectories, $\tilde{A}_t \approx 0$ regardless of the utility value. This effectively removes the gradient contribution from those actions, potentially biasing the policy toward trajectories that succeed early.

Scaling by u_θ introduces a new source of variance in the gradient estimate, because u_θ is learned and may be noisy. Let $\hat{u}_t = u_\theta(s_t, a_t, s_{t+1}, g)$ with variance σ_u^2. Then the variance of the modified policy gradient becomes:

$$\mathrm{Var}[\nabla_\theta J(\theta)] \approx \mathrm{Var}[\hat{u}_t\, G_t\, \nabla_\theta \log \pi_\theta(a_t \mid s_t)].$$

In principle, if u_θ accurately predicts action usefulness, scaling G_t improves sample efficiency by concentrating gradient updates on high-utility actions.

7.4 Using the utility function for reward densification

A second way to incorporate the utility function is through *reward shaping*. The idea is to use $u_\theta(s_t, a_t, s_{t+1}, g)$ not to replace the value function but as an auxiliary signal that densifies the sparse environment reward.

Reward shaping must be applied carefully, since arbitrary modification of the reward can change the optimal policy. The potential-based reward shaping (PBRS) framework (Ng et al., 1999) provides a sufficient condition for policy invariance: the shaped reward

$$r'_t = r_t + \gamma\phi(s_{t+1}) - \phi(s_t)$$

preserves the set of optimal policies for any potential function $\phi : \mathcal{S} \to \mathbb{R}$.

We define the immediate utility:

$$u_t = f_\theta(s_t, a_t, s_{t+1}, g),$$

and the discounted utility-to-go:

$$U_t = \sum_{k=t}^{T} \gamma^{k-t} u_k.$$

Now define the potential function as:

$$\phi_t = -U_t.$$

Then the PBRS shaping term becomes:

$$F_t = \gamma\phi_{t+1} - \phi_t = \gamma(-U_{t+1}) - (-U_t) = U_t - \gamma U_{t+1}.$$

Since $U_t = u_t + \gamma U_{t+1}$, we obtain:

$$F_t = u_t.$$

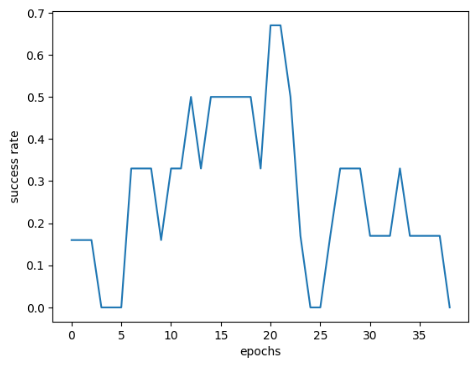
Thus, the shaping reward corresponds exactly to the immediate utility:

$$r'_t = r_t + \alpha\, u_t,$$

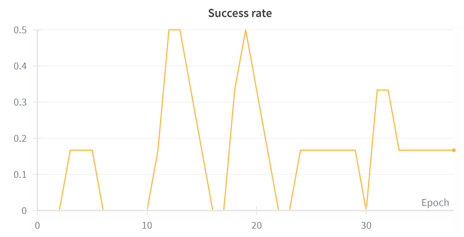
where α is a scaling coefficient. By PBRS theory, this transformation leaves the optimal policy invariant while providing dense intermediate feedback that can accelerate learning.

7.5 Experiments

We conduct experiments on the Minigrid doorkey environment under both reward densification and value function replacement settings by employing the dual policy framework with decayed exploration. The resulting success rate for value model replacement and reward densification is shown in the Figure 11.



(a) Success rate of the method based on value model replacement by utility function



(b) Success rate of policy based on reward densification by utility function.

Figure 11: Success rate of reward densification and value model replacement based on and by utility function.

Based on these results, the value function replacement by the utility function performs better, but still the training is not stable as towards the end of the training, where the trajectories are mostly sampled from π_{act}, the performance drops, which shows that the training has not been able to fully achieve its objectives. Instead, it seems that this algorithm optimizes the mixed policy, while that is not the objective at all.

8 Conclusion and future work

Despite our efforts and exploratory research, achieving stable reinforcement learning algorithm for LLM/VLM agents remains an open challenge. While we successfully addressed the exploration problem by introducing a dual-policy framework, learning effectively from trajectories sampled under a mixed behavioral policy has proven elusive. In particular, standard PPO fails to yield stable or convergent results when applied in this setting.

Nevertheless, our experiments demonstrate that dual-policy prompting is a promising direction. The scoring policy substantially improves exploration, enabling the collection of successful trajectories, while a gradual shift toward the direct action policy provides additional stability during training. This highlights the potential of structured exploration in overcoming the limitations of naive action prompting.

Looking forward, several research directions remain open. Ensuring tighter consistency between the action and scoring policies is essential, as is developing reinforcement learning algorithms capable of leveraging trajectories generated under a mixed or proxy behavioral policy. Furthermore, emerging techniques such as self-reflection offer a compelling opportunity for enhancing the self-improvement capabilities of LLM and VLM agents, and their integration with reinforcement learning presents a promising avenue for future exploration.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., et al. (2022). Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*.

Aissi, M. S., Grislain, C., Chetouani, M., Sigaud, O., Soulier, L., and Thome, N. (2025). Viper: Visual perception and explainable reasoning for sequential decision-making. *arXiv preprint arXiv:2503.15108*.

Alayrac, J.-B., Donahue, J., Luc, P., Miech, A., Barr, I., Hasson, Y., Lenc, K., Mensch, A., Millican, K., Reynolds, M., et al. (2022). Flamingo: a visual language model for few-shot learning. *Advances in neural information processing systems*, 35:23716–23736.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2018). Large-scale study of curiosity-driven learning. *arXiv preprint arXiv:1808.04355*.

Carta, T., Romac, C., Wolf, T., Lamprier, S., Sigaud, O., and Oudeyer, P.-Y. (2023). Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2023). Palm: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113.

Dong, Q., Li, L., Dai, D., Zheng, C., Ma, J., Li, R., Xia, H., Xu, J., Wu, Z., Liu, T., et al. (2022). A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Graybiel, A. M. (2008). Habits, rituals, and the evaluative brain. *Annu. Rev. Neurosci.*, 31(1):359–387.

Guez, A., Mirza, M., Gregor, K., Kabra, R., Racanière, S., Weber, T., Raposo, D., Santoro, A., Orseau, L., Eccles, T., et al. (2019). An investigation of model-free planning. In *International conference on machine learning*, pages 2464–2473. PMLR.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. (2016). Vime: Variational information maximizing exploration. *Advances in neural information processing systems*, 29.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pages 9118–9147. PMLR.

Jia, C., Yang, Y., Xia, Y., Chen, Y.-T., Parekh, Z., Pham, H., Le, Q., Sung, Y.-H., Li, Z., and Duerig, T. (2021). Scaling up visual and vision-language representation learning with noisy text supervision. In *International conference on machine learning*, pages 4904–4916. PMLR.

Jiang, A., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D., Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2024). Mistral 7b. arxiv 2023. *arXiv preprint arXiv:2310.06825*.

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Al Sallab, A. A., Yogamani, S., and Pérez, P. (2021). Deep reinforcement learning for autonomous driving: A survey. *IEEE transactions on intelligent transportation systems*, 23(6):4909–4926.

Kober, J., Bagnell, J. A., and Peters, J. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274.

Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. (2022). Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. (2017). Building machines that learn and think like people. *Behavioral and brain sciences*, 40:e253.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40.

Li, J., Li, D., Xiong, C., and Hoi, S. (2022). Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pages 12888–12900. PMLR.

Liu, H., Li, C., Wu, Q., and Lee, Y. J. (2023). Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916.

Ma, Y. J., Hejna, J., Fu, C., Shah, D., Liang, J., Xu, Z., Kirmani, S., Xu, P., Driess, D., Xiao, T., et al. (2024). Vision language models are in-context value learners. In *The Thirteenth International Conference on Learning Representations*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287. Citeseer.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Peng, X. B., Kumar, A., Zhang, G., and Levine, S. (2019). Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*.

Pezeshkpour, P. and Hruschka, E. (2023). Large language models sensitivity to the order of options in multiple-choice questions. *arXiv preprint arXiv:2308.11483*.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR.

Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y., Wu, Y., et al. (2024). Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Shridhar, M., Yuan, X., Côté, M.-A., Bisk, Y., Trischler, A., and Hausknecht, M. (2020). Alfworld: Aligning text and embodied environments for interactive learning. *arXiv preprint arXiv:2010.03768*.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Valmeekam, K., Marquez, M., Sreedharan, S., and Kambhampati, S. (2023). On the planning abilities of large language models-a critical investigation. *Advances in Neural Information Processing Systems*, 36:75993–76005.

Verma, M., Bhambri, S., and Kambhampati, S. (2024). On the brittle foundations of react prompting for agentic large language models. *arXiv preprint arXiv:2405.13966*.

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164.

Watkins, C. J. and Dayan, P. (1992). Q-learning. *Machine learning*, 8(3):279–292.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256.

Yang, Y., Zhou, T., Li, K., Tao, D., Li, L., Shen, L., He, X., Jiang, J., and Shi, Y. (2024). Embodied multi-modal agent trained by an llm from a parallel textworld. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 26275–26285.

Yang, Z., Li, L., Wang, J., Lin, K., Azarnasab, E., Ahmed, F., Liu, Z., Liu, C., Zeng, M., and Wang, L. (2023). Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Zhai, S., Bai, H., Lin, Z., Pan, J., Tong, P., Zhou, Y., Suhr, A., Xie, S., LeCun, Y., Ma, Y., et al. (2024). Fine-tuning large vision-language models as decision-making agents via reinforcement learning. *Advances in neural information processing systems*, 37:110935–110971.

Zhang, R., Zhang, B., Li, Y., Zhang, H., Sun, Z., Gan, Z., Yang, Y., Pang, R., and Yang, Y. (2024a). Improve vision language model chain-of-thought reasoning. *arXiv preprint arXiv:2410.16198*.

Zhang, Y., Bai, H., Zhang, R., Gu, J., Zhai, S., Susskind, J., and Jaitly, N. (2024b). How far are we from intelligent visual deductive reasoning? *arXiv preprint arXiv:2403.04732*.

Zhang, Z., Chen, Z., Li, M., Tu, Z., and Li, X. (2025). Rlvmr: Reinforcement learning with verifiable meta-reasoning rewards for robust long-horizon agents. *arXiv preprint arXiv:2507.22844*.

Zheng, C., Liu, S., Li, M., Chen, X.-H., Yu, B., Gao, C., Dang, K., Liu, Y., Men, R., Yang, A., et al. (2025). Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*.

Zhu, D., Chen, J., Shen, X., Li, X., and Elhoseiny, M. (2023). Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*.