Extended Abstract

Motivation Recent advancements in LLMs have demonstrated very high performance across many textual tasks with little to no task-specific fine-tuning. Reinforcement learning (RL) agents, on the other hand, are specialist agents that optimize for specific environments. Based on these observations, we were motivated to compare and contrast LLM-based, RL, and hybrid methods on Procgen environments. We decided to use Procgen due to its procedurally generated and continuous dynamic environments.

Method We explored five approaches across three Procgen environments. LLM-Actor directly queries Claude Sonnet 4 with each frame plus history to output actions. PPO and DQN serve as our RL baselines. Our hybrid approaches include LESR, which fuses ResNet50 visual features with LLM semantic embeddings to enhance RL state representation, and AWU, which switches between PPO and LLM based on PPO's action entropy. AWU Mode Switch extends this by using 10 consecutive LLM actions instead of single-timestep interventions when uncertainty is detected. Our novel contributions include the entropy-based switching mechanism and temporal consistency through sustained LLM control.

Implementation LLM-based agents used Claude Sonnet 4, while PPO and DQN employed Stable Baselines 3 implementations. LESR combined ResNet50 visual encoding with all-MiniLM-L6-v2 text embeddings through a three-layer CNN. AWU calculated PPO action entropy and switched to LLM when exceeding calibrated thresholds (70% for standard AWU, 40% for Mode Switch to encourage frequent switching). All experiments used consistent random seeds across Coinrun, Maze, and Caveflyer environments.

Results Our results reveal distinct performance patterns across environments and methods. In Coinrun, LLM-Actor achieved the highest average reward and greatest efficiency, outperforming PPO and AWU Mode Switch. In Maze, LLM-Actor again achieved the highest reward but with only 40% valid episodes, while PPO struggled with poor rewards despite 100% validity. In Caveflyer, PPO achieved the highest reward with full validity, while LLM-Actor reached 2.0 reward with 50% valid episodes. Notably, AWU Mode Switch consistently doubled the performance of standard AWU across environments, demonstrating the critical importance of sustained LLM intervention over single-timestep switching. DQN and LESR failed across all environments with zero or near-zero rewards, attributed to sparse reward challenges and insufficient training time, respectively. These results highlight that while LLMs excel in zero-shot planning for simpler tasks, hybrid strategies with temporal consistency best balance performance and efficiency across diverse procedurally generated environments.

Discussion Our study reveals fundamental trade-offs between generalist LLM reasoning and specialist RL optimization in interactive environments. While LLMs excel in zero-shot planning for simpler tasks like Coinrun, they struggle with fine-grained control and spatial reasoning in complex scenarios. The computational overhead of LLM queries (300x slower than PPO) raises questions about practical deployment, though this cost may be justified for high-value, low-frequency decision-making scenarios. Technical challenges included prompt engineering difficulties, low visual resolution constraints (64×64 pixels), and the need for careful entropy calibration in hybrid methods. Our attempts to use faster local models like LLaVa revealed a core challenge: efficient models lack the reasoning capabilities that make LLM-based planning attractive, while capable models remain computationally prohibitive for extensive training.

Conclusion This work provides a comprehensive comparison of zero-shot LLM planning versus traditional RL across procedurally generated environments with continuous dynamics. Our key finding is that effective LLM-RL integration requires temporal consistency as sustained LLM guidance through mode switching successfully combines strategic reasoning with learned behaviors, while brief interventions prove insufficient. The superior performance of AWU Mode Switch demonstrates that hybrid approaches represent a path forward, allocating control between specialized and generalist components based on situations. As LLM capabilities and computational efficiency continue to improve, our hybrid methodologies serve as stepping stones toward autonomous agents that combine the broad applicability of LLMs with the precision of reinforcement learning.

Comparative Study of Zero-Shot LLM Planning vs. Reinforcement Learning in Procgen

James Cheng
Department of Computer Science
Stanford University
jzcheng@stanford.edu

Andy Ouyang
Department of Computer Science
Stanford University
andyou@stanford.edu

Nishikar Paruchuri
Department of Computer Science
Stanford University
nishikar@stanford.edu

Abstract

As large language models (LLMs) demonstrate increasingly sophisticated reasoning capabilities, questions arise about their effectiveness compared to traditional reinforcement learning (RL) methods in interactive decision-making environments. We present a comprehensive comparative study of zero-shot LLM planning versus classical RL algorithms across three procedurally generated Procgen environments with continuous dynamics. Our evaluation includes LLM-Actor, traditional RL baselines (PPO and DQN), and novel hybrid approaches that dynamically combine both paradigms. We introduce Ask When Uncertain (AWU), an entropy-based method that switches between RL and LLM control based on policy confidence, AWU Mode Switch, which maintains temporal consistency through sustained LLM intervention, and LESR, which extracts visual and semantic information using LLMs and CNNs for RL agent training. Our results reveal distinct performance patterns: LLM-Actor achieves superior rewards in simpler environments like Coinrun while struggling with prompt compliance in complex scenarios. LESR struggles to perform due to requiring excessive compute during training. Critically, we find that effective LLM-RL integration requires temporal consistency—AWU Mode Switch consistently doubles standard AWU performance by using 10 consecutive LLM actions rather than single-timestep interventions. While LLMs demonstrate 300x computational overhead compared to PPO, our hybrid approaches represent a practical path forward, dynamically allocating control between specialized and generalist components. These findings provide actionable insights for LLM-RL integration in complex interactive domains and highlight the importance of sustained reasoning in hybrid autonomous systems.

1 Introduction

Recent advancements in large language models (LLMs) have demonstrated human and superhuman-level performance across diverse textual tasks and complex reasoning challenges. These models, trained on vast corpora of internet text encompassing everything from academic papers to programming code, have proven remarkably effective at solving novel problems with little to no task-specific fine-tuning. This zero-shot capability makes LLMs compelling examples of generalist agents that can adapt their broad knowledge to new domains without explicit retraining. Notable examples include

GPT-4's performance on standardized tests, Claude's reasoning abilities across scientific domains, and the emergence of sophisticated planning capabilities in models like GPT-4o and Gemini.

In contrast, reinforcement learning agents represent the paradigm of specialist optimization, where policies are meticulously trained for specific environments or objectives through extensive interaction and reward feedback. While RL has achieved remarkable successes in domains like game playing (AlphaGo, Chess), robotics control, and resource management, these agents typically exhibit brittle generalization and struggle when deployed beyond their training distribution. The fundamental tension between the sample efficiency and domain expertise of specialist RL agents versus the broad applicability but computational overhead of generalist LLM agents raises important questions about optimal approaches for interactive decision-making.

Given the rapidly expanding capabilities of LLMs in multimodal reasoning and their demonstrated success in planning tasks, it becomes increasingly important to understand how well they perform in interactive reinforcement learning environments that require temporal decision-making, spatial reasoning, and continuous control, especially when applied in zero-shot settings without domain-specific training. This question is particularly relevant as LLMs begin to be deployed in robotics, autonomous systems, and interactive applications where real-time decision-making under uncertainty is crucial. However, despite growing interest in LLM-based planning, there remains a significant gap in systematic empirical comparisons between LLM-based planning and traditional RL methods across challenging, visually complex environments.

In this paper, we address this gap by conducting a comprehensive comparative study of LLM-based planning versus classical RL algorithms (PPO and DQN) across multiple Procgen environments. Procgen provides an ideal testbed for this comparison as it features procedurally generated mini-games with continuous dynamics, high-dimensional visual observations, and discrete action spaces that are amenable to LLM processing while still requiring robust generalization capabilities. Our research investigates three key questions: (1) How do zero-shot LLM agents compare to trained RL agents in terms of task performance and efficiency across environments of varying complexity? (2) What are the fundamental trade-offs between generalist reasoning capabilities and specialist optimization in interactive decision-making scenarios? (3) Can hybrid approaches effectively combine the complementary strengths of LLMs and RL agents to achieve superior performance?

2 Related Work

2.1 Reinforcement Learning in Procedurally-Generated Benchmarks

Generalization has been a long-standing challenge in deep RL. Early benchmarks provided narrow fixed levels and allowed agents to overfit, which motivated new procedurally-generated suites that explicitly test generalization. For example, the Sonic the Hedgehog benchmark (Nichol et al. (2018)) and OpenAI's Coinrun (Cobbe et al. (2019)) demonstrated that agents easily memorize a few fixed levels but fail to generalize to unseen levels. Similarly, Cobbe et al. (2020) introduced the Procgen benchmark (16 diverse game-like tasks) with separate training and test level distributions, to measure both sample efficiency and generalization. On Procgen, standard RL algorithms can achieve high training scores but often exhibit large train-test gaps. Cobbe et al. (2020) found that PPO was relatively stable across environments, whereas a DQN-based algorithm sometimes outperformed PPO on individual games but was highly unstable overall. These results align with other studies like GVG-AI (Perez-Liebana et al. (2019)) and Obstacle Tower (Juliani et al. (2019)), showing "strong overfitting" of deep agents with limited level sets. To address this, researchers have explored architectural and algorithmic solutions. For instance, augmenting encoders with recurrence or stacking frames, using 3D convolutions, and adding dropout have all been shown to improve Procgen generalization by capturing richer spatiotemporal structure or reducing overfitting (Jesson and Jiang (2024)). In summary, while classical RL algorithms on procedural benchmarks can learn robust policies, they can still struggle to generalize without additional inductive bias or diversity.

2.2 Large Language Models for Planning and Decision Making

Large pretrained LMs have emerged as capable generalist planners and reasoners across domains. Although trained on text, these models broadly encode broad world knowledge that can be leveraged for goal-directed tasks. Recent work has exploited LLMs to generate multi-step action plans via

chain-of-thought reasoning of trajectory generation. For example, Yao et al. (2023) propose the ReAct framework, in which an LLM alternates between generating reasoning traces and task-specific actions to solve complex problems. On interactive benchmarks, ReAct significantly outperforms imitation or RL baselines by using the LLM to break tasks into subgoals. Other works demonstrate that combining LLMs with vision can help agents develop their planning skills. For instance, the RT-2 model fine-tunes a large vision-transformer on both web-scale data and robotic trajectories, enabling it to interpret novel object attributes and perform multi-stage reasoning in response to commands (Brohan et al. (2023)). Likewise, SayCan uses an LLM to propose sequences of high-level skills for a robot (Ahn et al. (2022)). These examples illustrate that LLMs, even without task-specific RL training, can plan over long horizons using semantic commonsense.

2.3 Hybrid RL-LLM Approaches

Motivated by the limitations of pure RL and the generality of LLMs, recent work has developed hybrid methods that combine RL policies with LLM guidance. One hybrid approach enhances the state representation using LLMs: in LLM-Empowered State Representation (LESR), an LLM is prompted to produce task-relevant semantic features or descriptors for each raw state, and these embeddings are appended to the RL agent's input (Wang et al. (2024)). Wang et al. (2024) show that LESR dramatically improves sample efficiency and outperforms strong baselines, helping the value network to learn more quickly. Another method, When2Ask, explicitly trains an RL agent to decide when to query an LLM for help (Hu et al. (2024)). Hu et al. (2024) formulate the problem as a Markov decision process where the agent learns a policy to decide when to solicit a high-level instruction from an LLM and when to act autonomously, balancing query cost and task success. Experiments on MiniGrid show that When2Ask can solve problems with only a few LLM queries, dramatically reducing unnecessary interactions while still achieving target goals.

2.4 Comparisons and Our Contributions

While prior work has explored applications of LLMs in planning tasks, there remains a significant gap in comprehensive, direct comparisons between LLM-based planning, traditional RL methods, and hybrid LLM/RL methods across diverse environments. Most existing comparative studies have been limited in scope, focusing on simplified discrete environments or single-task evaluations. For example, Park et al. (2025) investigated LLM applications in the MiniGrid-Empty-5x5-v0 environment, demonstrating that LLM-guided agents achieved higher success rates than PPO or DQN agents, but this evaluation was restricted to a single, relatively simple grid-world environment with discrete dynamics and clear symbolic structure.

Our work addresses these limitations by providing systematic side-by-side comparisons of pure LLM planning, traditional RL methods (PPO and DQN), and hybrid approaches across multiple Procgen environments, which feature continuous dynamics, high-dimensional visual observations, and procedural generation that requires genuine generalization rather than memorization. We introduce two novel hybrid approaches (inspired by, but significantly deviating from When2Ask and LESR): an entropy-based uncertainty method that dynamically switches between RL and LLM control based on the agent's confidence, and a mode-switching variant that maintains temporal consistency in decision-making. Our hybrid models focus on trying to create lightweight combinations of RL and LLM models to create an agent with situational strengths of both models. Unlike previous studies that relied on discrete grid-worlds, our evaluation on Procgen's visually complex and procedurally-generated environments provides a more challenging and realistic testbed for understanding when and why different approaches succeed or fail, contributing actionable insights for LLM-RL integration in complex interactive domains.

3 Method

We explored multiple different approaches for solving Procgen environments, including LLM-Actor, PPO, DQN, LLM-Empowered State Representation (LESR), and Ask When Uncertain (AWU).

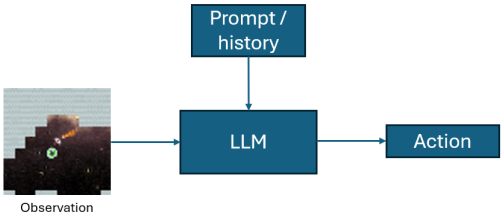


Figure 1: LLM-Actor steps. The LLM takes in an observation, a prompt (which includes the past three frames), and outputs a single action to take for that frame.

3.1 LLM-Actor

For our LLM-Actor agent (Figure 1), we fed each frame, along with a history of the past two frames, of the environment to the LLM (Claude Sonnet 4 on Amazon Bedrock), and asked it to respond with a single action ID to execute in the environment. We experimented with other multimodal models such as GPT-4o and BLIP-2 on HuggingFace, but decided to use Claude due to the high costs of GPT-4o and the poor capabilities of BLIP-2. Initially, we only fed one frame without history to the LLM, but we believed that adding some history would allow the LLM to better triangulate its position and dynamics in the environment. We also provided a description of the environment and the available actions, and instructed the LLM to choose a single action ID. In addition, for the Maze and Caveflyer environments, it would often get stuck, so we included in the prompt for the agent to try and get unstuck if it believes it is stuck (i.e. last three frames are identical). See the prompt in the appendix for a sample prompt for the Maze environment.

3.2 Baseline: Proximal Policy Optimization (PPO)

We implemented PPO as our primary on-policy reinforcement learning baseline due to its proven stability and effectiveness in visual control tasks. PPO addresses the key challenge of policy gradient methods by optimizing a clipped surrogate objective function that prevents overly large policy updates: $L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right]$, where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between new and old policies, $\hat{A}_t$ is the advantage estimate, and ϵ is the clipping parameter.

Our PPO agent employs a three-layer convolutional neural network architecture designed for processing the 64×64×3 RGB observations from Procgen environments. The convolutional encoder extracts hierarchical visual features and feeds them into separate policy and value function heads, enabling simultaneous policy optimization and value function learning. We employ Generalized Advantage Estimation (GAE) to reduce variance in policy gradients and include an entropy bonus to encourage exploration. With this combination of architectural design and training methodology, we expect to enable the PPO agent to generalize across the varied visual scenarios presented by procedural generation.

Hyperparameters: We trained PPO with rollout collection of 256 steps per environment, 4 optimization epochs per batch with batch size 1024, learning rate of 5×10^{-4}, discount factor $\gamma = 0.999$, GAE parameter $\lambda = 0.95$, clipping parameter $\epsilon = 0.2$, and entropy coefficient of 0.01. Training was conducted across 64 parallel environments for 4 million total timesteps, 16 environments for 4 million timesteps, and 32 environments per 5 million timesteps for the Coinrun, Maze, and Caveflyer environments, respectively.

3.3 Baseline: Deep Q-Network (DQN)

We implemented DQN as our off-policy reinforcement learning baseline to provide a comparison with value-based methods. DQN addresses the instability issues of traditional Q-learning in deep networks by learning a Q-function $Q(s, a; \theta)$ that estimates expected cumulative reward through minimizing the temporal difference error: $L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}} \left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$, where $\mathcal{D}$ is the replay buffer and θ^- represents target network parameters. The algorithm's stability comes

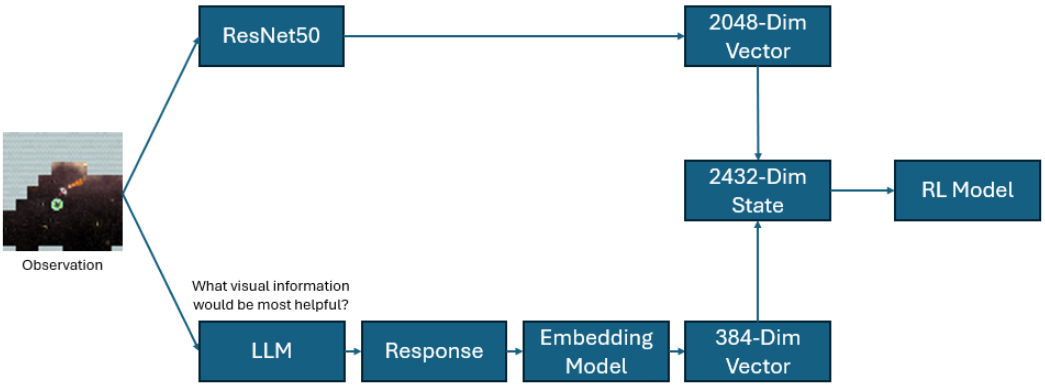


Figure 2: LLM-Empowered State Representation Steps. The observation is processed through two parallel streams: one visual (ResNet50) and one semantic (LLM). The resulting 2048-dim visual vector and 384-dim semantic embedding are concatenated to form a 2432-dim augmented state used for RL.

from experience replay, which breaks temporal correlations by storing and randomly sampling past transitions, and target networks, which provide consistent learning targets.

Our DQN implementation uses the same convolutional architecture as PPO, with the final output feeding into fully connected layers that produce Q-values for each discrete action. The training process maintains a replay buffer storing transitions from past interactions and employs ϵ-greedy exploration with a decaying schedule. While DQN's off-policy learning and experience replay are theoretically attractive for procedurally-generated environments, the method struggles with the visual complexity and sparse rewards characteristic of Procgen.

Hyperparameters: We trained DQN with learning rate 1×10^{-4}, discount factor $\gamma = 0.999$, replay buffer capacity of 100,000 transitions, and ϵ-greedy exploration decaying from 1.0 to 0.1. Learning commenced after 1000 initial exploration steps, with training every 4 environment steps using 3 gradient descent steps per update. Target network parameters were updated every 1000 training steps for 10 million total timesteps.

3.4 Hybrid: LLM-Empowered State Representation (LESR)

LESR enhances an RL agent's perceptual understanding by fusing traditional visual features with high-level semantic information extracted from an LLM. This method, illustrated in Figure 2, aims to create a richer and more informative state representation that can support better policy learning in complex, procedurally generated environments like those in Procgen.

Each frame (observation) from the environment is passed through a pretrained ResNet50 encoder to extract spatial visual features, resulting in a 2048-dimensional vector representing the low-level visual content of the observation. We chose ResNet50 due to its strong performance as a general-purpose visual backbone, offering high representational capacity while being computationally efficient. In parallel, the same observation is fed into an LLM (Claude Sonnet 4) with a prompt that asks "What visual information would be most helpful for an agent to know?" with more specific details shown in Appendix A. This open-ended question is designed to elicit task-relevant, concise natural language descriptions from the model.

The LLM's textual response is then passed through a sentence embedding model (all-MiniLM-L6-v2) to convert the free-form response into a fixed-size 384-dimensional vector representation. The 2048-dimensional ResNet50 features and 384-dimensional LLM semantic embedding are concatenated to form a single 2432-dimensional fusion vector. This fused representation is processed through a custom feature extractor consisting of the same three-layer convolutional architecture as our baseline PPO (32, 64, 64 filters with 8×8, 4×4, 3×3 kernels respectively), followed by a fully connected layer that outputs a 256-dimensional CNN feature vector. This 256-dimensional CNN output is then

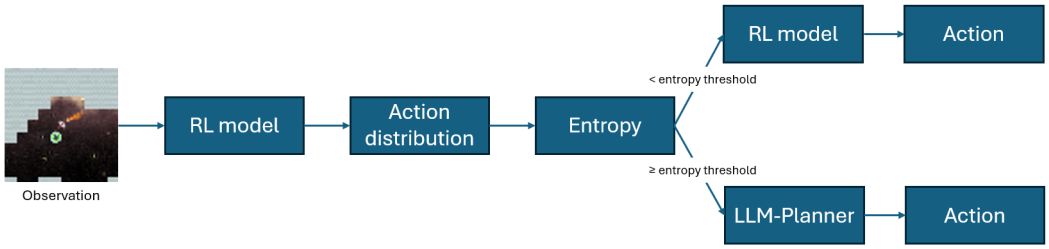


Figure 3: AWU steps. The agent chooses between using an RL model versus the LLM-Actor by calculating the entropy value of its actions and comparing it to a preset threshold.

concatenated with the 2432-dimensional fusion vector to create the final state representation for the policy and value networks.

The LESR agent uses a modified PPO architecture that accepts dictionary observations containing both "image" and "fusion" components, enabling the policy to leverage both raw visual perception and semantic reasoning. We initialize the compatible layers of our LESR policy with pretrained weights from our baseline PPO agent to accelerate learning. Since LESR requires an LLM call every timestep during training, we query the LLM every 30 frames and reuse the last semantic embedding for intermediate frames to reduce query cost and latency.

Hyperparameters: LESR uses the same PPO hyperparameters as our baseline but with reduced rollout collection (64 steps instead of 256) and correspondingly smaller batch size (64) to limit LLM queries during training. We trained for 100000 total timesteps using a single environment, with LLM queries occurring every 30 frames to balance semantic information freshness with computational cost. However, even with this optimization, 100,000 timesteps is shown to be too few for acceptable performance (while still requiring hours to train without showing signs of convergence).

3.5 Hybrid: Ask When Uncertain (AWU)

As shown in Figure 3, the AWU hybrid model alternates between using our PPO (pre-trained from 3.2) and LLM-generated actions based on the entropy of PPO's action distribution, treating entropy as a measure of the agent's uncertainty. The core intuition is that when PPO is confident about its decision (low entropy), it should maintain control, but when PPO exhibits high uncertainty (high entropy), the LLM's reasoning capabilities can provide valuable guidance for difficult states. The AWU hybrid is intended to be a primarily RL-based model, with LLM intervention in stages where PPO gets stuck. It is inspired by the When2Ask (Hu et al. (2024)) methodology, which trains an RL policy to determine which observations to use an LLM vs a pretrained RL agent. We opted for entropy instead of a new RL policy to keep a lightweight combination, and also due to computational + time limitations. As each LLM query costs 2 seconds, training an RL policy for 1M timesteps (which is still insufficient) would require excessive compute credits and take 23 days.

The method operates by first computing the entropy of PPO's action distribution for each observation: $H(\pi_\theta(a|s)) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s)$. To establish an appropriate threshold for determining when to query the LLM, we perform a calibration phase where we run PPO-only episodes across all evaluation levels and collect the distribution of entropy values. We then set the threshold at a specified percentile (e.g., 70th percentile) of this distribution, ensuring that LLM queries occur for states that represent genuine uncertainty rather than routine decision-making. We empirically set this threshold to be 70%.

During evaluation, at each timestep, we compare the current entropy value to the predetermined threshold. If the entropy falls below the threshold, indicating that PPO has a clear preference among actions, we execute PPO's recommended action directly. However, we query the LLM with the current observation and recent frame history to obtain an alternative action recommendation. This approach aims to leverage the complementary strengths of both methods: PPO's efficient learned behaviors for familiar situations and the LLM's general reasoning capabilities for novel or challenging scenarios.

3.6 Hybrid: AWU + Mode Switching

AWU + Mode Switching is defined similarly to AWU, but every time the entropy is above the threshold for a given observation, it switches to "LLM Mode," where it uses the LLM-Actor action for the next 10 timesteps, instead of just the individual timesteps with high entropy. This aims to prevent frequent switching off between LLM and PPO agents, which may result in both agents counteracting each other. As the PPO agent seems to get stuck in unfamiliar situations, 10 consecutive LLM actions are intended to move the agent to a new familiar action before relinquishing control back to the PPO agent. We expect this model to be less RL-focused than the base AWU model and instead encourage frequent querying of the LLM. We empirically set the entropy threshold to be 40% to encourage frequent mode switching.

4 Experimental Setup

We ran each method listed above on different Procgen environments, with the PPO and DQN methods as baselines. Procgen includes 16 different procedurally generated environments, which means each level is algorithmically generated on the fly (Cobbe et al. (2020)). It was generated by OpenAI to test how well RL agents can generalize. We decided to use Procgen specifically because it includes discrete actions (i.e., up, left, down, right), making the action space limited enough for LLMs to process. All the environments also have (approximately) continuous dynamics with physics-based simulations, making them harder to solve compared to clearly discrete environments such as MiniGrid. Here are the three specific Procgen environments we experimented with:

- Coinrun: A platformer where the goal is to collect the coin at the far right of the level while avoiding saw obstacles, enemies that go back and forth, and chasms that lead to death. We chose Coinrun for its simplicity of testing generalization.
- Maze: A maze where the player, a mouse, must find a single block of cheese. The mazes are generated using Kruskal's algorithm and range in size from 3x3 to 25x25. We chose Maze because it requires exploration and more advanced reasoning capabilities.
- Caveflyer: The player, a ship, must navigate through a network of caves to reach the exit. The player can rotate and travel forward and backward. Most of the reward is gained from reaching the end of the level, but additional reward can be gained from shooting target objects along the way with lasers. There are both moving and stationary lethal objects on the level. We chose Caveflyer for its requirement of more fine-grained control through its rotation and additional component of shooting down targets.

(a) Coinrun (b) Maze (c) Caveflyer

Figure 4: Example screenshots from the three Procgen environments used in our evaluation.

For each environment, we evaluated on 10 different easy-mode levels for LLM-based and hybrid agents due to query cost and time, and 200 different levels for purely RL-based agents. To evaluate our methods, we selected the following metrics: average reward, average steps, average LLM queries, and wall clock time. Average reward serves as a standard baseline for assessing overall task performance. Average steps indicate the efficiency of the learned policy, with a higher number of steps suggesting that the agent took longer to reach the goal. Average LLM queries provide insight into the agent's reliance on LLM guidance and the associated latency from querying. Lastly, wall clock time measures the average time required for each timestep, under PPO, DQN, and LLM-based computation.

5 Results

5.1 Quantitative Evaluation

Environment	Method	Eps	Avg. Rwd	Std. Rwd	Avg. Steps	Avg. Queries
Coinrun	LLM-Actor	1	**8**	4	56.3	56.3
	PPO	1	6.15	4.87	394.88	0
	AWU	1	3	4.58	127.9	10.2
	AWU Mode Switch	1	6	4.9	41.7	36.8
	DQN	1	0	0	1000	0
	LESR	1	0	0	1000	1000
Maze	LLM-Actor	.4	**2.5**	4.33	375.25	375.25
	PPO	1	0.80	2.71	460.18	0
	AWU	.7	1	3	449.10	10.5
	AWU Mode Switch	1	1	3	449.7	10.4
	DQN	1	0	0	500	0
	LESR	1	0	0	500	500
Caveflyer	LLM-Actor	.5	2	4	236.2	236.2
	PPO	1	**2.26**	4.17	630.42	0
	AWU	1	1	3	365.9	5.5
	AWU Mode Switch	.8	2	4	291.9	53.7
	DQN	1	0.05	.705	927.6	0
	LESR	1	0	0	1000	1000

Table 1: Comparison of LLM-Actor, PPO, and hybrid approaches across environments. "Eps" represents the proportion of episodes where LLM gave correct output (i.e., a single action ID, if applicable). "Avg. Rwd" was averaged over only the valid episodes.

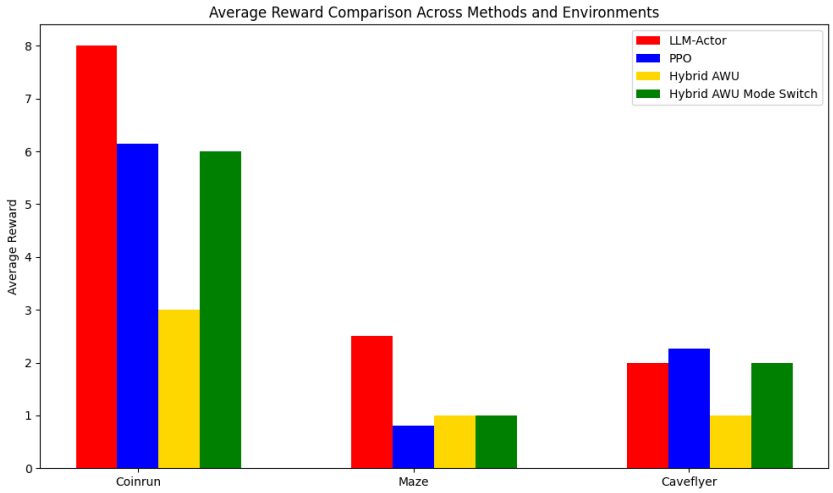


Figure 5: Average reward comparison of LLM-Actor, PPO, Hybrid AWU, and Hybrid AWU + Mode Switch across environments.

Table 1 and Figure 5 show the results for all our methods across all Procgen environments, including the aforementioned metrics and a comparison between the key models in the figure.

Across all environments, we see low and zero reward performance for DQN, as we see nearly all episodes hit the maximum timestep limit without approaching the reward. DQN's failure across all environments reflects fundamental challenges that value-based methods face in sparse reward settings characteristic of Procgen. With rewards structured as binary outcomes (0 for no progress, 10 for goal completion), DQN receives minimal learning signal during exploration phases, making Q-value propagation extremely slow. The replay buffer becomes dominated by unsuccessful trajectories with zero rewards, creating a negative feedback loop where the agent primarily learns from failure

experiences rather than successful behaviors. Additionally, the combination of complex 64×64×3 visual observations and procedural generation requires robust exploration strategies that simple ϵ-greedy exploration (decaying from 1.0 to 0.1) cannot adequately provide. The continuous dynamics further complicate Q-function approximation, as the agent must simultaneously learn effective visual representations and value functions in environments where positive feedback is rare and delayed.

We observe similarly low performance for LESR, with all episodes again hitting the maximum timesteps without approaching rewards. LESR's consistently poor performance across all environments stems from several critical limitations. Most significantly, LESR was only trained for 100,000 timesteps compared to PPO's 4 million timesteps, providing drastically insufficient learning opportunities for its complex architecture that fuses 2048-dimensional visual features with 384-dimensional semantic embeddings. The computational bottleneck of requiring LLM queries created a practical training constraint—even with optimization to query every 30 frames, each episode still required substantial wall-clock time due to the 2-second latency per LLM call. While we tried to reuse the 20-30 applicable layers from our pre-trained PPO agent, 100,000 timesteps still proved insufficient to allow parameters for LLM and ResNet extracted features to converge, resulting in zeroed-out performance. Furthermore, the high-level semantic information extracted from the LLM may not provide actionable guidance for the fine-grained control decisions required in these continuous dynamics environments. The semantic descriptions might have lacked the spatial precision needed for successful navigation in procedurally generated levels with sparse rewards.

5.1.1 Coinrun Analysis

For Coinrun, the LLM-Actor agent was the strongest with an average reward of 8. PPO and AWU Mode Switch were close behind with average rewards of 6.15 and 6, respectively. Compared to other Procgen environments in our study, Coinrun yielded significantly higher rewards across all methods. This disparity is likely attributed to the relative simplicity and clarity of Coinrun's objective, which is simply that the agent must navigate rightward while avoiding a limited set of obstacles to reach a visible coin. This makes it particularly easy for LLMs, even in a zero-shot setting, to infer the appropriate behavior even with limited context. Additionally, the low visual clutter and sparse distractors allow vision-based agents to form useful policies quickly. Something we found was that a policy that only outputted the action ("RIGHT", "UP") was surprisingly effective, often yielding strong performance without any sophisticated planning. Since the LLM was prompted with prior frames and a simple goal description, it could deduce that progressing to the right while jumping was sufficient for success. Unlike in more complex environments, this relatively deterministic and low-variance strategy happened to be near optimal in Coinrun, playing directly to the strengths of LLMs' prior knowledge and pattern-matching capabilities.

Furthermore, Hybrid AWU Mode Switch had the lowest Avg. Steps (41.7) per episode, showing that it is more efficient in finishing the episode. Even though it had a similar performance to the pure PPO agent, it completed its runs about 10 times faster. LLM-Actor was slightly less efficient with 56.3 average steps per episode, but it had a noticeably higher average reward as a tradeoff.

Additionally, AWU Mode Switch is primarily LLM dominated with 36.8 queries out of 41.7 timesteps, helping to explain why it is more efficient in reaching the goal. In addition, seeing how much more often it queries the LLM in comparison to the AWU without mode switch, it makes sense why its average reward is doubled. This aligns with our results since LLM-Actor did the best, and with more calls to the LLM, we would expect its average reward to increase.

Lastly, LLM-Actor required the most number of queries because it has a query for every frame. It is also interesting to note that all LLM-based methods had 100% valid episodes, demonstrating that the LLM was able to understand the prompt for the Coinrun environment, likely due to the LLM already being very familiar with the classic Coinrun environment used in RL research.

5.1.2 Maze Analysis

In Maze, all agents had relatively low rewards. The LLM-Actor again did the best in terms of average reward, but only 40% of its episodes were considered valid. Hybrid AWU had similar struggles of having only 70% valid episodes. The low rate of valid episodes highlights the LLM's struggle in interpreting the pixilated images and generating a coherent movement plan. PPO performed the worst with 0.80 average reward and required the largest number of steps on average, and both AWU hybrid

models did quite poorly as well. It makes sense that all RL-related models performed poorly on the maze due to the sparse rewards in complex maze setups. During training, the RL models rarely ever got to the reward, and thus never learned to prioritize actions going towards the reward.

As before, LLM-Actor had the highest number of queries. However, in contrast to Coinrun, which only had a modest decrease in LLM queries, we see there are 37 times more LLM queries in the LLM-Actor than in both hybrid AWU methods. We see that the hybrid AWU methods are much more PPO reliant than LLM reliant for this environment, as only about 2% of environment calls prompt LLM calls (as opposed to $\sim$90% in Coinrun). This indicates that even when the PPO agent gets stuck in the maze environment, it has low entropy and is confident in its decision, preventing the agent from getting the LLM-based gains. Similarly, because the LLM agent is queried rarely and also has low performance in the maze, adding the switch mode optimization to AWU does not change performance or help the PPO agent become unstuck.

5.1.3 Caveflyer Analysis

In Caveflyer, all agents again had low rewards. LLM-Actor struggled with valid responses, with only 50% valid episodes, and had an average reward of 2. Again, the low total reward and valid episode count underscore the LLM's struggles in interpreting the prompt and making a fine-grained plan in a low-resolution image. Adding complex dynamics such as rotation and shooting, along with simultaneous directional movement, proved too fine-tuned for the LLM to create a good plan. Contrastively, PPO had the highest average reward of 2.26, due to it being stronger at the fine-grained control needed for the environment. In this dynamic environment, task-specific training proves to be much more useful than few-shot general reasoning.

AWU Mode Switch was able to outperform AWU in terms of reward and steps required, signifying that mode switching to LLM allowed the agent to get unstuck and reach the goal more often at the cost of requiring more LLM queries. However, this reliance on the LLM seemed to leave the agent vulnerable to the LLM's inherent limitations in fine-grained control, as evidenced by the lower success rate (80% valid episodes) compared to pure PPO. The hybrid approaches in Caveflyer demonstrate an interesting trade-off: while AWU Mode Switch queries the LLM nearly 10 times more than standard AWU (53.7 vs 5.5 queries), this increased LLM reliance doubled the average reward and reduced completion time by 20%. This suggests that in navigation-heavy scenarios like Caveflyer, the LLM's spatial reasoning capabilities can complement PPO's control precision, but the benefits are limited by the LLM's struggle with the complex action space that requires coordinated rotation, movement, and shooting decisions. The relatively low query frequency in standard AWU (5.5 out of 365.9 steps) indicates that PPO maintains high confidence even in challenging situations, potentially missing opportunities where LLM guidance could provide strategic navigation insights, and explaining why even slightly hybrid strategies still fail to outperform pure PPO agents.

Notably, in both Caveflyer and Coinrun, we observe that using mode switching allows the hybrid model to recover LLM-level performance while normal AWU does not. This pattern suggests that single-timestep LLM interventions in standard AWU are insufficient to overcome challenging navigation scenarios—the agent needs sustained LLM guidance over multiple consecutive timesteps to effectively change its trajectory and escape local optima. The 10-timestep mode switching appears to provide enough temporal consistency for the LLM's strategic reasoning to meaningfully redirect the agent's behavior, whereas isolated LLM actions get quickly overridden by PPO's learned patterns. This finding highlights the importance of temporal coherence in hybrid RL-LLM systems, where brief LLM interventions may be diluted by the underlying policy's momentum.

5.1.4 Wall Clock Time Analysis

Table 2 shows that the LLM-Actor method is significantly more computationally expensive than both PPO and DQN, with per-timestep wall clock times exceeding 4 seconds in all environments, compared to under 25 milliseconds for PPO and near-instantaneous times for DQN. This disparity reflects the overhead of querying an LLM on each frame, making the LLM-based approaches impractical for real-time deployment without optimization. Even with accounting for latency costs, LLM wall-clock times are orders of magnitude higher than traditional RL. However, these results also emphasize the trade-off between generalist reasoning and efficiency. While LLMs can perform well in zero-shot settings, it is at the cost of speed and scalability.

Environment	Method	Wall Clock Time (s)
Coinrun	LLM-Actor	6.393
	PPO	0.020
	DQN	0.012
Maze	LLM-Actor	5.509
	PPO	0.021
	DQN	0.001
Caveflyer	LLM-Actor	4.130
	PPO	0.022
	DQN	0.001

Table 2: Wall clock time per timestep comparison across methods and environments.

5.2 Qualitative Analysis

Figure 6 demonstrates the various situations faced by the different agents: the agent would either get stuck or stall, fail/die in the episode, or achieve the goal. Similar figures for Maze and Caveflyer can be found in Figure 7 and Figure 8.

5.2.1 LLM-Actor

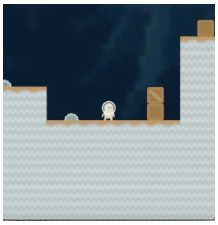
LLM-Actor was quite effective in Coinrun in reaching the goal coin across all levels. Since the coin was always on the right side of the level, the agent would almost always choose the action representing a right jump, which generally led it to the coin. However, the agent was not very good at more fine-tuned decisions, as it would occasionally jump and land onto a saw without maneuvering itself away from the hazard, which is technically possible due to the continuous dynamics. In Maze, the agent would frequently provide invalid responses; instead of outputting a single action ID, it would explain what it saw in the image and then reason through a good action, even though the prompt specified that it to provide just a single action ID. Even in the valid episodes, the majority of the time, the agent would do the same action repeatedly and get stuck in front of a wall. The agent would still get stuck even if we specified in the prompt for it to "turn around" if it sees itself getting stuck (i.e., past three frames are identical). In Caveflyer, the LLM-Actor would similarly provide invalid responses and also get stuck. However, after prompting it to get unstuck by rotating itself, the agent did significantly better and would get unstuck, demonstrating some capability of adapting based on its current and past states. Although it could get unstuck in Caveflyer, the agent would often hit hazards as well.

5.2.2 PPO

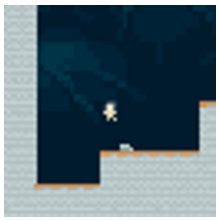
In Coinrun, PPO would generally either jump and move to the right or it would stand still and freeze. This is reflected by its higher rewards but a much higher number of steps per episode. In Maze, the agent would only learn to move in one direction (left, right, up, down), which reflects its inability to learn effectively in an environment that requires more reasoning and exploration. This is demonstrated by its low rewards and high number of steps. In Caveflyer, the agent behaved similarly to how it did in Coinrun. It would either move around a lot and eventually reach the goal (or crash), or it would freeze without moving at all. However, since the PPO model was specifically trained for this task, it was able to reach the goal more often.

5.2.3 AWU and AWU with Mode Switching

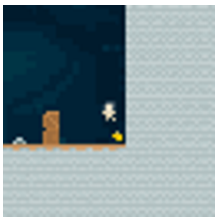
The AWU agent resulted in mostly jittered movement throughout all the environments except for Maze, where it was able to occasionally get unstuck with help from the LLM. The jittered movement is likely due to the PPO and LLM agents having conflicting actions. AWU with Mode Switching, on the other hand, had less jittered movement due to using the LLM for 10 straight timesteps instead of switching back and forth. This smoother movement led to better results, as demonstrated by its higher rewards in Coinrun and Caveflyer.



(a) Getting stuck/stalling (usually as PPO Agent)



(b) Falling onto a saw due to failing to maneuver mid-jump (usually as LLM Agent)



(c) Reaching the goal coin

Figure 6: Different Coinrun situations faced by agents

5.3 Overall Trends

Across both qualitative and quantitative settings, several key behavioral patterns emerge that help explain the relative performance of different approaches. PPO agents consistently tend to get stuck in local optima, leading to significantly longer episode durations across all environments. This is particularly evident in Coinrun, where PPO averaged 394.88 steps compared to LLM-Actor's 56.3 steps, and in Maze, where PPO required 460.18 steps on average. Qualitatively, PPO agents become trapped in repetitive behaviors or static positions, and they continue executing the same actions without making meaningful progress toward the goal, ultimately timing out at the maximum episode length.

In contrast, LLM agents demonstrate a preference for exploratory, seemingly random movement that, while sometimes appearing suboptimal, serves as an effective mechanism for avoiding local optima. This exploratory behavior often leads to either reward discovery through continued exploration or early episode termination through environmental hazards, explaining why LLM-based methods consistently achieve much shorter episode durations. However, this randomness comes at the cost of precision—LLM agents frequently fail at fine-grained control tasks, such as maneuvering around obstacles in Coinrun or executing coordinated rotation and movement in Caveflyer.

The superior performance of AWU Mode Switch across multiple environments can be understood through this behavioral lens. When PPO is not stuck and operating within familiar state spaces, it demonstrates the highest performance due to its specialized training and precise control capabilities. However, when PPO encounters unfamiliar situations or gets trapped in local optima, the sustained LLM intervention (10 timesteps) provides sufficient exploratory momentum to escape these problematic states and redirect the agent toward more promising regions of the state space. This explains why AWU Mode Switch consistently outperforms standard AWU—single-timestep LLM interventions are insufficient to overcome PPO's learned behavioral patterns, whereas sustained LLM control can effectively "reset" the agent's trajectory.

6 Discussion

Our study faced significant limitations that highlight the current challenges in LLM-RL integration. The computational overhead of LLM queries (approximately 2-6 seconds per call) severely constrained our evaluation scope, limiting LLM-based methods to only 10 episodes compared to 200 for pure RL approaches and preventing adequate training of hybrid methods like LESR. While PPO agents required about 4 million timesteps to achieve reasonable results, the computational bottleneck proved particularly problematic for LESR, which only had 100,000 timesteps of training, resulting in zero performance across all environments. We attempted to address this limitation by experimenting with local, open-source models like LLaVa that offered much smaller query times and latency. However, the fundamental problem with using smaller models was that they lacked the general reasoning ability that large, closed-source LLMs like Claude possessed, making models that were small and fast enough to be practical for extended training insufficient for meaningful performance improvements. This represents a core challenge in the field: the models with the strongest reasoning capabilities are too computationally expensive for extensive training, while efficient models lack the cognitive capabilities that make LLM-based planning attractive in the first place.

Additional technical challenges further constrained our findings. The 64×64 pixel resolution of Procgen observations proved particularly challenging for LLM visual understanding, with many failures in Maze and Caveflyer environments attributable to difficulty interpreting low-resolution imagery for precise spatial reasoning. Prompt engineering also presented ongoing difficulties, with LLMs frequently providing explanatory text rather than the requested single action IDs, particularly in complex environments where valid episode rates dropped as low as 40% for Maze and 50% for Caveflyer. The entropy-based switching mechanism in AWU required careful calibration of threshold values that proved sensitive to both environment and policy characteristics, while the choice of 10 timesteps for mode switching was based on intuition rather than systematic optimization. These limitations highlight the need for more principled approaches to hybrid system design and evaluation.

The 300x computational overhead of LLM-based approaches raises critical questions about resource efficiency and environmental impact at scale, suggesting these methods are best suited for high-value, low-frequency decision-making rather than real-time control. While LLMs offer promising democratization of AI agent development through zero-shot capabilities that require minimal domain-specific setup, our results demonstrate that specialized training remains crucial for optimal performance in complex environments. This positions hybrid approaches as the most practical path forward, with our mode switching results indicating that future autonomous systems should dynamically allocate control between specialized and generalist components based on situational demands, particularly relevant for robotics applications requiring both routine execution and novel problem-solving capabilities.

7 Conclusion

This study provides a comprehensive comparison of zero-shot LLM planning, traditional reinforcement learning, and hybrid approaches across multiple procedurally generated environments with continuous dynamics. Our findings reveal that while LLMs can achieve competitive performance in simpler navigation tasks through general reasoning capabilities, they face fundamental limitations when applied to complex interactive environments requiring precise control and spatial reasoning from low-resolution visual inputs. The key insight from our work is that effective LLM-RL integration requires temporal consistency. Brief LLM interventions are insufficient, but sustained guidance through mode switching can successfully combine the strategic reasoning of LLMs with the learned behaviors of RL agents.

Looking toward future research directions, several key challenges and opportunities emerge from our work. First, we would like to expand on the LESR model by exploring hierarchical hybrid approaches where LLMs provide high-level strategic guidance (versus just state representation as in LESR) while specialized policies handle low-level control; this could better leverage the natural division between strategic reasoning and motor control. In addition, systematic approaches to threshold selection, mode switching duration, and authority allocation could improve the reliability and performance of hybrid methods across diverse environments. As both LLM capabilities and computational efficiency continue to improve, the hybrid approaches demonstrated in this work may serve as stepping stones toward more capable and practical autonomous agents that combine the generalist reasoning of large language models with the specialized optimization of reinforcement learning.

8 Team Contributions

- **James Cheng:** Set up LLM pipeline for LLM-based methods such as LLM-Actor and hybrid models. Ran experiments for LLM-Actor.
- **Nishikar Paruchuri:** Set up pipeline for PPO, DQN, and LESR training. Ran experiments for PPO, DQN, LESR, and AWU.
- **Andy Ouyang:** Implemented hybrid methods, LESR and AWU.

Changes from Proposal We changed our testing environment from Minigrid to Procgen to evaluate on more complex and dynamically changing states. In addition, we also came up with and evaluated hybrid approaches that combined the strengths of LLMs and classical RL. As we added scope to our project, we updated our division of method implementation for equitable balance.

References

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. arXiv:2204.01691 [cs.RO] https://arxiv.org/abs/2204.01691

Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, Kanishka Rao, Krista Reymann, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Pierre Sermanet, Jaspiar Singh, Anikait Singh, Radu Soricut, Huong Tran, Vincent Vanhoucke, Quan Vuong, Ayzaan Wahid, Stefan Welker, Paul Wohlhart, Jialin Wu, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. arXiv:2307.15818 [cs.RO] https://arxiv.org/abs/2307.15818

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. 2020. Leveraging Procedural Generation to Benchmark Reinforcement Learning. arXiv:1912.01588 [cs.LG] https://arxiv.org/abs/1912.01588

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. 2019. Quantifying Generalization in Reinforcement Learning. arXiv:1812.02341 [cs.LG] https://arxiv.org/abs/1812.02341

Bin Hu, Chenyang Zhao, Pu Zhang, Zihao Zhou, Yuanhang Yang, Zenglin Xu, and Bin Liu. 2024. Enabling Intelligent Interactions between an Agent and an LLM: A Reinforcement Learning Approach. arXiv:2306.03604 [cs.AI] https://arxiv.org/abs/2306.03604

Andrew Jesson and Yiding Jiang. 2024. Improving Generalization on the ProcGen Benchmark with Simple Architectural Changes and Scale. arXiv:2410.10905 [cs.LG] https://arxiv.org/abs/2410.10905

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, Jonathan Harper, Ervin Teng, Hunter Henry, Adam Crespi, Julian Togelius, and Danny Lange. 2019. Obstacle Tower: A Generalization Challenge in Vision, Control, and Planning. arXiv:1902.01378 [cs.AI] https://arxiv.org/abs/1902.01378

Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. 2018. Gotta Learn Fast: A New Benchmark for Generalization in RL. arXiv:1804.03720 [cs.LG] https://arxiv.org/abs/1804.03720

Byeong-Ju Park, Sung-Jung Yong, Hyun-Seo Hwang, and Il-Young Moon. 2025. Optimizing Agent Behavior in the MiniGrid Environment Using Reinforcement Learning Based on Large Language Models. *Applied Sciences* 15, 4 (2025). https://doi.org/10.3390/app15041860

Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D. Gaina, Julian Togelius, and Simon M. Lucas. 2019. General Video Game AI: a Multi-Track Framework for Evaluating Agents, Games and Content Generation Algorithms. arXiv:1802.10363 [cs.AI] https://arxiv.org/abs/1802.10363

Boyuan Wang, Yun Qu, Yuhang Jiang, Jianzhun Shao, Chang Liu, Wenming Yang, and Xiangyang Ji. 2024. LLM-Empowered State Representation for Reinforcement Learning. arXiv:2407.13237 [cs.AI] https://arxiv.org/abs/2407.13237

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] https://arxiv.org/abs/2210.03629

A Implementation Details

Prompts for LESR

"""Environment: CoinRun A simple platformer. The goal is to collect the coin at the far right of the level, and the player spawns on the far left. The agent must dodge stationary saw obstacles, enemies that pace back and forth, and chasms that lead to death. Note that while the previously released version of CoinRun painted velocity information directly onto observations, the current version does not. This makes the environment significantly more difficult.
Question: Based on this image, what visual information would be most useful for an agent to know in order to successfully complete the level? Answer specifically in terms of the environment and obstacles present in the scene and where they are located in reference to the agent.""",

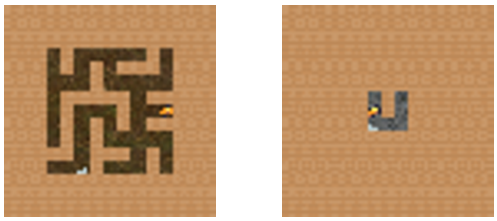
"""Environment: Maze The player, a mouse, must navigate a maze to find the sole piece of cheese and earn a reward. Mazes are generated by Kruskal's algorithm and range in size from 3x3 to 25x25. The maze dimensions are uniformly sampled over this range. The player may move up, down, left or right to navigate the maze.
Question: Based on this image, what visual information would be most useful for an agent to know in order to successfully complete the level? Answer specifically in terms of the environment and obstacles present in the scene and where they are located in reference to the agent.""",

"""Environment: CaveFlyer The player must navigate a network of caves to reach the exit. Player movement mimics the Atari game "Asteroids": the ship can rotate and travel forward or backward along the current axis. The majority of the reward comes from successfully reaching the end of the level, though additional reward can be collected by destroying target objects along the way with the ship's lasers. There are stationary and moving lethal obstacles throughout the level.
Question: Based on this image, what visual information would be most useful for an agent to know in order to successfully complete the level? Answer specifically in terms of the environment and obstacles present in the scene and where they are located in reference to the agent."""

Prompt for Maze

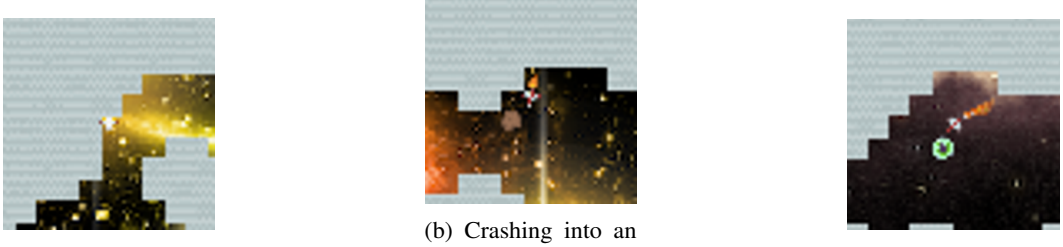
"You are an intelligent agent in ProcGen's maze environment."
"You will be shown several sequential frames, where higher frame numbers are more recent."
"Use them to understand how the scene is changing over time."
"You, a gray mouse, need to find the sole piece of yellow cheese and earn a reward."
"You must choose exactly one of the following discrete actions:"
"1: move left"
"3: move down"
"5: move up"
"7: move right"
"If you find yourself getting stuck in front of a wall (i.e., the past three frames are the same), please try to get unstuck by moving in the opposite direction of the wall."
"Please analyze the progression of the scene and respond with only the single most appropriate action ID (e.g., '3'). " "Do not explain your reasoning or describe the images. Just return one action ID from the list above."

B Additional Figures



(a) Getting stuck (b) Reaching the goal

Figure 7: Different situations faced by agents in the Maze environment



(a) Getting stuck (b) Crashing into an asteroid (c) Reaching the goal

Figure 8: Different Caveflyer situations faced by agents