Knowledge Distillation to a Reinforcement Learning Network from a Hierarchical GPT-4 Agent using Procedural Data Augmentation

Creighton Glasscock
creiglas@umich.edu

1 Introduction

The recent release of GPT-4 has opened the door to applying large language models to complex tasks that are adjacent to or incorporate natural language. As a result of this recent development, we are increasingly motivated to apply these recent advances in large language models, namely GPT-4, to the task of empowering real-world embodied agents. In this work, we develop a multi-tier GPT-4 agent to solve tasks in the Minigrid benchmark from only a brief description of the task (zero-shot) by encouraging the verbalization of subgoals that emerge during decision-making. We then apply a novel data augmentation strategy over these information-rich subgoals to generate, from just a few episodes of play by the multi-tier GPT-4 agent, enough unique quality trajectories to train a successful reinforcement learning from scratch using imitation learning. This agent can then be fine-tuned with online reinforcement learning much quicker than without the imitation learning step. This enables efficient learning in tasks where bootstrapping policy and value functions during online training would otherwise be highly sample inefficient due to sparse rewards. In addition, we apply this approach to use the multi-tier GPT-4 agent to extract sub-goals from episodes of human play on complex tasks which cannot be solved by the multi-tier GPT-4 agent alone, thus extending the novel sub-goal-based data augmentation strategy to episodes originated by skilled humans or an optimal policy, thereby drastically reducing the number of real expert trajectories needed for offline training of the reinforcement learning agent.

2 Setting

Minigrid is a simulated 2D gridworld environment built on the Python OpenAI Gym library, which provides infrastructure for the development of environments for reinforcement learning benchmarks. The complexity of the Minigrid benchmark varies between environments, ranging from low complexity in environments such as EmptyEnv, where the agent's only goal is to get to the goal tile in a small enclosed space without any obstacles or distractors, to BossRoom, where the agent needs to navigate multiple rooms to complete a complex series of interaction tasks in the correct order. Like other environments designed for reinforcement learning, each environment provides an action space, from which the agent samples its selected action each timestep and an observation space, from which the environment samples the observation conveyed to the agent each timestep.

Given that Minigrid is a gridworld environment, the action space is discrete. The agent has access to three actions for navigation – the agent can select "left" or "right", which rotate the agent 45 degrees in either direction, or "forward", which moves the agent one tile in the direction it is facing. In addition, the agent has access to three actions for interaction – the agent can select "pickup" to pick up an object in the same tile as the agent, or "toggle" to interact with a tile adjacent to it, for example, in order to open a closed door. The formulation of the Minigrid benchmark is egocentric; in other words, the agent must control the agent from the agent's perspective, managing the rotation of the agent and moving the agent only in the direction it is facing. This is in contrast to an allocentric approach to environment manipulation, in which the agent can be manipulated in any of the four

cardinal directions at any time, agnostic of the direction the agent is facing at that time. The egocentric formulation of the Minigrid benchmark adds an extra layer of complexity to navigational tasks which may be a roadblock for some agents.

The Minigrid environment is partially observable, and thus, the agent only receives information about the tiles a short range in front of it each timestep As such, the agent may need to perform some initial exploration of the environment before it finds the location of certain objects or locations crucial to the completion of its initial instructional task.

Unlike some traditional reinforcement learning benchmarks, in addition to the action and observation spaces, the environment also provides a mission space, which enumerates all possible instructions which can be given to an agent at the start of an episode. The type of instruction generated can vary the simple instruction in the EmptyEnv environment, "get to the goal square," to more complex instruction such as those in the PlaygroundEnv environment, e.g. "pick up the gray ball next to the yellow box, then open the green door". These more complex instructions are composed of phrases generated by combining an action, such as "reach" or "pick up", with an object, such as "goal" or "key". These objects can be further described with location descriptors, such as "next to <other object>", or color descriptors. These basic phrases can then be compounded with each other with words such as "and" or "then". The inclusion of this natural language mission makes this environment ripe for approaches that incorporate understanding of language, particularly recent LLMs such as GPT-4, which display the general ability to break down and plan over such tasks.

For the reward function, the environment uses a variation of Success weighted by Path Length (SPL) in order to incentivize quick and efficient exploration of the level. The reward is formulated as follows:

$$R = \begin{cases} 1 - 0.9 * \frac{N}{M} & \text{for success} \\ 0 & \text{for failure} \end{cases}$$

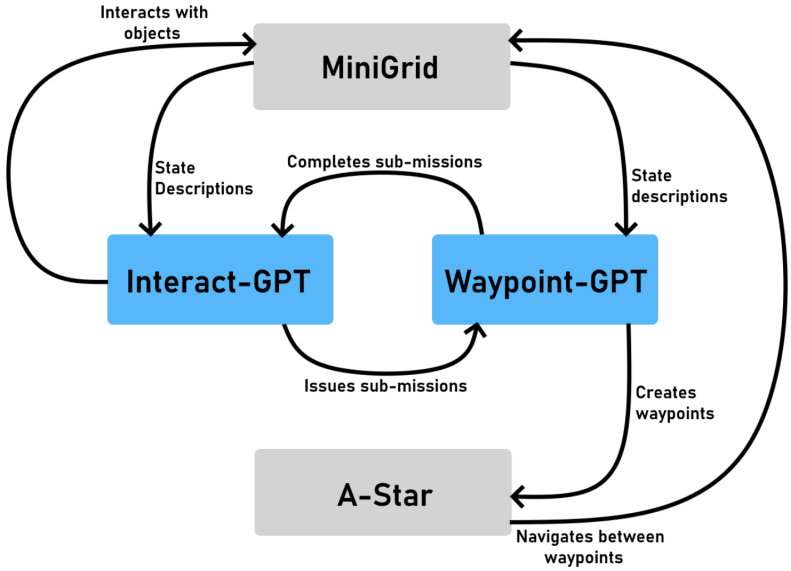
where N represents the current number of timesteps, and M represents the maximum number of timesteps before timeout, which is set to 100 by default. As such, the Minigrid environment features highly sparse reward. As a result, a reinforcement learning agent can be particularly difficult to train for the more complex tasks in this benchmark, where is it quite difficult to stumble upon a solution by random chance, until which point there is no reward with which to bootstrap the policy and value functions.

3 Multi-Tier GPT Agent

3.1 Approach

For this task, we construct an LLM agent using GPT-4. This agent is multi-tiered, consisting of two GPT-4 sub-agents, with different tasks abstracted to each of the two sub-agents. The interaction sub-agent interacts directly with the environment, and serves to decompose missions from the Minigrid environment into sub-goals. The current sub-goal is communicated to the waypointing sub-agent, and each sub-goal must end with the interaction or navigation to a given type of object. The waypointing sub-agent receives a prompt containing the sub-goal and a textual description of the state. This agent engages in step-by-step reasoning with the goal of placing short-term waypoints, one at a time, to guide the navigation through the environment. Once a waypoint is identified, the A pathfinding algorithm is used to identify the atomic actions needed to navigate the agent to that position. The waypointing sub-agent must place waypoints to navigate the agent to the object of that type, or explore the environment to discover it if it has not yet been found. The waypointing agent is prompted until the object has been found and the waypointing sub-agent places a waypoint on the object pertinent to the sub-goal; when a waypoint is placed on an object, prompting is then re-directed to the interaction sub-agent, which is tasked with identifying the correct type of interaction command for that object, deciding whether the sub-goal has been satisfied, and if it has, identifying the next sub-goal for the waypointing sub-agent to complete. This loop continues until the end of the episode.

Figure 1: An illustration of the construction of the multi-tier GPT-4 agent.



3.2 Additional Techniques

We used several additional techniques to manage and support the decision-making of the multi-tier GPT-4 agent.

1. **Border points.** In order to promote exploration by the waypointing sub-agent of hitherto unseen areas of the environment, we identify "border points" in the state description. We define "border points" as observed points which border any unobserved points. GPT-4 is not capable of identifying such points from a state description, but the knowledge of these points increases exploration, decreasing the unnecessary retreading of already explored areas and decreasing the average number of timesteps needed to solve exploration-heavy tasks.
2. **State description compression.** In order to reduce unnecessary token usage, we compress nearby cells of the same type together in the state description. For a group of cells that are vertically adjacent: for example, the sequence of coordinates (0,0), (0,1), (0, 2), (0,3) is compressed to the representation (0,0)-(0,3). This compression strategy is explicitly communicated to the waypointing sub-agent in the system prompt, and reduces average token usage per timestep by approximately 30% with no drop in average performance on each task.
3. **State description citation.** When the waypointing sub-agent to identify its next waypoint, we encourage it to list a few likely coordinate candidates and weight the benefits and drawbacks of each. In order to encourage GPT-4 to correctly identify the type of cell at a given point, we encourage via prompting the citation of coordinates in the given state description. For example, if the waypointing sub-agent is considering placing a waypoint at (3,3), we encourage it to then cite the portion of the state description where that cell occurred, perhaps "Open space: ... (3,2)-(3,4)". This citation strategy reduces the frequency of hallucinations in which GPT-4 misidentifies the type of cell at a given coordinate.
4. **Observation overlaying.** GPT-4 often fails to readily recall non-recent information in large contexts, which is problematic for this application. As such, for complex tasks, we save and overlay the arrays corresponding to the observation for each timestep, then convert this composed observation into the state description for prompt to GPT-4. This overlaid observation strategy prevents performance drop-off for long episodes and enables more ready citation of cells from earlier observations.
5. **Relative position tracking.** The absolute position and direction of the agent in the environment are not included in the observation received by the agent. Instead, the coordinates

of all observed cells within the agent's field of view are given with respect to the agent's position at that current timestep. As such, we have need for a common identification of a given cell across timestep where the agent is in a different position. As such, we represent all cells in the state description as relative to the agent's *initial* position and direction, which we assume to be (0,0) and North, respectively. This solution solves the aforementioned problem, allowing the use of the above strategies with minimal computational overhead.

6. **Waypointing sub-agent interruption.** In order to prevent unnecessary continued navigation between waypoints by the A pathfinding algorithm when new relevant information is discovered in the environment, we interrupt the navigation by A to prompt the waypointing sub-agent for a new waypoint whenever the observation for that timestep includes a cell of a type not yet observed by the agent. For example, we prompt the waypointing sub-agent the exact timestep it first discovers the goal cell, even if the agent is currently navigating to a waypoint elsewhere.
7. **Waypoint and mask rendering.** In addition, in order to visualize the waypoints selected by the waypointing sub-agent, we edited the source code of the Minigrid environment to render a red circle at the location of the current waypoint at each timestep. Furthermore, during the rendering process, we mask each cell which has not yet been observed by the agent in order to visualize the contents of the overlaid observation available to the agent.

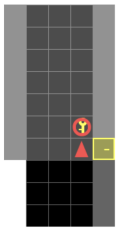


Figure 2: A timestep of play in the Minigrid DoorKey 16x16 task.

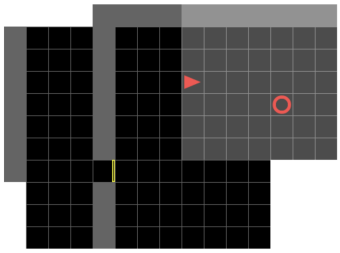


Figure 3: A later timestep of play in the same episode.

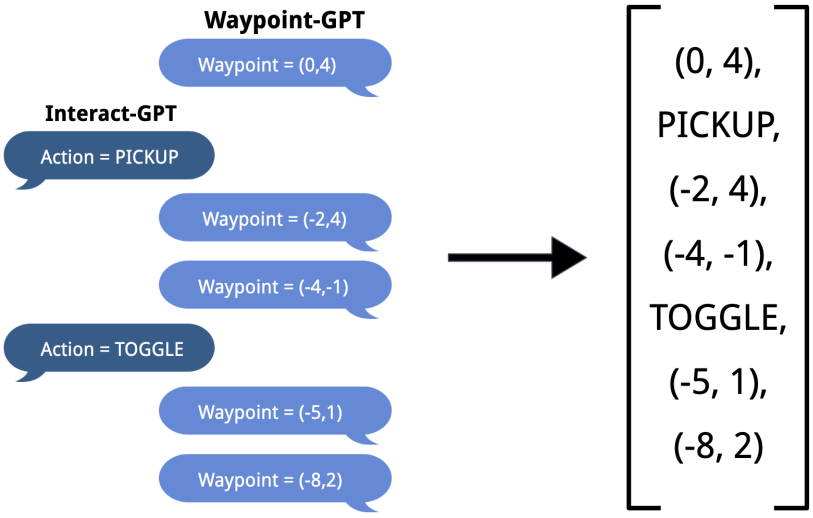
As an example, in Figure 2, we see that the waypointing sub-agent has placed a waypoint on the yellow key, which passes control to the interaction sub-agent to pick up the key and subsequently decide the next sub-goal. Figure 3 shows the waypointing sub-agent using waypoints to explore the remainder of the environment to discover the goal cell. Note the expansion of the overlaid observation as shown by the cells no longer obscured by the white mask.

4 Data Augmentation and Training Pipeline

The multi-tier GPT-4 agent outlined above is capable of solving many Minigrid tasks zero-shot. We are motivated to use the multi-tier GPT-4 agent to accelerate the training of a reinforcement learning agent using Proximal Policy Optimization (PPO). A reinforcement learning agent requires a large number of samples to learn a successful policy, but has been shown to be capable of successfully learning each task in the Minigrid benchmark. Testing an already trained reinforcement learning agent to solve an environment is much less computationally and financially expensive than solution with a GPT-4 based agent. As such, by using the zero-shot expertise of our multi-tier GPT-4 agent to bootstrap the learning of a reinforcement learning agent, we can obtain the benefits of both approaches. To this end, we apply a novel data augmentation strategy over the waypoints generated by the play of the multi-tier GPT-4 agent over an episode of a given task, which we use to train a policy offline using imitation learning, then fine-tuning online with reinforcement learning.

During an episode of play by the multi-tier GPT-4, we record and preserve the order of the interactions from the interaction sub-agent and waypoints from the waypointing sub-agent, as shown in Figure 4.

Figure 4: An illustration of the beginning of the data augmentation process.



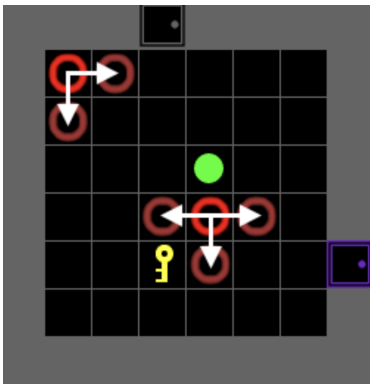


Figure 5: An illustration of the waypoint perturbation process for data augmentation.

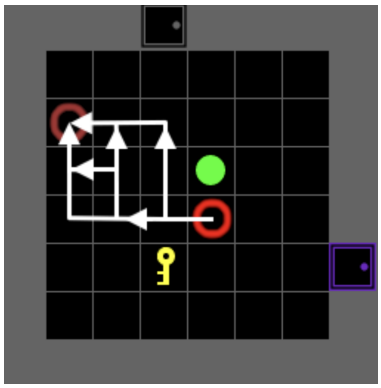


Figure 6: An illustration of the shortest-paths permutation process for data augmentation.

Then, for all waypoints not immediately preceding an interaction (in other words, the waypoints used for exploration, not exploration), we perturb those waypoints to each adjacent open cell, preserving the general location of the waypoint but altering the exact position slightly, as seen in Figure 5. Next, we take each permutation of successive waypoint pairs from these perturbed waypoints by picking one location from the list of waypoints perturbed from each original waypoint. Then, for each pair of waypoints, we apply Dijkstra's algorithm to consider every possible path of shortest length between those two waypoints, as seen in Figure 6. We then take every possible permutation of shortest paths for each possible permutation of pairs of successive pairs of perturbed waypoints. Then, for each permutation, we convert the sequences of paths between waypoints into a sequence of actions for the duration of an episode. Then, we run this list of actions back through the Minigrid environment for the same seed as the original task in order to recover a new, unique trajectory. The reward of this new trajectory will be within approximately 5% of the reward of the original episode. On average, this process generates 20,000 unique, similar-quality episodes from the output of the multi-tier GPT-4 agent over a single episode. Tasks that are longer and require more exploration (and thus result in a greater number of perturbed waypoints) are those that result in a larger number of augmented episodes.

As such, we need only to solve on the order of 10 different tasks with the multi-tier GPT-4 agent in order to automatically augment an average of 20,000 unique, high-quality trajectories, which is

sufficient for the successful offline training of a policy via imitation learning. Note that, during the imitation learning process, it is necessary to randomize the order of the trajectories. Since the trajectories are generated from so few episodes, if they are stratified in the order of the original episode they each were each generated from, then the policy overfits to the most recent seed, requiring unlearning and worsening the policy. After the imitation learning step, we fine-tune the policy online via PPO for 2.5 million timesteps. The results of this process are shown in Figures 7, 8, and 9.

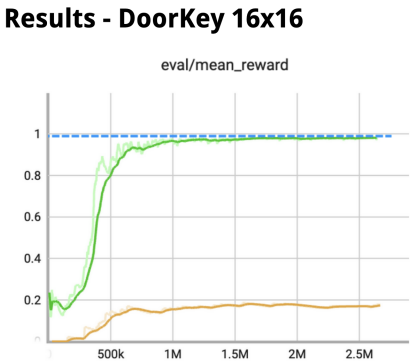


Figure 7: A comparison of evaluation reward over 2.5M steps of fine-tuning on the DoorKey 16x16 task.

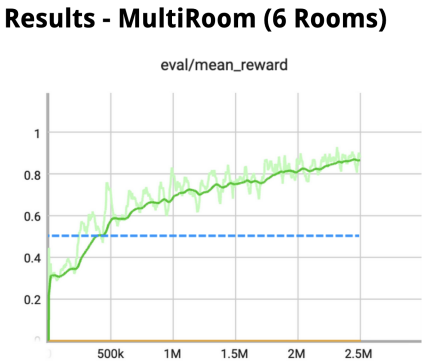


Figure 8: A comparison of evaluation reward over 2.5M steps of fine-tuning on the MultiRoom task.

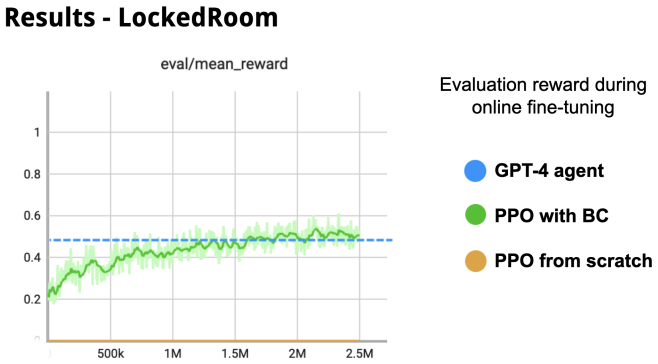


Figure 9: A comparison of evaluation reward over 2.5M steps of fine-tuning on the LockedRoom task.

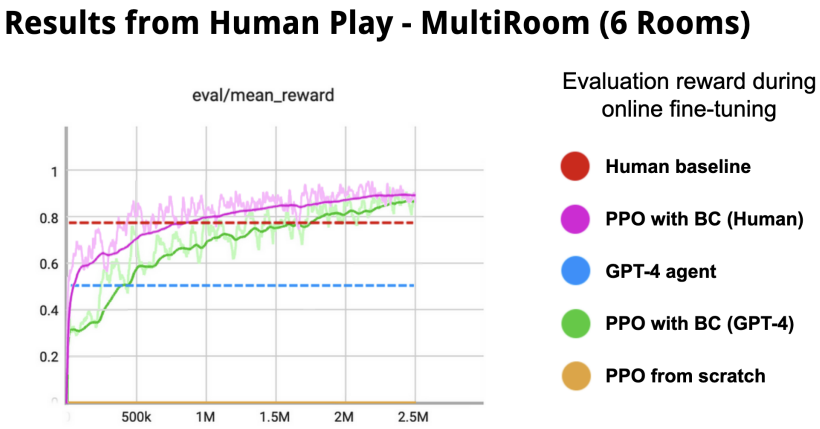
In these graphs, we compare the average zero-shot reward attained by the multi-tier GPT-4 agent over ten episodes of the task (in blue) against the evaluation reward during the online fine-tuning with PPO on a policy pre-trained by imitation learning on trajectories augmented from those ten original episodes (in green). We also compare this performance against that of a PPO agent trained online from scratch (in orange). We see that, as is expected following pre-training on quality trajectories, the training curve of the pre-trained PPO agent significantly outperforms that of the base PPO agent. Notably, this process is entirely automatic, with no need for human-generated trajectories anywhere in the pipeline (which are infeasible to collect in the number needed for pre-training) or any episodes generated by a pre-existing optimal policy (which is not available in many settings). Note that, in complex tasks where the multi-tier GPT-4 agent performs worse, the fune-tuning on the pre-trained PPO model sees less of an advantage over the base PPO model, as the generated trajectories for pre-training are lower-quality.

Each of the tasks in these figures (DoorKey 16x16, MultiRoom, and LockedRoom), are very challenging for a reinforcement learning agent to solve alone due to highly sparse reward. Notably, in the MultiRoom and LockedRoom tasks, the base PPO is unable to consistently obtain any non-zero reward, thus highlighting the degree to which pre-training on our automatically augmented trajectories is successful in bootstrapping the policy,

5 Augmenting From Human Trajectories

For complex tasks, the multi-tier GPT-4 agent often achieves lower zero-shot performance. The agent is still capable of solving these tasks on occasion, but the navigation tends to be less optimal, thus reducing the reward. This lowered reward on the original episodes used for data augmentation reduces the reward of the augmented episodes, and thus the data for offline training is lower-quality. This hampers the learning of the policy in the offline phase, thus requiring more effective fine-tuning in the online phase. One workaround is to use human-generated trajectories for offline training. However, since offline training requires hundreds of thousands of *unique* trajectories, generating all episodes directly from human play is not feasible. Our data augmentation strategy is not directly compatible with human trajectories, since, unlike those generated by the multi-tier GPT-4 agent, human-generated episodes do not contain the rich information contained within the waypoints and sub-goals generated by the multi-tier GPT-4 agent which are used for data augmentation. As such, we opt to adapt the multi-tier GPT-4 agent to interpret human-generated episodes to infer the required rich waypoint information. As such, we can largely recycle the prior pipeline by collecting 10 episodes from human play, applying the multi-tier GPT-4 agent to interpret those episodes, apply our data augmentation strategy over these episodes to generate hundreds of thousands of unique, similar-quality episodes as before, then continuing with the offline imitation learning and online reinforcement learning fine-tuning pipeline. Initial results (in Figure 10) show that this pipeline, when applied to high-quality human-generated human trajectories on complex tasks where the multi-tier GPT-4 agent performs lower than human performance, results in accelerated training compared to the same pipeline repeated over episodes on the same tasks generated only by the multi-tier GPT-4 agent.

Figure 10: A comparison of fine-tuning performance of the pipeline applied to human-generated episodes.



6 Future Work

1. **Extending Human Trajectory Collection.** In order to collect further results on imitation learning from the the novel data augmentation process over human-generated trajectories, we plan to expand the collection of data from human participants on complex Minigrid tasks.
2. **Identification of Task Complexity.** In this work, we frequently reference the complexity of a task to delineate those which can be easily solved zero-shot by the multi-tier GPT-4 agent, and those on which the multi- GPT-4 agent cannot meet human performance. As such, it is pertinent to propose a metric of complexity across the tasks in the Minigrid benchmark. Possible candidates for such a metric are some combination of the size of the environment, the number of rooms in the environment, the number of unique objects or distractors in the environment, the maximum step count for the task, or the number of clauses in the mission.
3. **Comparison to open-source LLMs.** GPT-4 is closed-source and closed-access. Prompting the model is expensive and there is no available insight into the workings of the model at

inference time. Competitive open-access models have continued to improve as of late, and as such, an avenue of investigation should be to compare the performance of our pipeline with the GPT-4 API against the performance of our pipeline with the API of an open-access model, such as Llama-2.

4. **Offline Reinforcement Learning.** Currently, we use imitation learning for the offline learning step of the pipeline. Compared to imitation learning, offline reinforcement learning has the possibility to be faster and more stable, and, in addition, might further accelerate the online fine-tuning portion of the training, since the policy and value portions of the network will have the opportunity to directly share weights between the offline and online steps.