In-Context Policy Iteration for Dynamic Manipulation

Anonymous Author(s)
Affiliation
Address
email

Abstract

Attention-based architectures trained on internet-scale language data have demonstrated state of the art reasoning ability for various language-based tasks, such as logic problems and textual reasoning. Additionally, these Large Language Models (LLMs) have exhibited the ability to perform few-shot prediction via in-context learning, in which input-output examples provided in the prompt are generalized to new inputs. This ability furthermore extends beyond standard language tasks, enabling few-shot learning for general patterns. In this work, we consider the application of in-context learning with pre-trained language models for dynamic manipulation. Dynamic manipulation introduces several crucial challenges, including increased dimensionality, complex dynamics, and partial observability. To address this, we take an iterative approach, and formulate our in-context learning problem to predict adjustments to a parametric policy based on previous interactions. We show across several tasks in simulation and on a physical robot that utilizing in-context learning outperforms alternative methods in the low data regime.

1 Introduction

Transformer-based architectures trained on internet-scale data, Large Language Models (LLMs), have recently become a powerful tool for a variety of language and image-based tasks [29]. Recent works have proposed robotics methods that exploits the reasoning ability of these LLMs over textual and visual inputs to perform high-level planning [9], generate robot control code [12], and derive reward functions [8]. These methods all exploit *in-weights learning*, exploiting the information stored in the weights of the networks. But what if a task is not well reflected in textual inputs or even in visual information?

Amongst the emergent abilities, these Large Language Models (LLMs) can perform few-shot learning, in which a small number of input-output examples are provided in the prompt to the model, before being provided with a test input [2]. This form of learning has been dubbed *in-context learning*. Curiously, this ability has been shown to extend to patterns that are not necessarily reflected in the training domain, suggesting that LLMs act as *general pattern machines* [17]. This invites an intriguing proposition: that large-scale language pre-training can enable few-shot learning in new domains. In this work, we explore how in-context learning can be applied for a class of robotic tasks which are not easy to reflect in textual and visual inputs alone: dynamic manipulation.

Robotic manipulation that exploits *dynamics* can introduce several benefits over quasi-static systems. Dynamic manipulation can increase the robot workspace [33, 32], improve task efficiency [7, 27], and extend robot capabilities with new skills such as dynamic in-hand regrasping [24]. Designing systems for dynamic manipulation introduces several challenges. First, the system dynamics are often dependent upon underlying physical properties that are not easy to

Submitted to 39th Conference on Neural Information Processing Systems (NeurIPS 2025). Do not distribute.

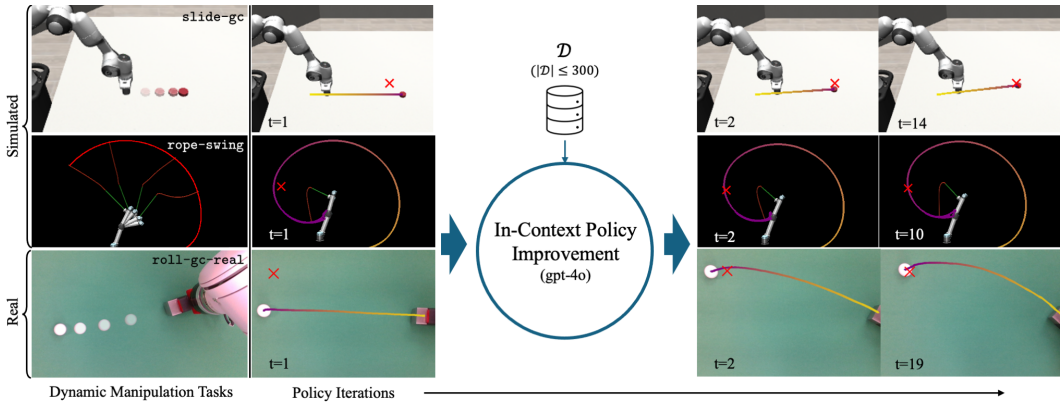


Figure 1: We investigate in-context learning for iteratively improving policy parameters for dynamic manipulation tasks. We demonstrate across a variety of tasks, both in simulation and on a real robot, that utilizing *in-context learning* with a pretrained Large Language Model (such as OpenAI's gpt-4o), can learn to improve dynamic manipulation policies interactively from a small (≤ 300) policy improvement dataset.

observe directly, including from vision, such as the mass of an object, friction of a surface, or density of a rope. These properties require dynamic interactions [28, 26] and/or additional sensing [24] to accurately observe. Second, the increased dimensionality of the system (such as deformable manipulation [7, 5]) and the complexity of the dynamics (such as frictional contact interactions) have largely limited current systems to single-task solutions. These solutions generally involve collecting large datasets and training task-specific solutions [5, 28, 13].

Humans performing dynamic manipulation may fail on a first attempt, but utilize their experience interacting with similar dynamical systems to iteratively improve their approach on a target system. In this work, we seek to enable robots that also can iteratively improve their interactions with a dynamical system. We investigate enabling a few-shot iterative policy improvement technique, utilizing in-context-learning with pre-trained LLMs. We formulate our problem as learning an improvement operator, which predicts adjustments to a parametric policy based on the results of previous interactions. Previous interactions are tokenized and fed as inputs along with policy parameters and in-context learning is utilized to output a change to the policy parameters that drives towards task success. We utilize a small dataset of improvement labels from interactions with similar tasks to form our in-context examples.

We apply our method to a variety of dynamic manipulation tasks, in simulation and on a physical robot (Fig. 1). Our results indicate the novel ability of pre-trained LLMs to perform in-context policy improvement for dynamic manipulation tasks without requirement of any fine-tuning or training. We show that our proposed approach outperforms utilizing in-weights reasoning or alternative in-context policy approaches [17] and compares favorably to alternative policy optimization approaches (e.g., Bayesian Optimization) and other policy improvement operators in the low data regime.

2 Related Work

2.1 Large Language Models and In-Context Learning

Large Language Models (LLMs), attention-based architectures [23] trained on internet-scale language data, have demonstrated remarkable ability in generating solutions to various language-based tasks such as logic problems and math puzzles [20, 22] and reasoning about joint visual and textual inputs [29]. As model scale increased, LLMs were shown to be capable of few-shot learning, where input-output examples for an unseen problem are provided in the prompt, without the need for model updates [2]. LLMs thus exhibit two forms of learning: *in-weights learning* involves information stored in the model weights via gradient updates on the training dataset, while *in-context learning* involves information provided only in the inputs at inference time (i.e., the prompt), with no weight updates. Recent work has shown that *in-context learning* can be applied outside of the training domain, suggesting that pre-trained LLMs can act as general pattern machines, capable of identifying input-output patterns in-context in new target domains [17].

In-context learning (ICL) emerges as a result of explicitly training on large contexts that cover multiple task examples [2] or implicitly due to distributional properties of the training data [3]. Some work suggests ICL is driven by induction heads, an attention mechanism which associates completions with previous, related completions in the prompt [19]. Behaviorally, transformers exhibit examplar-based generalization in-context, compared to rule-based generalization in-weights [4]. The phenomena is still not fully understood: [16] showed that for certain text-based tasks, randomizing the output labels in the context has little effect on performance, suggesting that exposure to input/output distributions and/or improved in-weights recall drives improvement, rather than learning from in-context labeled data. ICL has, however, been demonstrated on out of distribution tasks, suggesting labels are utilized in certain cases [3, 17].

2.2 LLM for Robotics

The majority of LLM-enabled robotics methods to date largely exploit in-weights reasoning. These systems utilize the reasoning capabilities of large pre-trained models to perform zero-shot (or few-shot) reasoning on a variety of robotics tasks by prompting LLMs to generate high-level plans [9, 21], robot control code [12], value functions for trajectory optimization [8, 31], reward functions for Reinforcement Learning (RL) [25], and motion trajectories [10]. These methods demonstrate flexible systems that can be applied to broad classes of tasks, purely exploiting the knowledge and reasoning in the pretrained models. The tasks considered, however, are largely quasi-static and kinematic in nature, such as pick-and-place, opening/closing, and object retrieval. The emphasis on largely high-level reasoning tasks may result in more subject matter overlap in an LLM training corpus. Reasoning about low-level dynamics, however, is unlikely to appear in the training data as the relevant data is rarely presented explicitly, let alone in text. This motivates our investigation of ICL as an LLM mechanism applicable to dynamic manipulation.

2.3 In-Context Learning for Robotics

Most related to our work is the sequence improvement demonstrated in [17]. They propose a purely in-context method that improves a trajectory for tasks such as cartpole or a reaching task, where the trajectory/policy is tokenized and iteratively improved based on execution feedback. In contrast, our method uses a small number of examples to fit a policy improvement operator in context. Also related is Keypoint Action Tokens, which performs behavior cloning via in-context learning [6]. Behavior cloning is difficult to apply to these dynamic manipulation tasks as the task parameters are not always available a priori. Finally, some work trains transformers from scratch for multi-task policy adaptation [30] or system identification [34]. This requires significant data and modeling effort. In contrast, our work relies solely on learning in-context using a pre-trained transformer.

3 Problem Formulation

We consider solving a dynamic manipulation task drawn from a task distribution $\tau \sim P(\tau)$. τ reflects the variation in the task, such as physical parameters of the system (e.g., friction or mass) or the desired goal for a goal-conditioned policy. We assume a parametric policy π_θ. We have access to the environment through $s_{1:T} = T_\tau(\pi_\theta, s_0)$, where s_t is the state of the system at time t and s_0 is the initial state. Finally, we have access to a task cost function $C_\tau(s_{1:T})$ which evaluates the cost of a particular rollout. The objective is to identify the best policy parameters for the given task:

$$\theta^* = \arg\min_\theta C_\tau(T_\tau(\pi_\theta, s_0)) \tag{1}$$

3.1 Tasks

We consider five different tasks using three robot setups, two simulated and one physical robot. We describe in each case the task parameters, state definition, and policy parameterization. In accordance with similar dynamic manipulation work, we utilize parametric motions as our policy, designed in accordance with each task [5]. The task setups are shown in Fig. 1.

3.1.1 Slide Task (`slide`)

In this simulated task, a cylindrical puck is placed on a support surface in front of a Franka Emika Panda robot manipulator with a cylindrical end effector attachment. The task is to strike the puck with the robot end effector, causing it to slide along the surface to some goal configuration. In this task iteration, the start and goal location is fixed, while the physical parameters of the puck are adjusted. In particular, we adjust the puck radius τ_r and the surface friction coefficient τ_μ. The task parameters τ_r, τ_μ are unknown to the policy. Our state space $s_t \in \mathbb{R}^2$ is the location of the puck center at time t on the plane of the table. The goal $\tau_g \in \mathbb{R}^2$ is a fixed location on the table. The task cost function is the distance to the goal configuration,

$$C_\tau^{\texttt{slide}}(s_{1:T}) = ||s_T - \tau_g||_2 \tag{2}$$

For this task, we use a 3-dimensional policy parameterization $\theta = (\theta_\alpha, \theta_d, \theta_t)$ which defines the robot motion. θ_α is the angle of the robot motion from start, with 0 degrees pushing straight forward. θ_d is the distance to move during the push. θ_t is the time to complete the push.

3.1.2 Slide Goal-Conditioned Task (`slide-gc`)

This is a goal-conditioned variation on the simulation slide task. We fix the puck radius and friction and instead adjust the goal configuration $\tau_g \in \mathbb{R}^2$. We use the same task cost function as the `slide` task and change the policy parameterization slightly so that θ_α is the approach angle to the puck to ensure contact is made.

3.1.3 Rope Swing Task (`rope-swing`)

Our next task is the simulated rope swing task from Chi et al [5]. In the task, a rod is attached to the end of a UR5e robot manipulator, and a rope is attached to the end of the rod. The goal of the task is to swing the rope so that the tip of the rope passes through a particular location. In this task iteration, the goal location is fixed. We adjust the rod length τ_r and rope length τ_l, and both values are unknown to the policy. The state space $s_t \in \mathbb{R}^2$ is the location of the rope end in the Y-Z plane, in which the swing occurs. The goal $\tau_g \in \mathbb{R}^2$ is a fixed location in the plane. The task cost function is the smallest distance from the state to the goal during the swing,

$$C_\tau^{\texttt{rope-swing}}(s_{1:T}) = \min_t ||s_t - \tau_g||_2 \tag{3}$$

We use the same policy parameterization from Chi et al [5]. Namely, we use a 3-dimensional policy parameterization $\theta = (\theta_v, \theta_{J_2}, \theta_{J_3})$, which defines the swing. θ_v is the angular velocity while θ_{J_2} and θ_{J_3} defines the end motion of the second and third joints of the robot.

3.1.4 Rope Swing Goal-Conditioned Task (`rope-swing-gc`)

This is a goal-conditioned variation on the simulated rope swing task. We fix the rod and rope lengths and adjust the goal configuration $\tau_g \in \mathbb{R}^2$. We use the same policy and task cost parameterization as the `rope-swing` task.

3.1.5 Real Roll Goal-Conditioned Task (`roll-gc-real`)

In this task we perform a goal-conditioned ball rolling task, using a real MELFA-Assista robot with a block end-effector for striking the ball. We place a billiards table in front of the robot and place the cue ball in a fixed position in front of the robot. The state space $s_t \in \mathbb{R}^2$ is the *pixel* location of the ball tracked in a top-down camera view. The task is to strike the ball causing it to roll to the sampled goal pixel location $\tau_g \in \mathbb{R}^2$. We use the same task cost function and policy parameterization as the slide goal conditioned task (`slide-gc`).

4 Method

When performing a dynamic manipulation task, it is often not feasible to expect success on the first try, as important task parameters may not be readily observable a priori or due to the complexity of the task. Instead, several interactions with the system allow one to observe an outcome and correct course. For example, [28] uses exploratory interactions to understand physical parameters before

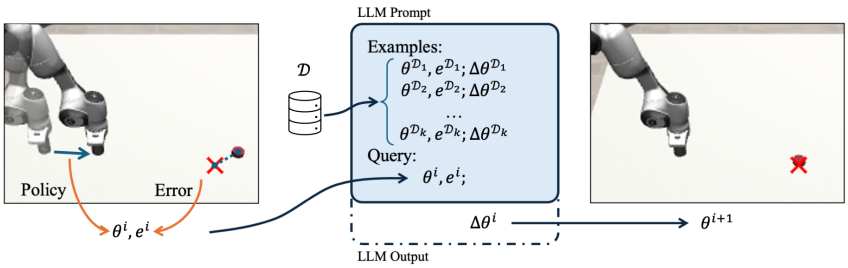


Figure 2: Overview of our proposed In-Context Policy Improvement (ICPI) method. We tokenize policy parameters and error of the current policy and provide it, along with examples from a small policy improvement dataset to the LLM in the prompt. The LLM then outputs the delta policy parameters.

executing a dynamic manipulation task. In [5], the authors propose executing a task, then learning how to adjust the trajectory to improve performance. While these demonstrate the value of iterative reasoning for dynamic tasks, they require large training datasets and complex neural architectures. We investigate if a similar iterative policy approach can be achieved utilizing the *in-context learning* ability of LLMs pretrained on language data.

4.1 Policy Improvement Operator

Our goal is ultimately to solve the optimization in Eq. 1. We propose to solve this by attempting to iteratively improve our policy parameterization θ. To achieve this, we formulate a *policy improvement operator* f, as a mapping that takes in the current best policy parameterization estimate θ^i and state trajectory $s^i_{1:T}$ and outputs a change to the policy $\Delta\theta^i$:

$$f(\theta^i, s^i_{1:T}) \rightarrow \Delta\theta^i \tag{4}$$

The improved policy is then $\theta^{i+1} = \theta^i + \Delta\theta^i$. To fit this operator, we assume access to a dataset $\mathcal{D} = \{d_1, d_2, ..., d_N\}$ of policy improvement labels, $d_i = (\theta^i, s^i_{1:T}, \Delta\theta^i)$. We discuss how this dataset can be collected in Sec. 4.4.

In the typical approach to solving for this data-driven operator, f would be setup as a data-driven model, such as a neural network, which is trained using the dataset $\mathcal{D}$. Here we instead propose to utilize *in-context learning*, leveraging pre-trained LLMs by inputting both our query and the dataset $\mathcal{D}$ into the LLM prompt:

$$f(\theta^i, s^i_{1:T}, \mathcal{D}) \rightarrow \Delta\theta^i \tag{5}$$

The result is a sample-efficient method that requires no model design or training, but rather fits the operator all in the forward pass of the pre-trained model.

We utilize the LLM by treating our policy improvement operator as a sequence-to-sequence completion task. The input sequence is the current policy and state trajectory and the output sequence is the policy change. We use the provided dataset to feed example input-output sequences, and the LLM is instructed to complete the sequence for the current policy information to estimate the corresponding policy update:

$$f(\{\theta^{\mathcal{D}_j}, s^{\mathcal{D}_j}_{1:T}, \Delta\theta^{\mathcal{D}_j}\}_{j=1}^k, \theta^i, s^i_{1:T}) \rightarrow \Delta\theta^i \tag{6}$$

We call our proposed method In-Context Policy Improvement (ICPI) (Fig. 2). Next, we highlight how we tokenize the policy and state trajectories for input to the LLM and discuss how examples are selected from the dataset to form our query.

4.2 Tokenization

To input our policy and state trajectory information, we must tokenize the information to text which the LLM can parse and reason over. Following existing in-context learning approaches [17, 6], we encode our policy and state information as numeric characters. For our policy and policy update values, θ and $\Delta\theta$, we simply tokenize each term as characters.

For the state trajectory $s^i_{1:T}$, we design a task-specific encoding to reflect the error of the execution. In many tasks, the relative change in an outcome is consistent even for differing task parameters. For

example, two pucks of different mass may behave differently, but if each is falling short of the goal, we want to push "harder" in both cases. Thus, correcting for similar relative errors may involve similar $\Delta\theta$. As such, we encode the state trajectory information as the relative error to the goal, $s^i_{1:T} \rightarrow e^i = s^i_t - \tau_g$. t is selected according to the task. For the `slide` and `roll` tasks, $t = T$. For `rope-swing`, $t = \arg\min_t ||s_t - \tau_g||_2$. This relative error is then tokenized as characters.

4.3 Dataset Example Selection

To enable in-context learning, we select a set of example input-output sequences from our dataset $\mathcal{D}$ to be provided along with the current policy execution input. In order to limit the size of the inputs to the LLM and focus the examples on "useful" examples, we utilize a K-Nearest Neighbors (KNN) lookup to find similar examples. In particular, we construct a k-D Tree using the vector $v^{\mathcal{D}_j} = [\theta^{\mathcal{D}_j}, e^{\mathcal{D}_j}]$, where we normalize the policy and error terms to have comparable scale along each dimension. For a query example $v^i = [\theta^i, e^i]$, we can then perform a KNN lookup to find the most similar examples in our dataset $\mathcal{D}$. Following existing ICL work [17], we then provide these k examples in order of decreasing distance to the query example. In our experiments, we set $k = 20$.

4.4 Dataset Construction

Finally, we discuss the creation of our dataset $\mathcal{D}$. For each task instance $\tau \sim P(\tau)$, let θ^* be a solution to Eq. 1. For any nearby execution θ, we can construct the delta label $\Delta\theta = \theta^* - \theta$. θ^* can be derived several different ways, including by demonstration or via some other algorithmic approach to solving Eq. 1, such as Reinforcement Learning or brute force search. Our method then learns to make adjustments based on the results of these laborious methods. This can be seen as a form of algorithm distillation [11], where we distill the progress of some other algorithm efficiently into our new sequence-to-sequence problem.

In practice, we construct our training data on a per-task basis. For the simulated `slide` and `rope-swing` setups, we derive θ^* using a brute-force search algorithm to find a solution θ^*. For our `roll-gc-real` task, executing a brute-force search on the real setup would be too slow. Since we are solving a goal conditioned task, we can assign goal labels in hindsight based on the final state of the policy rollout [15]. We can then generate our dataset by sampling alternative θ parameterizations, and computing the error to the final state of the "guiding" execution.

5 Results

5.1 Iterative Policy Improvement

First, we investigate how our proposed ICPI method performs and provide comparison to baselines. While more sophisticated modeling and learning techniques are state-of-the-art for dynamic manipulation, these methods generally require large-scale datasets, e.g. 54 million examples for rope swing in [5] or ∼5000 steps for Reinforcement Learning of puck sliding in [14]. As such, we focus our comparison to data-efficient baselines and alternate LLM-based methods.

5.1.1 Baselines

Random Shooting: we set up a random shooting method that samples a change to the current policy θ^i by sampling a randomized offset $\Delta\theta^i \sim \mathcal{N}(0; 0.5 \cdot c^i \cdot \Delta\theta_{\max})$, where $\Delta\theta_{\max} = \theta_{\max} - \theta_{\min}$.

Bayes Opt: this baseline seeks to directly solve Eq. 1 as a Bayesian optimization problem [18].

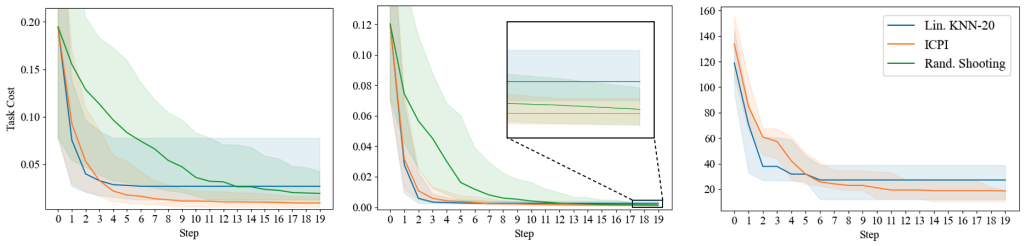
KNN-k: this baseline solves the policy iteration by performing only the KNN lookup within $\mathcal{D}$ as described in Sec. 4.3 and takes the average of the k closest labels. We set $k = 5$.

Linear KNN-k: this baseline performs the KNN lookup as described in Sec. 4.3 and fits a linear model to the k closest input-output labels, and inputs v^i to the linear model. This forms a piece-wise linear model of the policy improvement operator. We match ICPI and set $k = 20$.

In-Context Sequence-Improvement (ICSI): This method is based on Sequence Improvement technique introduced by [17]. In this method, they use cost conditioning to prompt a LLM to improve over previous iterations. We apply their method here by pairing policy parameters θ^i with their cost

Method	`slide`	`slide-gc`	`rope-swing`	`rope-swing-gc`	`roll-gc-real`
Rand. Shooting	0.037 (0.050)	0.077 (0.055)	**0.007 (0.022)**	0.004 (0.009)	-
Bayes Opt	0.054 (0.065)	0.087 (0.052)	0.019 (0.018)	0.014 (0.012)	-
KNN-5	0.106 (0.085)	0.071 (0.043)	0.020 (0.023)	0.010 (0.015)	-
Lin. KNN-20	0.053 (0.060)	**0.022 (0.017)**	0.042 (0.076)	0.006 (0.017)	33.824 (28.031)
ICSI [17]	0.030 (0.055)	0.107 (0.058)	0.042 (0.067)	0.021 (0.035)	-
In-Weights	0.029 (0.055)	0.102 (0.060)	0.027 (0.062)	0.022 (0.037)	-
ICPI (Ours)	**0.013 (0.014)**	0.025 (0.026)	**0.007 (0.012)**	**0.002 (0.006)**	**17.107 (9.796)**

Table 1: Final Best-Policy Mean Performance Comparison $C_\tau^{\texttt{task}}$ (↓)



(a) `slide` (b) `rope-swing-gc` (c) `roll-gc-real`

Figure 3: Task Cost convergence plots for the best policy so far at each step across three of our tasks comparing random shooting, piece-wise linear modeling, and our proposed method.

c^i. We then query the LLM to generate a new policy conditioned on previous c^i, θ^i examples, and prompting with an improved c^{i+1}, asking the LLM to predict the corresponding improved θ^{i+1}.

In-Weights Reasoning (IW): This method seeks to utilize the *in-weights reasoning* capability of LLMs for our tasks. For each task, we describe in natural language the task setup and action space. We also provide the previous executions as θ^i, e^i. We then directly query the model to reason about the system and provide a better policy. This baseline seeks to determine how well *in-weights reasoning* performs on our target tasks.

5.1.2 Policy Improvement Results

We generate a dataset $\mathcal{D}$ of ∼300 policy improvement examples as described in Sec. 4.4. For our simulated tasks (`slide`, `slide-gc`, `rope-swing`, `rope-swing-gc`) we run each policy iteration method on 100 sampled tasks. For our real task (`roll-gc-real`) we execute 10 sampled tasks and only compare to the piece-wise linear method, as we found it was one of the most competitive methods in simulation. All methods are initialized with the same starting action for fair comparison.

In Table 1, we show the average and standard deviation for the best policy cost after 20 iterations. We found that in most tasks, our proposed ICPI method outperformed the baselines, and was the only method that consistently performed well across all tasks. In Fig. 3, we compare the evolution of the best policy cost as a function of iteration step for random shooting, piece-wise linear (Lin. KNN-20), and our method. We see that ICPI consistently outperforms random samples and converges to a better result than the piece-wise linear approach, including on the real robot. Qualitative policy iteration results for ICPI are shown in Figs. 1 and 4.

Our in-context method outperformed utilizing in-weights reasoning on our task. This highlights that while dynamic reasoning may not be readily available in LLMs, their in-context abilities can still be useful. ICPI also outperformed the ICSI approach [17], showing the value of our proposed policy improvement formulation for unlocking the benefits of in-context learning.

5.2 Design Ablations

We next investigate how design decisions impact ICPI performance. In particular, we look at two variations on our method. In the first variation, we exchange our tokenization method, and instead

Inputs	LLM Model	`slide`	`slide-gc`	`rope-swing`	`rope-swing-gc`
θ^i, e^i	`gpt-3.5-turbo`	0.031 (0.045)	0.033 (0.026)	0.013 (0.019)	0.007 (0.014)
θ^i, e^i	`gpt-4o-mini`	0.026 (0.035)	0.039 (0.031)	0.010 (0.015)	0.009 (0.018)
θ^i, s_t^i, τ_g	`gpt-4o`	0.017 (0.022)	0.055 (0.050)	0.002 (0.002)	0.006 (0.012)
θ^i, e^i	`gpt-4o`	0.013 (0.014)	0.025 (0.026)	0.007 (0.012)	0.002 (0.006)

Table 2: ICPI Ablations on Final Best-Policy Mean Performance Comparison $C_\tau^{\texttt{task}}$ (↓). Last row is settings used for our experiments.

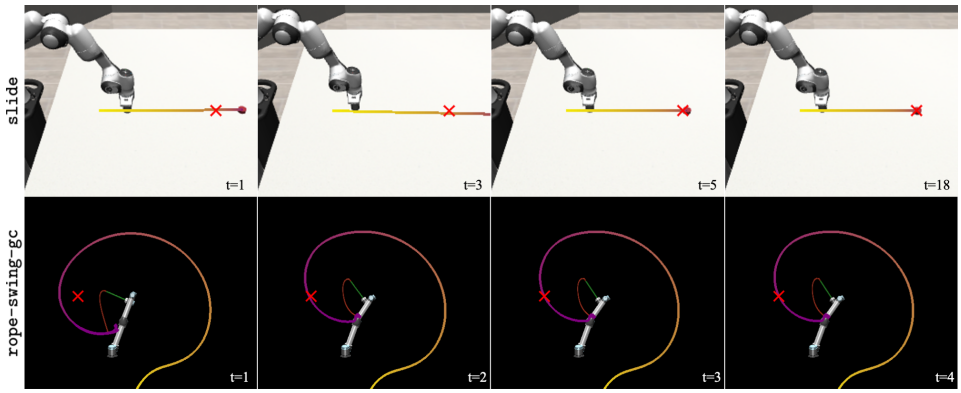


Figure 4: Qualitative examples of iterative in-context policy improvement using our proposed ICPI for the `slide` and `rope-swing-gc` tasks.

of directly providing e^i, we instead provide both s_t^i, τ_g to the model. Second, we investigate how the choice in LLM affects our method performance. The comparisons across our simulated tasks are shown in Tab. 2. We find that model choice has a notable impact on in-context learning performance. While we are not privy to all model information, we notice that the newer, large-scale reasoning model (`gpt-4o`) seems to outperform older (`gpt-3.5-turbo`) and smaller (`gpt-4o-mini`) models. We also see that directly providing the relative error e^i outperforms providing the state and goal s_t^i, τ_g separately, even though both contain the same information.

6 Discussion

We proposed In-Context Policy Improvement (ICPI), utilizing pre-trained Large Language Models (LLMs) for sample-efficient dynamic manipulation policy improvement. We demonstrate that while LLMs struggle to apply *in-weights* reasoning to dynamic tasks, *in-context learning* outperforms alternative policy iteration methods, both in simulation and on a real robot. This work adds to a growing body of work showing the utility of in-context learning in robotics [17, 6], which we hope motivates further investigation, both utilizing non-robotics and robotics specific transformer models.

6.1 Limitations

A limitation of our proposed method is the overhead of utilizing large pretrained models. These models incur both computational, financial, and environmental overhead [1] and the low-level details of models are not available, which can make it difficult to understand model choice impact. Our method relies on a policy improvement dataset, which may be difficult to collect for complex tasks. Self-play [15] and demonstrations [6] could be utilized to ease the data collection burden.

In this work, found that some amount of feature selection was important to task success (see drop in performance when providing s^i, τ_g vs. e^i in Sec. 5.2). This seems to indicate that in-context learning may not yet be capable of the feature learning ubiquitous in modern machine learning. While signs indicate that in-context learning may continue to improve with improved LLMs (see `gpt-4o` vs. `gpt-3.5-turbo` in Sec. 5.2 and [17]), it is unclear if more advanced pattern-based reasoning will be achievable purely in-context. Additionally, we only investigated relatively low-dimensional input-

output formats; while low-dimensional parametric actions are common in dynamic manipulation, they can limit the dexterity of the system. Our results present sufficient features for our proposed tasks - we leave to future work an extensive study of task dimensionality and feature extraction for in-context learning.

References

[1] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell. On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, FAccT '21, page 610–623, New York, NY, USA, 2021. Association for Computing Machinery.

[2] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020.

[3] S. Chan, A. Santoro, A. Lampinen, J. Wang, A. Singh, P. Richemond, J. McClelland, and F. Hill. Data distributional properties drive emergent in-context learning in transformers. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 18878–18891. Curran Associates, Inc., 2022.

[4] S. C. Chan, I. Dasgupta, J. Kim, D. Kumaran, A. K. Lampinen, and F. Hill. Transformers generalize differently from information stored in context vs in weights. *arXiv preprint arXiv:2210.05675*, 2022.

[5] C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song. Iterative Residual Policy for Goal-Conditioned Dynamic Manipulation of Deformable Objects. In *Proceedings of Robotics: Science and Systems*, New York City, NY, USA, June 2022.

[6] N. Di Palo and E. Johns. Keypoint action tokens enable in-context imitation learning in robotics. In *Proceedings of Robotics: Science and Systems*, 2024.

[7] H. Ha and S. Song. Flingbot: The unreasonable effectiveness of dynamic manipulation for cloth unfolding. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 24–33. PMLR, 08–11 Nov 2022.

[8] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 540–562. PMLR, 06–09 Nov 2023.

[9] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. T. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K.-H. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu. Do as i can, not as i say: Grounding language in robotic affordances. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Proceedings of The 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR, 14–18 Dec 2023.

[10] T. Kwon, N. D. Palo, and E. Johns. Language models as zero-shot trajectory generators. *IEEE Robotics and Automation Letters*, 9(7):6728–6735, 2024.

[11] M. Laskin, L. Wang, J. Oh, E. Parisotto, S. Spencer, R. Steigerwald, D. Strouse, S. Hansen, A. Filos, E. Brooks, et al. In-context reinforcement learning with algorithm distillation. *ICLR*, 2023.

[12] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 9493–9500, 2023.

[13] V. Lim, H. Huang, L. Y. Chen, J. Wang, J. Ichnowski, D. Seita, M. Laskey, and K. Goldberg. Real2sim2real: Self-supervised learning of physical single-step dynamic actions for planar robot casting. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8282–8289, 2022.

[14] Y. Luo, Y. Wang, K. Dong, Y. Liu, Z. Sun, Q. Zhang, and B. Song. Sirl: Self-imitation reinforcement learning for single-step hitting tasks. In *2023 International Conference on Advanced Robotics and Mechatronics (ICARM)*, pages 185–190. IEEE, 2023.

[15] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1113–1132. PMLR, 30 Oct–01 Nov 2020.

[16] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *EMNLP*, 2022.

[17] S. Mirchandani, F. Xia, P. Florence, B. Ichter, D. Driess, M. G. Arenas, K. Rao, D. Sadigh, and A. Zeng. Large language models as general pattern machines. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 2498–2518. PMLR, 06–09 Nov 2023.

[18] F. Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[19] C. Olsson, N. Elhage, N. Nanda, N. Joseph, N. DasSarma, T. Henighan, B. Mann, A. Askell, Y. Bai, A. Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[20] A. Srivastava, A. Rastogi, A. Rao, A. A. M. Shoeb, A. Abid, A. Fisch, A. R. Brown, A. Santoro, A. Gupta, A. Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

[21] L. Sun, D. K. Jha, C. Hori, S. Jain, R. Corcodel, X. Zhu, M. Tomizuka, and D. Romeres. Interactive planning using large language models for partially observable robotic tasks. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 14054–14061. IEEE, 2024.

[22] M. Suzgun, N. Scales, N. Schärli, S. Gehrmann, Y. Tay, H. W. Chung, A. Chowdhery, Q. V. Le, E. H. Chi, D. Zhou, et al. Challenging big-bench tasks and whether chain-of-thought can solve them. In *ACL (Findings)*, 2023.

[23] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[24] C. Wang, S. Wang, B. Romero, F. Veiga, and E. Adelson. Swingbot: Learning physical features from in-hand tactile exploration for dynamic swing-up manipulation. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5633–5640, 2020.

[25] Y. Wang, Z. Sun, J. Zhang, Z. Xian, E. Biyik, D. Held, and Z. Erickson. RL-VLM-f: Reinforcement learning from vision language foundation model feedback. In *Forty-first International Conference on Machine Learning*, 2024.

[26] Z. Wang and A. H. Qureshi. Implicit physics-aware policy for dynamic manipulation of rigid objects via soft body tools. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2025.

[27] Z. Xu, C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song. Dextairity: Deformable manipulation can be a breeze. In *Proceedings of Robotics: Science and Systems (RSS)*, 2022.

[28] Z. Xu, J. Wu, A. Zeng, J. Tenenbaum, and S. Song. Densephysnet: Learning dense physical object representations via multi-step dynamic interactions. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019.

[29] Z. Yang, L. Li, K. Lin, J. Wang, C.-C. Lin, Z. Liu, and L. Wang. The dawn of lmms: Preliminary explorations with gpt-4v (ision). *arXiv preprint arXiv:2309.17421*, 2023.

[30] M. Yoo, W. K. Kim, and H. Woo. In-context policy adaptation via cross-domain skill diffusion. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 22191–22199, 2025.

[31] W. Yu, N. Gileadi, C. Fu, S. Kirmani, K.-H. Lee, M. G. Arenas, H.-T. L. Chiang, T. Erez, L. Hasenclever, J. Humplik, B. Ichter, T. Xiao, P. Xu, A. Zeng, T. Zhang, N. Heess, D. Sadigh, J. Tan, Y. Tassa, and F. Xia. Language to rewards for robotic skill synthesis. In J. Tan, M. Toussaint, and K. Darvish, editors, *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pages 374–404. PMLR, 06–09 Nov 2023.

[32] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser. Tossingbot: Learning to throw arbitrary objects with residual physics. *IEEE Transactions on Robotics*, 36(4):1307–1319, 2020.

[33] H. Zhang, J. Ichnowski, D. Seita, J. Wang, H. Huang, and K. Goldberg. Robots of the lost arc: Self-supervised learning to dynamically manipulate fixed-endpoint cables. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4560–4567, 2021.

[34] X. Zhang, S. Liu, P. Huang, W. J. Han, Y. Lyu, M. Xu, and D. Zhao. Dynamics as prompts: In-context learning for sim-to-real system identifications. *IEEE Robotics and Automation Letters*, 2025.

A Appendix

A.1 Example Prompts

We provide example prompts and responses for our method along with the other LLM baselines. The examples provided are for the simulated `slide` task.

A.1.1 In-Context Policy Improvement (Ours)

See Sec. 4 for in-depth discussion of how we construct our prompts. We note the task-agnostic header, which is fixed across our experiments.

```
Prompt:
You are a pattern generator machine.  I will give you a series of patterns with
INPUTS and OUTPUTS as examples.  Then you will receive a new INPUTS, and you have to
generate OUTPUTS following the pattern that appears in the data.

Patterns are provided per-line as:  INPUTS;OUTPUTS

Only reply with an estimate for the OUTPUTS. OUTPUTS should be 3 values separated by
spaces.

-0.061 0.242 0.771 -0.001 -0.017;-0.022 0.002 -0.065
0.050 0.284 0.830 0.027 0.001;0.024 -0.007 -0.059
...
0.065 0.276 0.723 -0.086 0.002;0.009 0.001 0.049
-0.031 0.252 0.672 -0.076 -0.011;0.039 -0.036 -0.137

-0.016 0.269 0.780 -0.155 -0.004;

Response:
0.004 -0.011 -0.060
```

A.1.2 In-Context Sequence-Improvement [17]

Following **(author?)** [17], we perform sequence improvement in-context. We follow their proposed cost prompting strategy of prompting with a randomly sampled improved cost.

```
Prompt:
You are a pattern generator machine.  I will give you a series of patterns with
INPUTS and OUTPUTS as examples.  Then you will receive a new INPUTS, and you have to
generate OUTPUTS following the pattern that appears in the data.

Patterns are provided per-line as:  INPUTS;OUTPUTS

Only reply with an estimate for the OUTPUTS. OUTPUTS should be 3 values separated by
spaces.

0.550;0.153 0.112 0.812
0.528;0.464 0.112 0.500
...
0.094;0.298 0.350 0.825
0.044;0.153 0.269 0.825

0.000;

Response:
0.153 0.112 0.825
```

A.1.3 In-Weights Reasoning

Our final LLM baseline seeks to employ the *in-weights* capabilities of the model. As such, we provide a detailed description of each task, setting, and action space, along with previous interac-

tions. The descriptions for the other tasks are provided in similar specificity to attempt to ground the in-weights reasoning to the task.

```
Prompt:
You are a planner responsible for providing an action for a robot to execute.  You
should reason carefully about the physics of the scenario when planning.  You may
receive a description of previous interactions with the target system and should
consider these previous interactions when deciding how to act.  Each task is solved
via a single action execution - it does not take multiple actions to complete the
task.
Only reply with the action the robot should execute, as a space-separated list of
numbers.
A robot is positioned in front of a table.  The robot has an arm which is
extended over the table.  The tool connected to the robot is a cylinder, starting
perpendicular to the tabletop.  In front of the robot arm on the table is a puck,
which is also a cylinder, and it can move freely on the tabletop.  The task is to
use the robot arm to slide the puck on the table to a goal position.  The robot tool
and the puck start so that planar motions over the tabletop will cause the two to
collide (i.e., no vertical motion of the robot tool is required).  The robot has a
single action to complete the task - it cannot take multiple actions.
The robot tool starts at location (0.000, 0.000) on the tabletop.  The puck starts
at location (0.100, 0.000) on the tabletop.  The goal is to move the puck to (0.800,
0.000).
The action space of the robot is the polar coordinates for a planar motion of the
robot tool relative to its starting position and the time it takes to complete
the motion.  The first term is the angle of motion, where 0 degrees moves straight
forward along the x dimension.  The second term is the distance in that direction
the robot will move.  The third term is the time it will take to complete the
motion.
The actions are bounded.  The angle of motion must lie between -0.464 and 0.464
radians.  The distance must be between 0.112 and 0.350 meters.  The motion can take
between 0.500 and 5.000 seconds to complete.  The provided actions will be clipped
into this range.  The action is mapped to a dense set of tool locations via linear
interpolation and executed on the system.
The robot has already interacted with the system several times.  You will receive
a set of descriptions of these interactions.  You should consider the previous
interactions and how to improve upon them when selecting the next action to attempt.
The previous interactions will be provided with one interaction per line.  Each line
will describe the interaction as the action taken and the final position of the puck
relative to the goal location, separated by a semi-colon.  Each item is provided as
a space-separated list of values.

0.000 0.350 0.780;-0.654 -0.000
0.000 0.350 0.780;-0.654 -0.000
...
0.000 0.269 0.780;-0.162 -0.000
-0.016 0.269 0.780;-0.155 -0.004

Response:
0.000 0.350 0.780
```