In-Context Policy Iteration

Ethan Brooks[1], **Logan Walls**[2], **Richard L. Lewis**[2], **Satinder Singh**[1]
[1]Computer Science and Engineering, University of Michigan
[2]Department of Psychology, University of Michigan
{ethanbro,logwalls,rickl,baveja}@umich.edu

Abstract

This work presents In-Context Policy Iteration, an algorithm for performing Reinforcement Learning (RL), in-context, using foundation models. While the application of foundation models to RL has received considerable attention, most approaches rely on either (1) the curation of expert demonstrations (either through manual design or task-specific pretraining) or (2) adaptation to the task of interest using gradient methods (either fine-tuning or training of adapter layers). Both of these techniques have drawbacks. Collecting demonstrations is labor-intensive, and algorithms that rely on them do not outperform the experts from which the demonstrations were derived. All gradient techniques are inherently slow, sacrificing the "few-shot" quality that made in-context learning attractive to begin with. In this work, we present an algorithm, ICPI, that learns to perform RL tasks without expert demonstrations or gradients. Instead we present a policy-iteration method in which the prompt content is the entire locus of learning. ICPI iteratively updates the contents of the prompt from which it derives its policy through trial-and-error interaction with an RL environment. In order to eliminate the role of in-weights learning (on which approaches like Decision Transformer rely heavily), we demonstrate our algorithm using Codex (Chen et al., 2021b), a language model with no prior knowledge of the domains on which we evaluate it.

1 Introduction

In-context learning describes the ability of sequence-prediction models to generalize to novel downstream tasks when prompted with a small number of exemplars (Lu et al., 2021; Brown et al., 2020). The introduction of the *Transformer* architecture (Vaswani et al., 2017) has significantly increased interest in this phenomenon, since this architecture demonstrates much stronger generalization capacity than its predecessors (Chan et al., 2022). Another interesting property of in-context learning in the case of large pre-trained models (or "foundation models") is that the models are not directly trained to optimize a meta-learning objective, but demonstrate an emergent capacity to generalize (or at least specialize) to diverse downstream task-distributions (Brown et al., 2020; Wei et al., 2022a).

A litany of existing work has explored methods for applying this remarkable capability to downstream tasks (see Related Work), including Reinforcement Learning (RL). Most work in this area either (1) assumes access to expert demonstrations — collected either from human experts (Huang et al., 2022a; Baker et al., 2022), or domain-specific pre-trained RL agents (Chen et al., 2021a; Lee et al., 2022; Janner et al., 2021; Reed et al., 2022; Xu et al., 2022). — or (2) relies on gradient-based methods — e.g. fine-tuning of the foundation models parameters as a whole (Lee et al., 2022; Reed et al., 2022; Baker et al., 2022) or newly training an adapter layer or prefix vectors while keeping the original foundation models frozen (Li & Liang, 2021; Singh et al., 2022; Karimi Mahabadi et al., 2022).

Our work presents an algorithm, In-Context Policy Iteration (ICPI) which relaxes these assumptions. ICPI is a form of policy iteration in which the prompt content is the locus of learning. Because our method operates on the prompt itself rather than the model parameters, we are able to avoid gradient methods. Furthermore, the use of policy iteration frees us from expert demonstrations because suboptimal prompts can be improved over the course of training.

We illustrate the algorithm empirically on six small illustrative RL tasks— *chain, distractor-chain, maze, mini-catch, mini-invaders*, and *point-mass*—in which the algorithm very quickly finds good

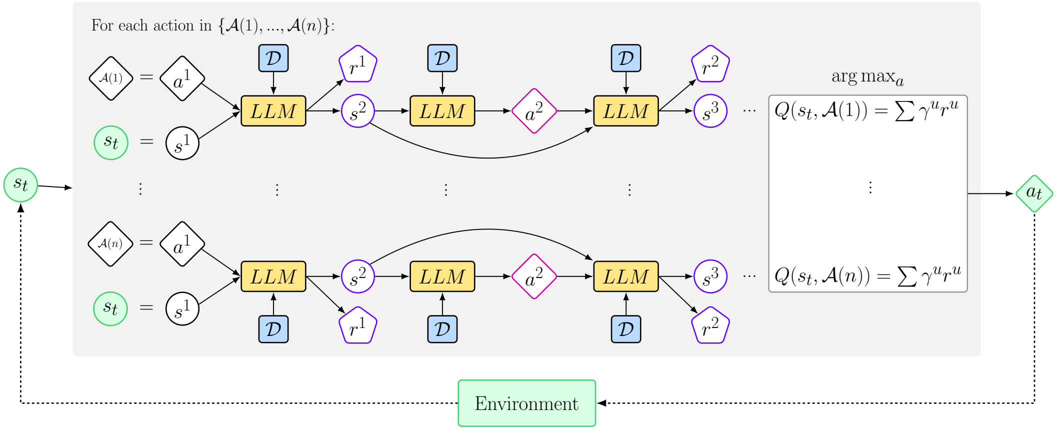
For each action in $\{\mathcal{A}(1), ..., \mathcal{A}(n)\}$:
$\mathcal{A}(1)$ = a^1
s_t = s^1
$\mathcal{A}(n)$ = a^1
s_t = s^1
$\mathcal{D}$
LLM
r^1
s^2
a^2
s^3
r^2
arg max$_a$
$Q(s_t, \mathcal{A}(1)) = \sum \gamma^u r^u$
$Q(s_t, \mathcal{A}(n)) = \sum \gamma^u r^u$
s_t
a_t
Environment

Figure 1: For each possible action $\mathcal{A}(1), \ldots, \mathcal{A}(n)$, the LLM generates a rollout by alternately predicting transitions and selecting actions. Q-value estimates are discounted sums of rewards. The action is chosen greedily with respect to Q-values. Both state/reward prediction and next action selection use trajectories from $\mathcal{D}$ to create prompts for the LLM. Changes to the content of $\mathcal{D}$ change the prompts that the LLM receives, allowing the model to improve its behavior over time.

policies. We also compare five pretrained Large Language Models (LLMs), including two different size models trained on natural language—OPT-30B and GPT-J—and three different sizes of a model trained on program code—two sizes of Codex as well as InCoder. On our six domains, we find that only the largest model (the `code-davinci-001` variant of Codex) consistently demonstrates learning.

2 Related Work

A common application of foundation models to RL involves tasks that have language input, for example natural language instructions/goals (Garg et al., 2022a; Hill et al., 2020) or text-based games (Peng et al., 2021; Singh et al., 2021; Majumdar et al., 2020; Ammanabrolu & Riedl, 2021). Another approach encodes RL trajectories into token sequences, and processes them with a foundation model, model representations as input to deep RL architectures (Li et al., 2022; Tarasov et al., 2022; Tam et al., 2022). Finally, a recent set of approaches treat RL as a sequence modeling problem and use the foundation models itself to predict states or actions. This section will focus on this last category.

2.1 Learning from Demonstrations

Many recent sequence-based approaches to reinforcement learning use demonstrations that come either from human experts or pretrained RL agents. For example, Huang et al. (2022a) use a frozen LLM as a planner for everyday household tasks by constructing a prefix from human-generated task instructions, and then using the LLM to generate instructions for new tasks. This work is extended by Huang et al. (2022b). Similarly, Ahn et al. (2022) use a value function that is trained on human demonstrations to rank candidate actions produced by an LLM. Baker et al. (2022) use human demonstrations to train the foundation model itself: they use video recordings of human Minecraft players to train a foundation models that plays Minecraft. Works that rely on pretrained RL agents include Janner et al. (2021) who train a "Trajectory Transformer" to predict trajectory sequences in continuous control tasks by using trajectories generated by pretrained agents, and Chen et al. (2021a), who use a dataset of offline trajectories to train a "Decision Transformer" that predicts actions from state-action-reward sequences in RL environments like Atari. Two approaches build on this method to improve generalization: Lee et al. (2022) use trajectories generated by a DQN agent to train a single Decision Transformer that can play many Atari games, and Xu et al. (2022) use a combination of human and artificial trajectories to train a Decision Transformer that achieves few-shot generalization on continuous control tasks. Reed et al. (2022) take task-generality a step farther and use datasets generated by pretrained agents to train a multi-modal agent that performs a wide array of RL (e.g. Atari, continuous control) and non-RL (e.g. image captioning, chat) tasks.

Some of the above works include non-expert demonstrations as well. Chen et al. (2021a) include experiments with trajectories generated by random (as opposed to expert) policies. Lee et al. (2022) and Xu et al. (2022) also use datasets that include trajectories generated by partially trained agents in addition to fully trained agents. Like these works, our proposed method (ICPI) does not rely on expert demonstrations—but we note two key differences between our approach and existing approaches. Firstly, ICPI only consumes self-generated trajectories, so it does not require any demonstrations (like Chen et al. (2021a) with random trajectories, but unlike Lee et al. (2022), Xu et al. (2022), and the other approaches reviewed above). Secondly, ICPI relies primarily on in-context learning rather than in-weights learning to achieve generalization (like Xu et al. (2022), but unlike Chen et al. (2021a) & Lee et al. (2022)). For discussion about in-weights vs. in-context learning see Chan et al. (2022).

2.2 Gradient-based Training & Finetuning on RL Tasks

Most approaches involve training or fine-tuning foundation models on RL tasks. For example, Janner et al. (2021); Chen et al. (2021a); Lee et al. (2022); Xu et al. (2022); Baker et al. (2022); Reed et al. (2022) all use models that are trained from scratch on tasks of interest, and Singh et al. (2022); Ahn et al. (2022); Huang et al. (2022b) combine frozen foundation models with trainable components or adapters. In contrast, Huang et al. (2022a) use frozen foundation models for planning, without training or fine-tuning on RL tasks. Like Huang et al. (2022a), ICPI does not update the parameters of the foundation model, but relies on the frozen model's in-context learning abilities. However, ICPI gradually builds and improves the prompts within the space defined by the given fixed text-format for observations, actions, and rewards (in contrast to Huang et al. (2022a), which uses the frozen model to select good prompts from a given fixed library of goal/plan descriptions).

2.3 In-Context Learning

Several recent papers have specifically studied in-context learning. Min et al. (2022) demonstrates that LLMs can learn in-context, even when the labels in the prompt are randomized, problemetizing the conventional understanding of in-context learning and showing that label distribution is more important than label correctness. Chan et al. (2022) and Garg et al. (2022b) provide analyses of the properties that drive in-context learning, the first in the context of image classification, the second in the context of regression onto a continuous function. These papers identify various properties, including "burstiness," model-size, and model-architecture, that in-context learning depends on. Chen et al. (2022) studies the sensitivity of in-context learning to small perturbations of the context. They propose a novel method that uses sensitivity as a proxy for model certainty.

3 Algorithm

How can standard policy iteration make use of in-context learning? Policy iteration is either *model-based*—using a world-model to plan future trajectories in the environment—or *model-free*—inferring value-estimates without explicit planning. Both methods can be realized with in-context learning. We choose model-based learning because planned trajectories make the underlying logic of value-

Algorithm 1 Training Loop

```
function TRAIN(environment)
    initialize $\mathcal{D}$                    ▷ replay buffer containing full history of behavior
    while training do
        $\mathrm{s}_0 \leftarrow$ Reset environment.
        while episode is not done do
            $\mathrm{a}_t \leftarrow \arg\max_a Q(\mathrm{s}_t, a, \mathcal{D})$                    ▷ policy improvement
            $\mathrm{s}_{t+1}, \mathrm{r}_t, \mathrm{b}_t \leftarrow$ Execute $\mathrm{a}_t$ in environment.
            $t \leftarrow t + 1$
        end while
        $\mathcal{D} \leftarrow \mathcal{D} \cup (\mathrm{s}_0, \mathrm{a}_0, \mathrm{r}_0, \mathrm{b}_0, \mathrm{s}_1, \ldots, \mathrm{s}_t, \mathrm{a}_t, \mathrm{r}_t, \mathrm{b}_t, \mathrm{s}_{t+1})$    ▷ add trajectory to buffer
    end while
end function
```

Algorithm 2 Computing Q-values

```
function $Q(\mathbf{s}_t, \mathrm{a}, \mathcal{D})$
    $u \leftarrow t$
    $\mathbf{s}^1 = \mathbf{s}_t$
    $\mathrm{a}^1 = \mathrm{a}$
    repeat                                              ▷ All samples come from the experience buffer $\mathcal{D}$
        $\mathcal{D}_\mathrm{b} \sim$ time-steps with action $\mathrm{a}^u$                     ▷ balancing terminal and non-terminal
        $\mathrm{b}^u \sim \mathrm{LLM}\,(\mathcal{D}_\mathrm{b}, \mathbf{s}^u, \mathrm{a}^u)$
        $\mathcal{D}_\mathrm{r} \sim$ time-steps with action $\mathrm{a}^u$ and termination $\mathrm{b}^u$          ▷ balancing reward
        $\mathrm{r}^u \sim \mathrm{LLM}\,(\mathcal{D}_\mathrm{r}, \mathbf{s}^u, \mathrm{a}^u)$
        $\mathcal{D}_\mathbf{s} \sim$ time-steps with action $\mathrm{a}^u$ and termination $\mathrm{b}^u$          ▷ no balancing
        $\mathbf{s}^{u+1} \sim \mathrm{LLM}\,(\mathcal{D}_\mathbf{s}, \mathbf{s}^u, \mathrm{a}^u)$
        $\mathcal{D}_\mathrm{a} \sim c$ recent trajectories
        $\mathrm{a}^{u+1} \sim \mathrm{LLM}\left(\mathbf{s}^{u+1}, \mathcal{D}_\mathrm{a}\right)$
        $u \leftarrow u + 1$
    until $\mathrm{b}^u$ is terminal
    return $\sum_{k=1}^{u} \gamma^{k-1} \mathrm{r}^k$
end function
```

estimates explicit to our foundation model backbone, providing a concrete instantiation of a trajectory that realizes the values. This ties into recent work (Wei et al., 2022b; Nye et al., 2021) demonstrating that "chains of thought" can significantly improve few-shot performance of foundation models.

Model-based RL requires two ingredients, a rollout-policy used to act during planning and a world-model used to predict future rewards, terminations, and states. Since our approach avoids any mutation of the foundation model's parameters (this would require gradients), we must instead induce the rollout-policy and the world-model using in-context learning, i.e. by selecting appropriate prompts. We induce the rollout-policy by prompting the foundation model with trajectories drawn from the current (or recent) behavior policy (distinct from the rollout-policy). Similarly, we induce the world-model by prompting the foundation models with transitions drawn from the agent's history of experience. Note that our approach assumes access to some translation between the state-space of the environment and the medium (language, images, etc.) of the foundation models. This explains how an algorithm might plan and estimate values using a foundation model. It also explains how the rollout-policy approximately tracks the behavior policy.

How does the policy improve? When acting in the environment (as opposed to planning), we choose the action that maximizes the estimated Q-value from the current state (see Training Loop pseudocode, line 6). At time step t, the agent observes the state of the environment (denoted s_t) and executes action $a_t = \arg\max_{\mathrm{a} \in \mathcal{A}} Q^{\pi_t}(\mathbf{s}_t, \mathrm{a})$, where $\mathcal{A} = \{\mathcal{A}(1), \cdots, \mathcal{A}(n)\}$ denotes the set of n actions available, π_t denotes the policy of the agent at time step t, and Q^π denotes the Q-estimate for policy π. Taking the greedy ($\arg\max$) actions with respect to Q^{π_t} implements a new and improved policy.

Computing Q-values This section provides details on the prompts that we use in our computation of Q-values (see Computing Q-values pseudocode & Figure 1). During training, We maintain a buffer $\mathcal{D}$ of transitions experienced by the agent. To compute $Q^{\pi_t}(\mathbf{s}_t, \mathrm{a})$ at time step t in the real-world we rollout a simulated trajectory $\mathbf{s}^1 = \mathbf{s}_t, \mathrm{a}^1 = \mathrm{a}, \mathrm{r}^1, \mathbf{s}^2, \mathrm{a}^2, \mathrm{r}^2, \cdots, \mathbf{s}^T, \mathrm{a}^T, \mathrm{r}^T, \mathbf{s}^{T+1}$ by predicting, at each simulation time step u: reward $\mathrm{r}^u \sim \mathrm{LLM}\,(\mathcal{D}_\mathrm{r}, \mathbf{s}^u, \mathrm{a}^u)$; termination $\mathrm{b}^u \sim \mathrm{LLM}\,(\mathcal{D}_\mathrm{b}, \mathbf{s}^u, \mathrm{a}^u)$; observation $\mathbf{s}^{u+1} \sim \mathrm{LLM}\,(\mathcal{D}_\mathbf{s}, \mathbf{s}^u, \mathrm{a}^u)$; action $\mathrm{a}^{u+1} \sim \mathrm{LLM}\,(\mathcal{D}_\mathrm{a}, \mathbf{s}^u)$. Termination b^u decides whether the simulated trajectory ends at step u.

The prompts $\mathcal{D}_\mathrm{r}$, $\mathcal{D}_\mathrm{b}$ contain data sampled from the replay buffer. For each prompt, we choose some subset of replay buffer transitions, shuffle them, convert them to text (examples are provided in table 4.1) and clip the prompt at the 4000-token Codex context limit. We use the same method for $\mathcal{D}_\mathrm{a}$, except that we use random trajectory subsequences.

In order to maximize the relevance of the prompt contents to the current inference we select transitions using the following criteria. $\mathcal{D}_\mathrm{b}$ contains $(\mathbf{s}_k, \mathrm{a}_k, \mathrm{b}_k)$ tuples such that a_k equals a^u, the action for which the LLM must infer termination. $\mathcal{D}_\mathrm{r}$ contains $(\mathbf{s}_k, \mathrm{a}_k, \mathrm{r}_k)$ tuples, again constraining $\mathrm{a}_k = \mathrm{a}^u$ but also constraining $\mathrm{b}_k = \mathrm{b}^k$ — that the tuple corresponds to a terminal time-step if the LLM inferred $\mathrm{b}^u =$ true, and to a non-terminal time-step if $\mathrm{b}^u =$ false. For $\mathcal{D}_\mathbf{s}$, the prompt includes $(\mathbf{s}_k, \mathrm{a}_k\mathbf{s}_{k+1})$ tuples with $\mathrm{a}_k = \mathrm{a}^u$ and $\mathrm{b}_k =$ false (only non-terminal states need to be modelled).

We also maintain a balance of certain kinds of transitions in the prompt. For termination prediction, we balance terminal and non-terminal time-steps. Since non-terminal time-steps far outnumber terminal time-steps, this eliminates a situation wherein the randomly sampled prompt time-steps are entirely non-terminal, all but ensuring that the LLM will predict non-termination. Similarly, for reward prediction, we balance the number of time-steps corresponding to each reward value stored in $\mathcal{D}$. In order to balance two collections of unequal size, we take the smaller and duplicate randomly chosen members until the sizes are equal.

In contrast to the other predictions, we condition the rollout policy on trajectory subsequences, not individual time-steps. Prompting with sequences better enables the foundation model to apprehend the logic behind a policy. Trajectory subsequences consist of $(\mathbf{s}_k, \mathrm{a}_k)$ pairs, randomly clipped from the c most recent trajectories. More recent trajectories will, in general demonstrate higher performance, since they come from policies that have benefited from more rounds of improvement.

Finally, the Q-value estimate is simply the discounted sum of rewards for the simulated episode. Given this description of Q-value estimation, we now return to the concept of policy improvement.

Policy-Improvement The $\arg\max$ (line 6 of Algorithm 1) drives policy improvement in ICPI. Critically this is not simply a one-step improvement as with Trajectory Transformer (Janner et al., 2021) but a mechanism that builds improvement on top of improvement. This occurs through a cycle in which the $\arg\max$ improves behavior. The improved behavior is stored in the buffer $\mathcal{D}$, and then used to condition the rollout policy. This improves the returns generated by the LLM during planning rollouts. These improved rollouts improve the Q-estimates for each action. Completing the cycle, this improves the actions chosen by the $\arg\max$. Because this process feeds into itself, it can drive improvement without bound until optimality is achieved.

Note that this process takes advantage of properties specific to in-context learning. In particular, it relies on the assumption that the rollout policy, when prompted with trajectories drawn from a mixture of policies, will approximate something like an average of these policies. Given this assumption, the rollout policy will improve with the improvement of the mixture of policies from which its prompt-trajectories are drawn. This results in a kind of rapid policy improvement that works without any use of gradients.

Prompt-Format The LLM cannot take non-linguistic prompts, so our algorithm assumes access to a textual representation of the environment—of states, actions, terminations, and rewards—and some way to recover the original action, termination, and reward values from their textual representation (we do not attempt to recover states). Since our primary results use the Codex language model (see Table 2), we use Python code to represent these values (examples are available in Table 1).

In our experiments, we discovered that the LLM world-model was unable to reliably predict rewards, terminations, and next-states on some of the more difficult environments. We experimented with providing domain *hints* in the form of prompt formats that make explicit useful information — similar to Chain of Thought Prompting (Wei et al., 2022b). For example, for the *chain* domain, the hint includes an explicit comparison (== or `!=`) of the current state with the goal state. Note that while hints are provided in the initial context, the LLM must infer the hint content in rollouts generated from this context.

We use a consistent idiom for rewards and terminations, namely `assert reward == x` and `assert done` or `assert not done`. Some decisions had to be made when representing states and actions. In general, we strove to use simple, idiomatic, concise Python. On the more challenging environments, we did search over several options for the choice of hint. We anticipate that in the future, stronger foundation models will be increasingly robust to these decisions.

4 EXPERIMENTS

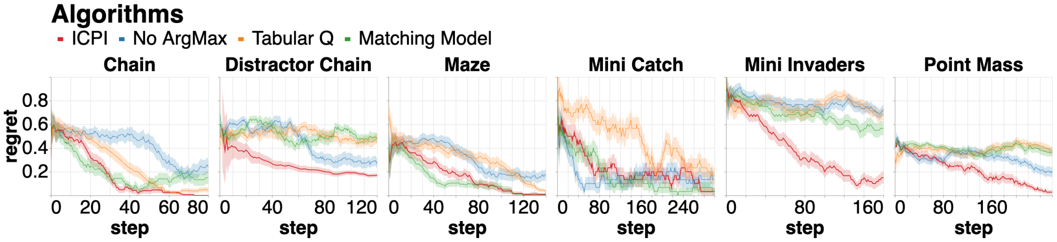

Figure 2: Comparison of ICPI with three baselines, "No $\arg\max$", "Tabular Q," and "Nearest Neighbor." The y-axis depicts regret (normalized between 0 and 1), computed relative to an optimal return with a discount-factor of 0.8. The x-axis depicts time-steps during training. Error bars are standard errors from four seeds.

Chain	`assert state == 6 and state != 4`
	`state = left()`
	`assert reward == 0`
	`assert not done`

Distractor	`assert state == (6, 3)`
	`and state != (4, 3)`
	`state = left()`
	`assert reward == 0`
	`assert not done`

Maze	`assert state == C(i=2, j=1)`
	`and state != C(i=1, j=0)`
	`state, reward = left()`
	`assert reward == 0`
	`assert not don`

Mini Catch	`assert paddle == C(2, 0)`
	`and ball == C(0, 4)`
	`and paddle.x == 2 and ball.x == 0`
	`and paddle.x > ball.x`
	`and ball.y == 4`
	`reward = paddle.left()`
	`ball.descend()`
	`assert reward == 0`
	`assert not done`

Mini Invaders	`assert ship == C(2, 0)`
	`and aliens == [C(3, 5), C(1, 5)]`
	`and (ship.x,`
	`aliens[0].x,`
	`aliens[1].x) == (2, 3, 1)`
	`and ship.x < aliens[0].x`
	`and ship.x > aliens[1].x`
	`ship.left()`
	`assert reward == 0`
	`for a in aliens:`
	`a.descend()`
	`assert not done`

Point-Mass	`assert pos == -3.45 and vel == 0.00`
	`and pos < -2 and vel == 0`
	`pos, vel = decel(pos, vel)`
	`assert reward == 0`
	`assert not done`

Table 1: Prompt formats. Basic prompt format in black, additional hint formats in grey (see §3 and §4.1 for further discussion).

We have three main goals in our experiments: (1) Demonstrate that the agent algorithm can in fact quickly learn good policies, using pretrained LLMs, in a set of six simple illustrative domains of increasing challenge; (2) provide evidence through an ablation that the policy-improvement step—taking the $\arg\max$ over Q-values computed through LLM rollouts—accelerates learning; and (3) investigate the impact of using different LLMs (see Table 2)—different sizes and trained on different data, in particular, trained on (mostly) natural language (GPT-3 and GPT-J) vs. program code (Codex and InCoder). We next describe the six domains and their associated prompt formats, and then describe the experimental methodology and results.

4.1 Domains and prompt format

Chain. In this environment, the agent occupies an 8-state chain. The agent has three actions: *Left*, *right*, and *try goal*. The *try goal* action always terminates the episode, conferring a reward of 1 on state 4 (the goal state) and 0 on all other states. Episodes also terminate after 8 time-steps. States are represented as numbers from 0 to 7, as in `assert state == n`, with the appropriate integer substituted for `n`. The actions are represented as functions `left()`, `right()`, and `try_goal()`. For the hint, we simply indicate whether or not the current state matches the goal state, 4.

Distractor Chain. This environment is an 8-state chain, identical to the *chain* environment, except that the observation is a *pair* of integers, the first indicating the true state of the agent and the second acting as a distractor which transitions randomly within $\{0, \ldots, 7\}$. The agent must therefore learn to ignore the distractor integer and base its inferrences on the information contained in the first integer. Aside from the addition of this distractor integer to the observation, all text representations and hints are identical to the *chain* environment.

Maze. The agent navigates a small 3×3 gridworld with obstacles. The agent can move *up*, *down*, *left*, or *right*. The episode terminates with a reward of 1 once the agent navigates to the goal grid, or with a reward of 0 after 8 time-steps. This environment tests our algorithms capacity to handle 2-dimensional movement and obstacles, as well as a 4-action state-space. We represent the states as namedtuples — `C(x, y)`, with integers substituted for `x` and `y`. Similar to *chain*, the hint indicates whether or not the state corresponds to the goal state.

Mini Catch. The agent operates a paddle to catch a falling ball. The ball falls from a height of 5 units, descending one unit per time step. The paddle can *stay* in place (not move), or move *left* or *right* along the bottom of the 4-unit wide plane. The agent receives a reward of 1 for catching the ball and 0 for other time-steps. The episode ends when the ball's height reaches the paddle regardless of whether or not the paddle catches the ball. We chose this environment specifically to challenge the action-inference/rollout-policy component of our algorithm. Specifically, note that the success condition in Mini Catch allows the paddle to meander before moving under the ball—as long as it gets there on the final time-step. Successful trajectories that include movement *away* from the ball thus make a good rollout policies more challenging to learn (i.e., elicit from the LLM via prompts).

Again, we represent both the paddle and the ball as namedtuples `C(x, y)` and we represent actions as methods of the `paddle` object: `paddle.stay()`, `paddle.left()`, and `paddle.right()`. For the hint, we call out the location of the paddle's x-position, the ball's x-position, the relation between these positions (which is larger than which, or whether they are equal) and the ball's y-position. Table 1 provides an example. We also include the text `ball.descend()` to account for the change in the ball's position between states.

Mini Invaders. The agent operates a ship that shoots down aliens which descend from the top of the screen. At the beginning of an episode, two aliens spawn at a random location in two of four columns. The episode terminates when an alien reaches the ground (resulting in 0 reward) or when the ship shoots down both aliens (the agent receives 1 reward per alien). The agent can move *left*, *right*, or *shoot*. This domain highlights ICPI's capacity to learn incrementally, rather than discovering an optimal policy through random exploration and then imitating that policy, which is how our "No $\arg\max$" baseline learns (see Comparison of ICPI with baseline algorithms). ICPI initially learns to shoot down one alien, and then builds on this good but suboptimal policy to discover the better policy of shooting down both aliens. In contrast, random exploration takes much longer to discover the optimal policy and the "No $\arg\max$" baseline has only experienced one or two successful trajectories by the end of training.

We represent the ship by its namedtuple coordinate (`C(x, y)`) and the aliens as a list of these namedtuples. When an alien is shot down, we substitute `None` for the tuple, as in `aliens == [C(x, y), None]`. We add the text `for a in aliens: a.descend()` in order to account for the change in the alien's position between states.

Point-Mass. A point-mass spawns at a random position on a continuous line between -6 and $+6$ with a velocity of 0. The agent can either *accelerate* the point-mass (increase velocity by 1) or *decelerate* it (decrease the velocity by 1). The point-mass position changes by the amount of its velocity each timestep. The episode terminates with a reward of 1 once the point-mass is between -2 and $+2$ and its velocity is 0 once again. The episode also terminates after 8 time-steps. This domain tests the algorithm's ability to handle continuous states.

States are represented as `assert pos == p and vel == v`, substituting floats rounded to two decimals for `p` and `v`. The actions are `accel(pos, vel)` and `decel(pos, vel)`. The hint indicates whether the success conditions are met, namely the relationship of `pos` to -2 and $+2$ and the whether or not `vel == 0`. The hint includes identification of the aliens' and the ship's x-positions as well as a comparison between them.

4.2 Experiment Methodology and Results

Methodology and evaluation. For the results, we record the agent's regret over the course of training relative to an optimal policy computed with a discount factor of 0.8. For all experiments $c = 8$ (the number of most recent successful trajectories to include in the prompt). We did not have time for hyperparameter search and chose this number based on intuition, however the $c = 16$ baseline demonstrates results when this hyperparameter is doubled. All results use 4 seeds.

For both versions of Codex, we used the OpenAI Beta under the API Terms of Use. For GPT-J (Wang & Komatsuzaki, 2021), InCoder (Fried et al., 2022) and OPT-30B (Zhang et al., 2022), we used the open-source implementations from Huggingface Transformers (Wolf et al., 2020), each running on one Nvidia A40 GPU. All language models use a sampling temperature of 0.1.

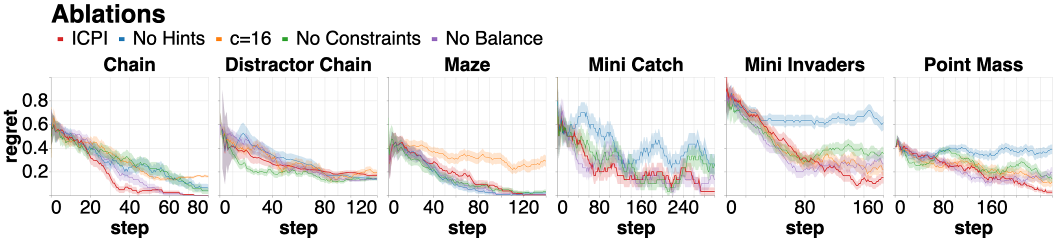

Figure 3: Comparison of ICPI with ablations. The y-axis depicts regret (normalized between 0 and 1), computed relative to an optimal return with a discount-factor of 0.8. The x-axis depicts time-steps during training. Error bars are standard errors from four seeds.

Comparison of ICPI with baseline algorithms. We compare ICPI with three baselines (Fig. 2).

The "No $\arg\max$" baseline learns a good policy through random exploration and then imitates this policy. This baseline assumes access to a "success threshold" for each domain — an undiscounted cumulative return greater than which a trajectory is considered successful. The action selection mechanism emulates ICPI's rollout policy: prompting the LLM with a set of trajectories and eliciting an action as output. For this baseline, we only include trajectories in the prompt whose cumulative return exceeds the success threshold. Thus the policy improves as the number of successful trajectories in the prompt increases over time. Note that at the start of learning, the agent will have experienced too few successful trajectories to effectively populate the policy prompt. In order to facilitate exploration, we act randomly until the agent experiences 3 successes.

"Tabular Q" is a standard tabular Q-learning algorithm, which uses a learning rate of 1.0 and optimistically initializes the Q-values to 1.0.

"Matching Model" is a baseline which uses the trajectory history instead of an LLM to perform modelling. This baseline searches the trajectory buffer for the most recent instance of the current state, and in the case of transition/reward/termination prediction, the current action. If a match is found, the model's outputs the historical value (e.g. the reward associated with the state-action pair found in the buffer). If no match is found, the modelling rollout is terminated. Recall that ICPI breaks ties randomly during action selection so this will often lead to random action selection.

As our results demonstrate, only ICPI learns good policies on all domains. We attribute this advantage to ICPI's ability to generalize from its context to unseen states and state/action pairs (unlike "Tabular Q" and "Matching Model"). Unlike "No $\arg\max$" ICPI is able to learn progressively, improving the policy before experiencing good trajectories.

Ablation of ICPI components. With these experiments, we ablate those components of the algorithm which are not, in principle, essential to learning (Fig. 3). "No Hints" ablates the hints described in the Prompt-Format paragraph. "No Balance" removes the balancing of different kinds of time-steps described in the Computing Q-values paragraph (for example, $\mathcal{D}_\text{b}$ is allowed to contain an unequal number of terminal and non-terminal time-steps). The "No Constraints" baseline removes the constraints on these time-steps described in the same paragraph. For example, $\mathcal{D}_\text{r}$ is allowed to contain a mixture of terminal and non-terminal time-steps (regardless of the model's termination prediction). Finally, "$c = 16$" prompts the rollout policy with the last 16 trajectories (instead of the last 8, as in ICPI). We find that while some ablations match ICPI's performance in several domains, none match its performance on all six.

Comparison of Different Language Models. While our lab lacks the resources to do a full study of scaling properties, we did compare several language models of varying size (Fig. 4). See Table 2 for details about these models. Both `code-davinci-002` and `code-cushman-001` are variations of the Codex language model. The exact number of parameters in these models is proprietary according to OpenAI, but Chen et al. (2021b) describes Codex as fine-tuned from GPT-3 Brown et al. (2020), which contains 185 billion parameters. As for the distinction between the variations, the OpenAI website describes `code-cushman-001` as "almost as capable as Davinci Codex, but slightly faster."

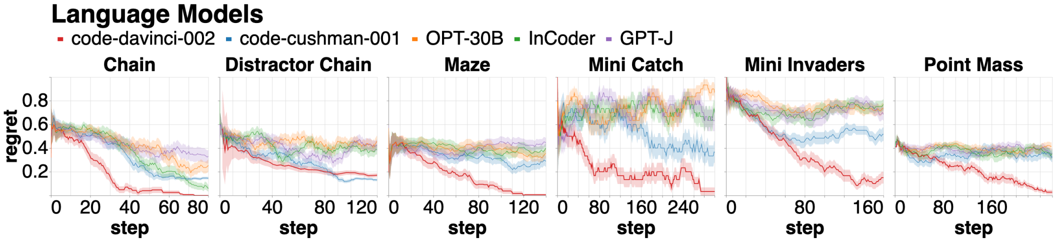

Figure 4: Comparison of ICPI with ablations. The y-axis depicts regret (normalized between 0 and 1), computed relative to an optimal return with a discount-factor of 0.8. The x-axis depicts time-steps of training. Error bars are standard errors from four seeds.

Table 2: Pretrained Large Language Models (LLMs) Used in Experiments

Model	**Parameters**	**Training data**
GPT-J (Wang & Komatsuzaki, 2021)	6 billion	"The Pile" (Gao et al., 2020), an 825GB English corpus incl. Wikipedia, GitHub, academic pubs
InCoder (Fried et al., 2022)	6.7 billion	159 GB of open-source StackOverflow code
OPT-30B (Zhang et al., 2022)	30 billion	180B tokens of predominantly English data including "The Pile" (Gao et al., 2020) and "PushShift.io Reddit" (Baumgartner et al., 2020)
Codex (Chen et al., 2021b)	185 billion	179 GB of GitHub code

We found that none of the smaller models were capable of demonstrating learning on any domain but the simplest, *chain*. Examining the trajectories generated by agents trained using these models, we noted that in several cases, they seemed to struggle to apprehend the underlying "logic" of successful trajectories, which hampered the ability of the rollout policy to produce good actions. Since these smaller models were not trained on identical data, we are unable to isolate the role of size in these results. However, the failure of all of these smaller models to learn suggests that size has some role to play in performance. We conjecture that larger models developed in the future may demonstrate comparable improvements in performance over our Codex model.

Limitations In principle, ICPI can work on any control task with a discrete state space. All that the method requires is a sequence prediction model (in the paper we use Codex) capable of inferring state transitions and action probabilities given a history of behavior. Such a model will ensure that the rollouts (described in Computing Q-values) are unbiased monte-carlo estimates of value for the current policy. Given such estimates, the algorithm will implement policy iteration, a method which is proven to converge. The domains in our paper were limited by the abilities of Codex to accurately predict transitions and actions in more complex domains. As sequence models mature, we anticipate that more domains will become tractable for ICPI.

5 Conclusion

Our main contribution is a gradient-free policy iteration algorithm, using foundation models to solve RL tasks, in which the entire locus of learning is improved prompt content. The algorithm uses a foundation models as both a world model and policy to compute Q-values via rollouts. Although we presented the algorithm here as text-based, the approach is quite general and could be applied to any foundation models that works through prompting, including multi-modal models like Reed et al. (2022) and Seo et al. (2022). In experiments we showed that the algorithm works in six illustrative domains imposing different challenges for ICPI, confirming the benefit of the LLM-rollout-based policy improvement. While the empirical results are quite modest, we believe the approach provides an important new way to use LLMs that will increase in effectiveness as the models themselves become better, as our size comparison study suggests.

REFERENCES

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, 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, and Mengyuan Yan. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances, April 2022. URL http://arxiv.org/abs/2204.01691. arXiv:2204.01691 [cs].

Prithviraj Ammanabrolu and Mark Riedl. Learning Knowledge Graph-based World Models of Textual Environments. In *Advances in Neural Information Processing Systems*, volume 34, pp. 3720–3731. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/hash/1e747ddbea997a1b933aaf58a7953c3c-Abstract.html.

Bowen Baker, Ilge Akkaya, Peter Zhokhov, Joost Huizinga, Jie Tang, Adrien Ecoffet, Brandon Houghton, Raul Sampedro, and Jeff Clune. Video PreTraining (VPT): Learning to Act by Watching Unlabeled Online Videos, June 2022. URL http://arxiv.org/abs/2206.11795. arXiv:2206.11795 [cs].

Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. The Pushshift Reddit Dataset, January 2020. URL http://arxiv.org/abs/2001.08435. Number: arXiv:2001.08435 arXiv:2001.08435 [cs].

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020. URL https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Stephanie C. Y. Chan, Adam Santoro, Andrew K. Lampinen, Jane X. Wang, Aaditya Singh, Pierre H. Richemond, Jay McClelland, and Felix Hill. Data Distributional Properties Drive Emergent In-Context Learning in Transformers, May 2022. URL http://arxiv.org/abs/2205.05055. arXiv:2205.05055 [cs].

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision Transformer: Reinforcement Learning via Sequence Modeling. *arXiv:2106.01345 [cs]*, June 2021a. URL http://arxiv.org/abs/2106.01345.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating Large Language Models Trained on Code. Technical Report arXiv:2107.03374, arXiv, July 2021b. URL http://arxiv.org/abs/2107.03374.

Yanda Chen, Chen Zhao, Zhou Yu, Kathleen McKeown, and He He. On the Relation between Sensitivity and Accuracy in In-context Learning, September 2022. URL http://arxiv.org/abs/2209.07661. arXiv:2209.07661 [cs].

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A Generative Model for Code Infilling and Synthesis. *arXiv:2204.05999 [cs]*, April 2022. URL `http://arxiv.org/abs/2204.05999`.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv:2101.00027 [cs]*, December 2020. URL `http://arxiv.org/abs/2101.00027`.

Divyansh Garg, Skanda Vaidyanath, Kuno Kim, Jiaming Song, and Stefano Ermon. LISA: Learning Interpretable Skill Abstractions from Language. *arXiv:2203.00054 [cs]*, February 2022a. URL `http://arxiv.org/abs/2203.00054`.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. What Can Transformers Learn In-Context? A Case Study of Simple Function Classes, August 2022b. URL `http://arxiv.org/abs/2208.01066`. arXiv:2208.01066 [cs].

Felix Hill, Sona Mokra, Nathaniel Wong, and Tim Harley. Human Instruction-Following with Deep Reinforcement Learning via Transfer-Learning from Text. *arXiv:2005.09382 [cs]*, May 2020. URL `http://arxiv.org/abs/2005.09382`.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. *arXiv:2201.07207 [cs]*, March 2022a. URL `http://arxiv.org/abs/2201.07207`.

Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner Monologue: Embodied Reasoning through Planning with Language Models, July 2022b. URL `http://arxiv.org/abs/2207.05608`. arXiv:2207.05608 [cs].

Michael Janner, Qiyang Li, and Sergey Levine. Offline Reinforcement Learning as One Big Sequence Modeling Problem, November 2021. URL `http://arxiv.org/abs/2106.02039`. arXiv:2106.02039 [cs].

Rabeeh Karimi Mahabadi, Luke Zettlemoyer, James Henderson, Lambert Mathias, Marzieh Saeidi, Veselin Stoyanov, and Majid Yazdani. Prompt-free and Efficient Few-shot Learning with Language Models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3638–3652, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.254. URL `https://aclanthology.org/2022.acl-long.254`.

Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, and Igor Mordatch. Multi-Game Decision Transformers, May 2022. URL `http://arxiv.org/abs/2205.15241`. arXiv:2205.15241 [cs].

Shuang Li, Xavier Puig, Chris Paxton, Yilun Du, Clinton Wang, Linxi Fan, Tao Chen, De-An Huang, Ekin Akyürek, Anima Anandkumar, Jacob Andreas, Igor Mordatch, Antonio Torralba, and Yuke Zhu. Pre-Trained Language Models for Interactive Decision-Making. *arXiv:2202.01771 [cs]*, February 2022. URL `http://arxiv.org/abs/2202.01771`.

Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. January 2021. doi: 10.48550/arXiv.2101.00190. URL `https://arxiv.org/abs/2101.00190v1`.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. Pretrained Transformers as Universal Computation Engines. *arXiv:2103.05247 [cs]*, March 2021. URL `http://arxiv.org/abs/2103.05247`.

Arjun Majumdar, Ayush Shrivastava, Stefan Lee, Peter Anderson, Devi Parikh, and Dhruv Batra. Improving Vision-and-Language Navigation with Image-Text Pairs from the Web. *arXiv:2004.14973 [cs]*, May 2020. URL `http://arxiv.org/abs/2004.14973`.

Sewon Min, Xinxi Lyu, Ari Holtzman, Mikel Artetxe, Mike Lewis, Hannaneh Hajishirzi, and Luke Zettlemoyer. Rethinking the Role of Demonstrations: What Makes In-Context Learning Work? *arXiv:2202.12837 [cs]*, February 2022. URL http://arxiv.org/abs/2202.12837.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show Your Work: Scratchpads for Intermediate Computation with Language Models, November 2021. URL http://arxiv.org/abs/2112.00114. arXiv:2112.00114 [cs].

Xiangyu Peng, Mark O. Riedl, and Prithviraj Ammanabrolu. Inherently Explainable Reinforcement Learning in Natural Language. *arXiv:2112.08907 [cs]*, December 2021. URL http://arxiv.org/abs/2112.08907.

Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas. A Generalist Agent, May 2022. URL http://arxiv.org/abs/2205.06175. arXiv:2205.06175 [cs].

Younggyo Seo, Kimin Lee, Fangchen Liu, Stephen James, and Pieter Abbeel. HARP: Autoregressive Latent Video Prediction with High-Fidelity Image Generator, September 2022. URL http://arxiv.org/abs/2209.07143. arXiv:2209.07143 [cs].

Aaditya K. Singh, David Ding, Andrew Saxe, Felix Hill, and Andrew K. Lampinen. Know your audience: specializing grounded language models with the game of Dixit, June 2022. URL http://arxiv.org/abs/2206.08349. arXiv:2206.08349 [cs].

Ishika Singh, Gargi Singh, and Ashutosh Modi. Pre-trained Language Models as Prior Knowledge for Playing Text-based Games. *arXiv:2107.08408 [cs]*, December 2021. URL http://arxiv.org/abs/2107.08408.

Allison C. Tam, Neil C. Rabinowitz, Andrew K. Lampinen, Nicholas A. Roy, Stephanie C. Y. Chan, D. J. Strouse, Jane X. Wang, Andrea Banino, and Felix Hill. Semantic Exploration from Language Abstractions and Pretrained Representations. *arXiv:2204.05080 [cs]*, April 2022. URL http://arxiv.org/abs/2204.05080.

Denis Tarasov, Vladislav Kurenkov, and Sergey Kolesnikov. Prompts and Pre-Trained Language Models for Offline Reinforcement Learning. March 2022. URL https://openreview.net/forum?id=Spf4TE6NkWq.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model, May 2021. URL https://github.com/kingoflolz/mesh-transformer-jax.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. Emergent Abilities of Large Language Models, June 2022a. URL http://arxiv.org/abs/2206.07682. arXiv:2206.07682 [cs].

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of Thought Prompting Elicits Reasoning in Large Language Models. *arXiv:2201.11903 [cs]*, April 2022b. URL http://arxiv.org/abs/2201.11903.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua B. Tenenbaum, and Chuang Gan. Prompting Decision Transformer for Few-Shot Policy Generalization, June 2022. URL http://arxiv.org/abs/2206.13499. arXiv:2206.13499 [cs].

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. OPT: Open Pre-trained Transformer Language Models, 2022. _eprint: 2205.01068.