

TEACHING LARGE LANGUAGE MODELS TO PLAY ATARI GAMES

Leon Guertler
U2023443J
leon002

Huang Jingfang
U2020128A
jhuang058

Kim Chae Yoon
U1923512A
kimc0004

1 INTRODUCTION

”Human-level control through deep reinforcement learning” Mnih et al. (2013) is not only one of the most important works in recent AI history, but also, arguably, the work that kick started the recent AI renaissance. Their groundbreaking approach demonstrated the feasibility of training a neural network to excel at Atari games, surpassing human performance. Remarkably, they achieved this feat by providing the agents only the screen captures of the games as input, eliminating the need for game-specific knowledge. This revolutionary paradigm laid the foundation for a surge of innovative deep reinforcement learning (DRL) algorithms, each striving to build upon or surpass the initial achievements of Mnih et al. (2013).

The cascade of innovations following the initial DQN breakthrough underscores the explosive growth and dynamism of the field. Researchers have continually refined and expanded the tool-set of deep RL, pushing the boundaries of what agents can achieve in complex environments like Atari games. These advancements have not only solidified the role of deep RL in game-playing AI but have also opened the doors to its applications across a vast spectrum of real-world applications.

Despite the advancements in various algorithms, the issue of sample efficiency remains a common challenge. While numerous models surpass human-level performance on many Atari games, and some have even achieved mastery across all games (Badia et al., 2020), these remarkable feats come at the cost of immense sample sizes, often requiring millions of steps equivalent to weeks of playtime. In contrast, humans can often grasp these games within a mere two hours. This discrepancy motivated the inception of the Atari-100k benchmark, which restricts agents to 100,000 steps, mimicking the approximate time a human would spend learning to play the game.

Our hypothesis is that the vast disparity in sample efficiency might be attributed to the innate world-model that humans possess and employ when learning new games. Incorporating such a world-model into a randomly initialized RL agent is no straightforward task. However, we speculate that harnessing the world models embedded in a pre-trained LLM could offer a solution. Our aim is to utilize the world knowledge of the current sota vision language model (VLM), Fuyu (Bavishi et al., 2023).

However, given the complexity of the experiments, we will first compare a number of standard DRL algorithms on the atari-100k benchmark.

The structure of this paper is as follows: we first highlight relevant works in Section 2, subsequently, Section 3 dives into the DRL algorithms under evaluation; Section 3.4 shows how we can utilize LLMs (or more specifically VLMs) in a DRL set-up, Section 4 shows our experiments and their results; in Section 4.3 we offer an analysis of the results and finally, we conclude in Section 5.

2 RELATED WORK

In this section we will summarize relevant related works in traditional deep reinforcement learning (Section 2.1), and recent papers that have attempted to utilize large language models (LLMs) in a Deep Reinforcement Learning context (Section 2.2).

2.1 TRADITIONAL DEEP REINFORCEMENT LEARNING

Deep Q Networks (DQN) As the pioneering work by Mnih et al. (2013), the Deep Q Network (DQN) algorithm combined traditional Q-learning with deep neural networks. This was revolutionary, as it allowed RL agents to handle high-dimensional input spaces, like raw pixel data from video games. The introduction of techniques such as experience replay and target networks made training more stable and efficient.

Double DQN (DDQN) Building on the success of DQN, van Hasselt et al. (2015) introduced the Double DQN, which mitigates the overestimation bias of DQN by using two networks: one to select actions and the other to evaluate these actions. This results in more stable and sometimes faster learning.

Prioritized Experience Replay: Schaul et al. (2016) recognized that not all experiences are equally valuable for learning. They proposed prioritizing more "surprising" or "unexpected" transitions, leading to faster convergence and improved learning stability.

Dueling DQN Wang et al. (2015) introduced an architectural change to the neural network used in DQNs by splitting it into two streams: one to estimate the state value and another to estimate the advantage for each action. This separation allows the network to learn more efficiently when only the value of the state changes and not the actions.

Rainbow Hessel et al. (2017) combined the strengths of the aforementioned enhancements into a single algorithm named Rainbow. By integrating DQN, DDQN, Prioritized Experience Replay, Dueling DQN, and other techniques, Rainbow became one of the state-of-the-art algorithms for Atari games, showing substantial improvements over each individual component.

A3C and A2C Beyond DQN-based methods, Mnih et al. (2016) also introduced the Asynchronous Advantage Actor-Critic (A3C) method, which uses multiple agents working in parallel to update a global model. This decentralization accelerates learning and promotes exploration. A2C, a synchronous variation, later emerged to offer more stable updates by averaging over multiple experiences.

MuZero A significant advancement in the field was achieved with MuZero, developed by Schrittwieser et al. (2019), which extends the capabilities of previous models to games where the environment's dynamics are unknown. MuZero uses a model-based approach that, instead of learning a model of the environment's dynamics for planning, learns a value and policy directly from the data. It combines tree-based search with a learned model to master both Atari-like environments and board games like chess and Go, outperforming previous methods without the need for domain knowledge.

2.2 LARGE LANGUAGE MODELS IN DEEP REINFORCEMENT LEARNING

LLMs as Reward Models The most popular application of LLMs in DRL is using them as a reward estimator (Adeniji et al., 2023; Du et al., 2023; Kwon et al., 2023; Ma et al., 2023). These models mostly differ in how the reward is calculated. Some of them focus on an exploration based reward via text and language encodings (Rocamonde et al., 2023; Adeniji et al., 2023), generating reward functions in executable code (Ma et al., 2023) or returning binary rewards based on how well the current trajectory satisfies the objective (Kwon et al., 2023),

LLMs as DRL Agents The literature on employing LLMs as decision-making agents in robotics and game settings is particularly important to our work. Generally speaking, the literature can be subdivided into 5 main approaches, namely: Fine-tuning the prompt (which can be done via a secondary trainable model (Yao et al., 2023) or NLP feedback by a secondary frozen LLM (Singh et al., 2022; Shinn et al., 2023)); using a frozen LLM as a high-level planner and training a secondary (smaller) model as an action module (Ahn et al., 2022); using a frozen LLM to write code that is executed on the environment (Wang et al., 2023a); pre-training an LLM on both robotics control and language data (Brohan et al., 2023); and, finally, simply using it as a zero-shot action predictor (Lin et al., 2023).

3 MODELS

In this section we will first introduce the three standard DRL models we will be using (Section 3.1, Section 3.2 and Section 3.3), highlighting all design choices. Subsequently, in Section 3.4 we will introduce our approach for utilizing a Large Language Model (to be more specific a Vision Language Model) in the context of DRL on Atari games.

3.1 STANDARD DQN

Our baseline model is a simple DQN (Mnih et al., 2013) with a decaying epsilon-greedy exploration strategy and without a priority replay module. The objective is to maintain simplicity, allowing a clearer examination of how the design decisions outlined for Mini-Rainbow (refer to Section 3.2) impact performance.

3.2 MINI RAINBOW

To improve the performance of the standard DQN, we borrow a number of ideas introduced in Rainbow Hessel et al. (2017), namely:

3.2.1 DEEPER NETWORK ARCHITECTURE & BATCH NORMALIZATION

Though this is not entirely unique to Rainbow, one of the key differences between our Mini Rainbow and the Standard DQN model is that we increase the size of the network and introduce batch normalization (Ioffe & Szegedy, 2015) layers. The reason for doing so is that the standard DQN we built was rather shallow and thus might have struggled to properly learn the visual features of the game. Additionally, the motivation for adding the batch normalization is that it will speed up training by reducing internal covariate shifts (i.e. the layer doesn't need to adapt to the changing distribution of inputs during training).

3.2.2 NOISY NETS FOR EXPLORATION

Noisy Linear Layers are one of the core components of Rainbow, addressing exploration in contrast to the conventional Epsilon-Greedy approach employed in Standard DQN. Noisy Linear Layers work by using parameters that introduce noise to the weights of the linear layer. As part of our implementation, we substituted all linear layers with the noisy layers.

3.2.3 DOUBLE DQN

Another key component of Rainbow is the Double DQN. In standard DQN, there can be a tendency to overestimate the Q-values. This arises because of the max operation used to select the estimated future reward for the best action in a given next state. Opting for the maximum action when we lack certainty about the true action values can introduce a systematic bias. Double DQNs (DDQNs) address this issue by decoupling the action selection from the Q-value. In a DDQN:

1. **Action Selection:** it uses the **policy network** (the one that gets regularly updated) to select what it deems the optimal action for the next state.
2. **Action Evaluation:** it uses the **target network** (the one that gets updated less frequently) to estimate the Q-value of taking that action in the next state.

So, while the action is chosen based on the policy network, its value is estimated by the target network, leading to a less biased estimate.

3.2.4 DUELING DQN

Another key component of Rainbow is the Dueling DQN. The key idea behind this design is that not all actions hold equal significance across all states. In many states, the chosen action might have minimal impact on the immediate outcome. Dueling DQN addresses this by separating the Q-value estimation into two streams:

1. **Value Stream (V):** This estimates the value of being in state s , without considering any specific action.
2. **Advantage Stream (A):** This estimates the advantage of taking action a in state s , reflecting how much better or worse action a is compared to the other possible actions.

The Q-value is obtained by combining the outputs of these two streams ($Q(s, a) = V(s) + (A(s, a) - \text{average}(A(s, \text{all_action})))$) (the subtraction of the average advantage ensures that the Q-values are centered around the value function, $V(s)$).

3.2.5 REWARD SCALING

This is an approach that was used in the original DQN paper (Mnih et al., 2013) and in Rainbow (Hessel et al., 2017) (but not in our standard DQN implementation from above). It helps in stabilizing the learning by ensuring reward don't have disproportionate effects on the Q-value updates. For this model, we clip the rewards in range $[-1, 1]$.

However, there are some aspects of Rainbow that we did not implement in our Mini Rainbow models, namely:

3.2.6 PRIORITIZED EXPERIENCE REPLAY (PER)

Rainbow uses PER to sample important transitions more frequently based on the TD error. However, we wanted to minimize the number of things changed from model to model, so used uniform sampling for both the standard DQN and the Mini Rainbow model.

3.2.7 MULTI-STEP LEARNING

Rainbow incorporates n-step returns. instead of using single-step Q-learning updates, it uses multi-step bootstrap targets, which can lead to faster learning since it considers multiple steps into the future. However, we stick to single-step Q-learning updates for both the standard DQN and Mini Rainbow.

Additionally to the abstract DRL concepts, we also experimented with different architectures, including the following:

Attention Adding an attention layer to the DQN architecture introduces a mechanism for the model to selectively focus on features from the input state representation. The selective focus mimics a form of short-term memory, allowing the network to weigh parts of the input differently based on their current relevance to the task. The attention mechanism can potentially improve the learning efficiency of the network by guiding the backpropagation of errors to emphasize more relevant pathways within the neural architecture.

LSTM The LSTM layer enhances a traditional DQN's ability to handle temporal dependencies, thus allowing the network to learn from sequences of observations rather than from the current state only. Therefore, incorporating an LSTM layer into the DQN for the "Centipede" environment allows the model to capture the temporal sequence of events crucial for gameplay, where actions and their consequences unfold over time. Additionally, understanding the movement patterns of the centipede and the destruction of mushrooms requires a memory of past states to strategize effectively. By maintaining a memory of past observations through its hidden states, the LSTM layer allows the DQN to make more informed decisions that consider the sequence of past events, leading to potentially better and more nuanced decisions.

3.3 MUZERO (SCHRITTWIESER ET AL., 2019)

The third of the standard DRL models we will be using is MuZero. It is a state-of-the-art reinforcement learning algorithm developed by DeepMind, positioned at the intersection of model-free and model-based reinforcement learning, MuZero combines the strengths of both paradigms. While traditional model-based approaches require explicit knowledge of environment dynamics for planning, MuZero learns to simulate these dynamics without any prior knowledge, achieving superhuman performance in various domains.

Key Ideas:

1. **Unified Model-Based and Model-Free Learning** MuZero integrates the advantages of model-based reinforcement learning, which involves learning a model of the environment and then planning using this model, with the direct policy and value optimization techniques of model-free learning.
2. **Implicit Learning of Dynamics** Instead of relying on predefined environment rules, MuZero learns the environment dynamics implicitly. This adaptability allows it to be deployed in situations where environment rules are unknown or too intricate to codify explicitly.
3. **Monte Carlo Tree Search with Learned Model** MuZero uses its learned model of the environment to simulate future outcomes. Through Monte Carlo Tree Search (MCTS), the algorithm plans actions based on these simulations, effectively navigating complex decision spaces.

MuZero works by utilizing three separate neural networks:

1. **Representation NN** MuZero begin by converting the observed environment state into a compact latent vector using its representation NN (in the case of atari, we use multiple convolutional layers to map the screen capturing to a flat vector).
2. **Dynamics NN** For planning, it employs the dynamics function to predict the next latent state and reward given the current latent state and an action.
3. **Prediction Function** This function provides estimates of policy (action probabilities) and value for a given latent state.

The orchestration of these neural networks is leveraged in a MCTS framework. Within this search tree, the algorithm leverages its learned model to simulate potential outcomes, instead of accessing the real environment as traditional MCTS would. Post search, the algorithm's action choices are based on the most traversed actions in the search tree.

Training MuZero involves an end-to-end approach where sequences of actions and observations are utilized. The policy output is trained to align with the MCTS visit counts, the value function aims to predict the actual return, and the dynamics function learns to approximate real-world transitions and rewards.

Since the implementation of MuZero is more complex than that of the previous two models, we base our code on the work of Michaelnny. It is also worth to point out that training MuZero takes orders of magnitude of more time than the other two approaches.

3.4 LLM AGENT

Our approach is most similar to that of Lin et al. (2023). In their work, the authors used a number of different neural networks to create modality specific transcriptions of various frames for a given input video. These transcriptions are then combined using GPT-4. Since atari games are much more simplistic than most videos, instead we will be using a single frame as an input, and because running GPT-4V would be prohibitively expensive, instead we use the current open-source state-of-the-art Vision Language Model (Bavishi et al., 2023). Furthermore, following (Ahn et al., 2022), we restrict the output space to the legal actions. Thus, to summarize, the frozen Fuyu model will receive the most recent atari game frame as input, and return the next action. As for all experiments, we use the standard frame-skip of 4.

Since we will be employing the VLM as a zero-shot agent, we instead provide it with the important information from the original atari 2600 game manual. This information is extracted by first transcribing the text part of the game manual (cen, a) and captioning all relevant figures using GPT-4V. Subsequently, the transcription with the captions inserted where necessary are passed to GPT-4, whom is tasked with extracting task relevant information (See Appendix B).

4 EXPERIMENTS

In this section we will both introduce the set-up of our experiments (Section 4.1), their results (Section 4.2) and an analysis thereof (Section 4.3).

4.1 EXPERIMENT SET-UP

As mentioned in the introduction, the goal of this work is to address the sample efficiency issue in modern DRL. The most popular benchmark for this is the Atari-100k benchmark, where the agent is limited to a total of 100k steps in each of the Atari 2600 environments. Following the standard set-up, we apply a frame-skip of 4, and for the standard DRL methods, stack the most recent 4 frames, grayscale inputs, and crop them to size (84, 84). Generally speaking, the atari 100k benchmark entails 29 out of the 57 atari 2600 environments. However, because of the limited compute we have available, we will be focusing our experiments on a single environment: Centipede. The reason for why we chose Centipede is because it seems to have the smoothest learning curve among the environments presented in popular DRL papers, and thus, hopefully allows for easier differentiation of performances in a low data setting.

Before delving into the experiments it is worth pointing out that 100k steps corresponds to just under two hours of human game play, which was the amount of time used by DeepMind to create the human 'expert' scores. However, since then Pouncy et al. (2017) has pointed out that for the games they analyzed, humans were able to match the 'expert' performance within 15 minutes of game play (13.5k steps). Thus, even if an agent performs well on the atari 100k benchmark, one can not conclude that it exhibits human speed learning.

4.2 RESULTS

Model	Score
DQN	3,056
MiniRainbow-base	5,587
MiniRainbow-lstm	2,932
MiniRainbow-attn	2,907
MuZero	251,045
Fuyu	5,889

Table 1: The final test score achieved by the various methods after training for 100k steps on the Centipede Atari 2600 environment.

Table 1 shows the scores achieved by each method after 100k environment steps of training. The only exception to this is Fuyu (our VLM-based agent), which plays in a zero-shot set-up without any training. As expected, MiniRainbow improves upon the performance of the DQN, and MuZero outperforms all other method by a large margin. Interestingly, our VLM based method outperforms both DQN and MiniRainbow, but also comes nowhere near the performance of MuZero. Before analyzing the reasons for the performance differences of the various methods (See Section 4.3), we will comment on the MiniRainbow performances with the various architectures.

As can be seen in the results table, the base version performs substantially better than the LSTM and attention based ones. This is not necessarily surprising as LSTM networks are well known to have instable convergence, and attention networks typically require a lot of data for decent performance (Dosovitskiy et al., 2020).

4.3 ANALYSIS

In this section we will first analyse the performances of the mainstream drl methods (Section 4.4), and afterwards attempt to understand why the VLM based method performed so poorly (Section 4.5).

4.4 ANALYSIS MAINSTREAM DRL METHODS

This section will focus on analysing the performances of the standard DQN and the Mini-Rainbow. Before delving into the actual analysis it is worth pointing out that these results were expected. The Atari-100K is a challenging benchmark and both the standard DQN and the rainbow variant thereof are rather data hungry models, usually requiring on the order of hundreds of thousands of episodes, rather than 100k steps. Thus, them strongly underperforming the MuZero method is no surprise.

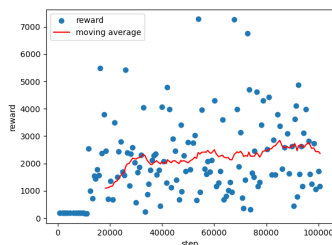


Figure 1: The performance of the standard DQN method across the 100k steps visualized as a scatter plot with a superimposed moving-average line (n=25).

Figure 1 shows the performance of the standard DQN method across the 100k steps. As can be seen, the first 12 steps exhibit a very poor performance, which is because the first few games are solely used to populate the experience replay and no training is happening. Subsequently the performance shoots up to the low thousands, and only marginally increases throughout the rest of training (which can be seen by the slight slope exhibited by the moving average line). With substantially more environment steps available for training, we would expect the performance to continuously increase.

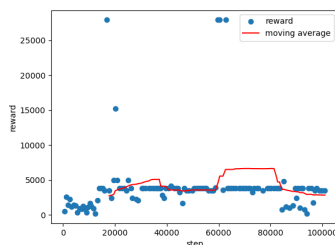


Figure 2: The performance of MiniRainbow-base method across the 100k steps visualized as a scatter plot with a superimposed moving-average line (n=25).

As can be seen in Figure 2, the MiniRainbow model follows a similar initial pattern where the first 12 games have a very lower performance (since they are before the first training iteration) and subsequently the performance goes up. Interestingly enough, this model exhibits periods where the performance goes all the way up to $> 25,000$, but unfortunately decreases again subsequently. The reason for why it stays at this elevated level for a number of steps before decreasing is that the target net is updated periodically. We hypothesize that the reason for the irregular performances are our hyperparameters, which, because of the few training steps available, we choose to be quite aggressive (thus the model might pick up on some patterns rapidly, but also runs the risk of overwriting/forgetting them).

4.5 ANALYSIS VLM

As can be seen in Table 1, the VLM approach we proposed in Section 3.4 performed slightly worse than expected. Whilst it did manage to outperform the standard DQN and our MiniRainbow method, it came nowhere near the performance of MuZero. One reason for this could simply be the fact that we are using the VLM in a Zero-shot setting, whilst we are actually training the other methods on 100k environment steps. An alternative hypothesis could be that the model struggles with properly

recognizing objects in the environment. The reason we are thinking this is because when observing the model play, the strategy seems good (it focuses on staying at the bottom and far away from sprites), but the aim seems off. So much so that sometimes it will shoot on the left side of the screen, whilst the centipede is on the right hand side. Naturally a multitude of reasons can cause this, but in this section we focus on the complexity of properly captioning frames from the atari game. This is challenging because of the un-common resolution of (84, 84), which is substantially smaller than the training images for most SoTA VLMs, and the abstraction of the objects (in atari video games, most sprites are represented as 8-bit pixelart, which can make it rather hard to recognize what that object actually is). To test whether this is indeed a problem, we pass a single frame (Figure 3) to Fuyu and task the model with coming up with a caption. Unfortunately, Fuyu only returns a single new-line character and then the end-of-sequence token.

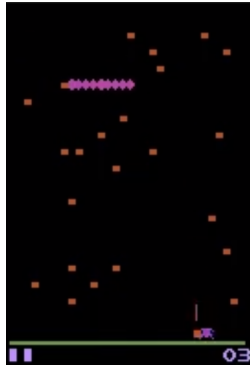


Figure 3: A game-play frame from Centipede.

Even though Fuyu is amongst the best open-source models, it’s performance is still no where near that of GPT-4(v). Thus, to check whether scale and the related emergent capabilities are the problem, we passed a screenshot of the same frame (Figure 3), with the task of finding a detailed caption, to GPT-4(V). Before showing the results it is worth pointing out that even if GPT-4(V) is capable of accurately captioning the frame, it would unfortunately not be a viable option for gameplay, because of the exorbitant cost. Nevertheless, this would be an encouraging result, as open-source models tend to match the performance of their closed-source cousins within 6-12 months. GPT-4(V) caption: *“The image appears to be a screenshot from a classic arcade-style video game, likely from the early era of video games judging by its pixelated graphics and simple color palette. The game is set against a black background, which is typical for many early video games due to the limitations of display technology at the time. At the top, there is a pink object composed of several joined pink squares, which could represent an enemy or an obstacle that the player must avoid or destroy. Below this pink object, there is a field of individual orange pixels scattered across the screen in a seemingly random pattern. These could represent projectiles, stars, or some other form of in-game elements that the player must navigate through or interact with. At the bottom of the screen, there is a purple object with a light green base. This object is distinctly separate from the rest and has a form that suggests it could be the player’s character or vehicle, which can move horizontally along the bottom of the screen. The purple and green colors are used to differentiate it from the rest of the elements. There are two indicators in the bottom right corner: a double-line symbol and the number 03. These could be part of the game’s user interface, possibly indicating the remaining lives, a score, or some other game-related metric.”*

Obviously, this caption is infinitely better than that generated by Fuyu. It is also worth pointing out that even though some information is still missing, this might very well be because of the ‘compression’ to natural text. These results are encouraging and suggest that more capable open-source models could in fact be able to perform well on the Atari-100K benchmark.

Beyond waiting for the development of more robust open source models (though such anticipation may prove unnecessary given the recent release of CogVLM (Wang et al., 2023b), which unfortunately happened after we completed this project), we recommend another strategy for dealing with this issue. Since the vision module is clearly at least part of the problem, we could address this by fine-tuning it on data that is more similar to the Atari suit. The easiest way of getting such training data is to simply generate 8-bit sprites of a plethora of objects using a fine-tuned diffusion model and a neural

style transfer model. Subsequently, a number of these objects can be randomly placed on an (84, 84) canvas, and the model is task with not just identifying the objects correctly, but also returning their positions (only the vision module weights should be fine-tuned for this).

5 CONCLUSION

In this work, we assessed the efficacy of several prominent DRL approaches and a VLM-based agent in mastering the Atari 2600 game, Centipede, within a limited data context (100k steps). As expected, we found that MuZero surpassed both the conventional DQN and our modified 'mini-Rainbow' by a large margin. The VLM-based agent on the other hand was able to outperform both the standard DQN and our MiniRainbow version, but unfortunately came nowhere near the performance of MuZero. In our analysis section we explore the reason for this shortcoming, namely the vision component of the model, and offer two approaches for resolving this: waiting for stronger VLM base models or generating synthetic 8-bit training data using a fine-tuned diffusion/neural style transfer model.

REFERENCES

- a. URL https://www.gamesdatabase.org/media/system/atari_2600/manual/formated/centipede_-_1982_-_atari.pdf.
 - b. URL https://atariage.com/manual_html_page.php?SoftwareID=911.
- Ademi Adeniji, Amber Xie, Carmelo Sferrazza, Younggyo Seo, Stephen James, and Pieter Abbeel. Language reward modulation for pretraining reinforcement learning, 2023.
- Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022.
- Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. Agent57: Outperforming the atari human benchmark. *CoRR*, abs/2003.13350, 2020. URL <https://arxiv.org/abs/2003.13350>.
- Rohan Bavishi, Erich Elsen, Curtis Hawthorne, Maxwell Nye, Augustus Odena, Arushi Somani, and Sağnak Taşlılar. Introducing our multimodal models, 2023. URL <https://www.adept.ai/blog/fuyu-8b>.
- 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. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *CoRR*, abs/2010.11929, 2020. URL <https://arxiv.org/abs/2010.11929>.

- Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models, 2023.
- Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL <http://arxiv.org/abs/1710.02298>.
- Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL <http://arxiv.org/abs/1502.03167>.
- Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models, 2023.
- Kevin Lin, Faisal Ahmed, Linjie Li, Chung-Ching Lin, Ehsan Azarnasab, Zhengyuan Yang, Jianfeng Wang, Lin Liang, Zicheng Liu, Yumao Lu, Ce Liu, and Lijuan Wang. Mm-vid: Advancing video understanding with gpt-4v(ision), 2023.
- Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2023.
- Michaelnny. Michaelnny/muzero: A pytorch implementation of deepmind’s muzero agent. URL <https://github.com/michaelnny/muzero>.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL <http://arxiv.org/abs/1312.5602>.
- Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL <http://arxiv.org/abs/1602.01783>.
- Thomas Pouncy, Samuel Gershman, Pedro Tsividis, Jacqueline Xu, and Joshua Tenenbaum. Human learning in atari. 01 2017.
- Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning, 2023.
- Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay, 2016.
- Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy P. Lillicrap, and David Silver. Mastering atari, go, chess and shogi by planning with a learned model. *CoRR*, abs/1911.08265, 2019. URL <http://arxiv.org/abs/1911.08265>.
- Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.
- Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models, 2022.
- Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015. URL <http://arxiv.org/abs/1509.06461>.
- Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models, 2023a.
- Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models, 2023b.

Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. *CoRR*, abs/1511.06581, 2015. URL <http://arxiv.org/abs/1511.06581>.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh Murthy, Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization, 2023.

A HOW TO RUN THE CODE

For ease of replication of all of our results, in this section we offer a step-by-step guide for executing our code. Generally speaking, all four models follow a similar setup. Once the code has been loaded onto an ubuntu server, first all necessary dependencies need to be installed

```
bash install_necessary.sh (1)
```

which will also install all necessary python libraries. This has to be done for each of the four models individually, as they rely on different libraries.

Subsequently, the main code can be executed using:

```
python main.py (2)
```

If you wish to adjust any hyperparameters, please directly edit the .yaml file in the config folder.

B THE MANUAL

In this section we will explain in detail how the modified version of the original Centipede manual was created.

The end goal is to have a single text document that describes all aspects of the game well enough that we can use GPT-4 to extract all relevant information. To that end, we start with the text transcript (cen, b) of the original centipede pdf manual (cen, a). While the transcript is well-done, it lacks explanations for the various figures found in the original manual. To address this, we extracted all relevant figures from the original manual and generated detailed captions for each one using GPT-4(V) (see Figure 4 for an example of this).

There are a total of 9 relevant figures in the manual. To caption each one, we utilize the following prompt: *The above is a screenshot of a figure in the atari 2600 Centipede manual. I want to use the text transcription of the manual to better train AI agents, and to that end, need an as specific as possible caption of the image. Please consider be as specific in terms of location, description, and usefulness for actually playing the game as possible..*

It is noteworthy that as depicted in Figure 4 GPT-4(V) is capable of generating decent captions for the game frame. As we'll discover later, these captions surpass those generated by the current leading open-source models by a significant margin.

Subsequently, the text manual and captioned images are fed to a standard GPT-4 model, which is tasked with extracting all relevant information using the following prompt: *[text text based manual] Above is the game manual for the Atari 2600 game Centipede. I want to use this game manual to teach an AI how to play the game. Please generate a minimal description of how each object in the game works, what the observations look like, and generally what the purpose of the game is and what to look out for. Make sure to be brief and describe how all relevant objects look..* The resultant game manual summary can be found in the *Centipede.txt* file and is used as input for the VLM-based agent.

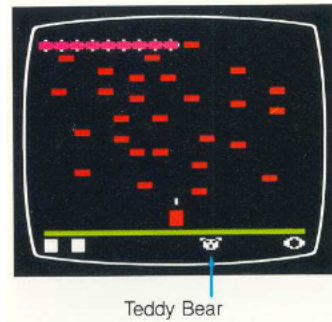


Figure 4: GPT-4(V) caption: A digital illustration depicts a simplified representation of the 'Centipede' gameplay screen for the Atari 2600. The screen is framed within a stylized, black monitor border. At the top, a series of magenta blocks, possibly representing the centipede, moves through a field of red, rectangular obstacles or 'mushrooms' scattered across the middle of the play area. The bottom of the screen features two white rectangular shapes on either side, likely representing the player's area and the zone from which the player can shoot. In the center bottom, there is a small, white line with a label pointing to it that reads 'Teddy Bear,' which might be indicating the player's character or avatar.