HYDRA: Hybrid Robot Actions for Imitation Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract: Imitation Learning (IL) is a sample efficient paradigm for robot learning using expert demonstrations. However, policies learned through IL suffer from state distribution shift at test time, due to compounding errors in action prediction which lead to previously unseen states. Choosing an action representation for the policy that minimizes this distribution shift is critical in imitation learning. Prior work propose using temporal action abstractions to reduce compounding errors, but they often sacrifice policy dexterity or require domain-specific knowledge. To address these trade-offs, we introduce HYDRA, a method that leverages a hybrid action space with two levels of action abstractions: *sparse high-level waypoints* and *dense low-level actions*. HYDRA dynamically switches between action abstractions at test time to enable both coarse and fine-grained control of a robot. In addition, HYDRA employs action relabeling to increase the consistency of actions in the dataset, further reducing distribution shift. HYDRA outperforms prior imitation learning methods by 30-40% on seven challenging simulation and real world environments, involving long-horizon tasks in the real world like making coffee and toasting bread. Videos are found on our website: https://tinyurl.com/3mc6793z

1 Introduction

In recent years, supervised learning methods have made remarkable advancements in computer vision (CV), natural language processing (NLP), and human-level game playing [1, 2, 3, 4, 5, 6, 7]. In robotics, *imitation learning* (IL) has emerged as a data-driven and sample efficient approach for programming robots using expert demonstrations. More specifically, behavioral cloning (BC) methods treat IL as a supervised learning problem and directly train a policy to map states to actions. BC methods are often favored in practice for their simplicity but suffer from the well-known distribution shift problem, where the test time state distribution deviates from the training state distribution, primarily caused by the accumulation of errors in action predictions [8, 9, 10].

Broadly, prior work has explored reducing distribution shift by interactively adding new data [9], incorporating large prior datasets [11, 12], choosing better state representations (inputs) [13, 14], or altering model or loss structure [15, 16, 14]. A less explored but critical factor is the *action* representation (outputs): action prediction error partially stems from how difficult it is for the policy to capture the expert demonstrated actions, so action representations are a critical line of defense against distribution shift. Prior work studying action representations generally fall into two camps: (1) methods that use *temporal abstractions* to treat long action sequences as a single action (i.e., reducing the effective task horizon) and thus reduce the potential for compounding errors, and (2) methods that make the action representation more *expressive* to minimize the single-step prediction error [17, 18, 16, 19, 15]. However, both approaches come with a number of shortcomings.

Methods using *temporal abstractions* often come at the cost of either the dexterity of the robot or the generality to new settings. One prior approach is for the robot to follow waypoints that cover multiple time steps [17, 14]; however, waypoints alone are not reactive enough for dynamic, dexterous action sequences (e.g., inserting a coffee pod). Other works use structured movement

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

primitives that can capture more dynamic behaviors like skewering food items or helping a person get dressed [20, 18, 21], but relying on pre-defined primitives often sacrifice generalizability to new settings (e.g., new primitives beyond skewering for food manipulation). Today, we lack temporal abstractions that reduce distribution shift without losing policy dexterity and generality.

Other methods design each action to be more *expressive* to capture the multi-modality present in human behavior [19, 15, 16]; however, these expressive action spaces often lead to overfitting, high training time, or complex learning objectives. Rather than making the policy more expressive, a more robust approach is to make the actions in the dataset more *consistent* at a given state and easier to learn (e.g., showing one consistent way to insert a coffee pod rather than many conflicting ways). Prior work shows that more action consistency (e.g., consistent human demonstrators) with sufficient state coverage lead to better policies [19, 14, 22], likely by reducing online policy errors [23].

To enable both a better temporal abstraction and more consistent actions in the dataset, our key insight is to leverage the fact that most robotics tasks are hierarchical in nature – they can be divided into two distinct *modes* of behaviors: *reaching high-level waypoints* (e.g., free-space motion) or *fine-grained manipulation* (e.g., object interaction). Then, we can learn a policy that dynamically switches between these modes – this is in fact similar to models of human decision making, where it is widely believed that humans can discover action abstractions and switch between them during a task [24, 25]. Capturing both waypoints and fine-grained actions enables us to compress action sequences (i.e., reduce distribution shift) without sacrificing the dynamic parts of the task, thus maintaining dexterity. In practice, this abstraction is general enough to represent most tasks in robot manipulation. Another notable advantage of partitioning tasks into two modes is that, during the waypoint reaching phase, we can *relabel* our actions with more consistent waypoint-following behaviors, thus increasing action consistency in the dataset.

Leveraging this insight, we propose HYDRA, a method that dynamically switches between two action representations: *sparse* waypoint actions for free-space linear motions and *dense*, single-step delta actions for contact-rich manipulation. HYDRA learns to switch between these action modes with human-labeled modes, which are provided after or during data collection with minimal additional effort. In addition, HYDRA *relabels* low-level actions in the dataset during the waypoint periods – where the robot is moving in free space (e.g., when reaching a coffee pod) to follow consistent paths. These consistent actions simplify policy learning, which reduces action prediction error in the dataset overall and thus reduces distribution shift. HYDRA outperforms baseline imitation learning approaches across seven challenging, long-horizon manipulation tasks spanning both simulation and the real world. In addition, it is able to perform a complex coffee making task involving many high precision stages with 80% success, 4x the performance of the best baseline, BC-RNN.

2 Related Work

Data Curation: Several prior works aim to *curate* data based on some notion of data quality, in order to reduce distribution shift. Most works define quality as the state diversity present in a dataset, To increase state diversity, Ross et al. [9] proposed to interactively collect *on-policy* demonstration data, but this requires experts to label actions for newly visited states. To reduce expert supervision, some methods use interventions to relabel on-policy data, where interventions can be automatically or human generated [26, 27, 28, 29, 22, 30]. Laskey et al. [31] inject noise during data collection to increase state diversity to achieve similar performance as interactive methods. Recent work has sought to formalize a broader notion of data quality beyond just state diversity [23]. HYDRA takes this broader definition into account, increasing data quality through action consistency.

Model and State Priors: Rather than changing the data, many prior works build in structure to the model itself to address distribution shift. Object-centric state representations have been shown to make policies more generalizable [13]. Similarly, pretrained state representations trained on multi-task data have been shown to improve sample efficiency and robustness [12, 32]. Adding structure into the model itself, for example using implicit representations or diffusion-based policies, has also been shown to improve performance [16, 15]. The changes in HYDRA affect the action space and thus are compatible with many of these prior approaches.

Action Representations: Another approach is to change the action representation to reduce compounding errors. One category of prior works leverage *temporal action abstractions* to reduce the number of policy steps. Several works have learned skills from demonstrations, usually requiring lots of data but struggling to generalize [33, 34, 14]. Others use parameterized action primitives or motion primitives, but despite being more sample efficient, these often require privileged state information or are not general enough for all scenes [20, 18, 21]. Waypoint action spaces have also been shown to be a sample efficient temporal abstraction; however, they fail to capture dynamic and dexterous tasks in the environment [35, 16]. For more dexterity, Johns [36] proposes Coarse-to-Fine Imitation Learning by modeling a single demonstrated trajectory as two parts: an approaching trajectory followed by an interaction trajectory. This approach, however, cannot easily scale to multi-step manipulation tasks with multiple stages of unstructured object interaction. HYDRA builds on these works, combining waypoints and low-level actions into one model to reduce compounding errors without losing dexterity or generality. Another category of works seek to increase the *expressivity* of a single action to reduce action prediction error, for example with Gaussian mixture models or energy models [19, 15, 16]. However, increasing expressivity often leads to overfitting, more complex learning objectives, and increased training and evaluation time. Instead of increasing expressivity, HYDRA takes a more robust approach by increasing action *consistency* in the data. Prior work shows the importance of consistent actions for minimizing distribution shift [19, 23]. HYDRA relabels actions in the dataset after data collection to increase consistency.

3 Preliminaries

Imitation learning (IL) assumes access to a dataset $\mathcal{D} = \{\tau_1, \ldots, \tau_n\}$ of n expert demonstrations. Each demonstration τ_i is a sequence of observation-action pairs of length N_i, $\tau_i = \{(o_1, a_1), \ldots, (o_{N_i}, a_{N_i})\}$, with observations $o \in \mathcal{O}$ and actions $a \in \mathcal{A}$. $\mathcal{O}$ often consists of robot proprioceptive data such as end effector poses and gripper widths, denoted $s_p \in \mathcal{P}$, as well as environment observations such as images or object poses, denoted $s_e \in \mathcal{E}$, such that $\mathcal{O} = \mathcal{P} \oplus \mathcal{E}$. The true state of the environment is $s \in \mathcal{S}$. In robotics, the action space usually consists of either torque, velocity, or position commands for the robot. While velocity actions are most common, prior works also use position actions in the form of target waypoints [14, 35]. The IL objective is to learn a policy $\pi_\theta : \mathcal{O} \to \mathcal{A}$ mapping from observations to actions via the supervised loss:

$$\mathcal{L}(\theta) = -\mathbb{E}_{(o,a)\sim p_{\mathcal{D}}} \left[\log \pi_\theta(a|o)\right] \tag{1}$$

At test time, the learned policy π_θ is rolled out under environment dynamics $f : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$. Per step, we observe o_t, sample an action $\tilde{a}_t \sim \pi(\cdot|o_t)$, and obtain the next state $s_{t+1} = f(s_t, \tilde{a}_t)$.

Distribution Shift in IL. A fundamental challenge with imitation learning is state *distribution shift* between training and test time. Considering training sample $(\ldots o_t, a_t, o_{t+1} \ldots)$: if the learned policy outputs $\tilde{a}_t \sim \pi(\cdot|o_t)$, which has a small action error $\epsilon_t = \tilde{a}_t - a_t$, the next state following this action will also deviate: $\tilde{s}_{t+1} = f(s_t, a_t + \epsilon_t)$, which in turn affects the policy output at the next step. For real world dynamics, this change in next state can be highly disproportionate to $||\epsilon_t||$. For example in the coffee task in Fig. 1, with a slight change in gripper position (small ϵ_t) the policy can misgrasp the coffee pod (large change in s_{t+1} and o_{t+1}). Furthermore, noise in the dynamics f can lead to even larger changes in o_{t+1}. As we continue to execute for the next $N - t$ steps, this divergence from the training distribution can compound, often leading to task failure.

Therefore, reducing distribution shift requires reducing ϵ_t for all $t \in \{1, \ldots, N\}$ or increasing the coverage of states s_t. One approach to reduce policy error is increasing **action consistency**, which prior work defines as lowering the entropy of the expert policy π_E at each state: $\mathcal{H}_{\pi_E}(a|s)$ [23]. However, there is a trade-off between state coverage and action consistency during data collection, since less consistent actions often lead to more diverse states [23, 19]. HYDRA reduces distribution shift by using a temporal abstraction for the action space – which shortens the number of policy steps N and thus reduces compounding errors – and by improving action consistency in offline data – which reduces $||\epsilon_t||$ without reducing state coverage.

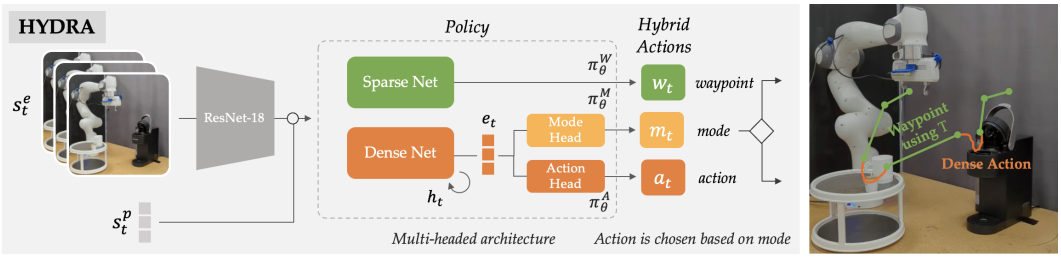

Figure 1: Multi-headed architecture of HYDRA: During training, we learn to predict waypoints, low level actions, and the mode label for each time step. One network (Dense Net) predicts the low level action a_t and the mode m_t; both the action and mode heads of Dense Net share an intermediate representation e_t. A separate network (Sparse Net) predicts the high level waypoint w_t. At test time, we sample m_t and either servo to reach a waypoint ($m_t = 0$) without requerying the policy, or follow a dense action for one time step ($m_t = 1$). An example of how sparse and dense modes can be arbitrarily stitched together at test time is shown on the right.

4 HYDRA: A Hybrid Action Representation

To reduce distribution shift, our insight is that most robot manipulation tasks are a combination of *sparse* waypoint-reaching, such as reaching for an object or lifting a mug towards a shelf, and *dense* low-level actions, such as grasping an object or balancing a mug stably on a shelf. Waypoints capture free-space motions but struggle to capture dexterous or precise behaviors. Conversely, low-level actions capture these dynamic behaviors but are often redundant during long free-space motions.

Instead of learning from only velocities or waypoints, HYDRA learns a *hybrid action representation* consisting of both high-level waypoints in the robot's proprioceptive space $w \in \mathcal{P}$ and low-level actions $a \in \mathcal{A}$. Additionally, we learn to dynamically switch between these modes by predicting which mode $m \in \{0, 1\}$, sparse or dense, should be executed at each demonstrated state. Mode labels are annotated with little extra cost by experts either during or after data collection. This flexible abstraction leads to (1) a compressed action space that reduces compounding errors without sacrificing dexterity or generality, and (2) a more consistent, simple low-level action distribution through action relabeling during the sparse periods. This section presents an overview of the approach, followed by discussions on mode labeling, action relabeling, and training/testing procedures.

Overview: The multi-headed architecture of HYDRA is outlined in Fig. 1, with heads $\pi_\theta^M : \mathcal{O} \to \{0, 1\}$, $\pi_\theta^A : \mathcal{O} \to \mathcal{A}$, $\pi_\theta^W : \mathcal{O} \to \mathcal{P}$, for mode, action, and waypoint respectively. One network, Dense Net, predicts the low-level action a_t and the mode m_t at each input $o_t = \{s_t^e, s_t^p\}$. Another network, Sparse Net, separately outputs the desired *future* waypoint w_t for input o_t. We assume waypoints can be reached using a known controller $\mathrm{T} : \mathcal{O} \times \mathcal{P} \to \mathcal{A}$ which converts the state and desired waypoint into a low-level action (e.g. a linear controller, see the right side of Fig. 1). In practice, Dense Net is recurrent since both the mode and action are highly history-dependent. Sparse Net in contrast only uses the current observation, since waypoints are less multi-modal and history dependent than actions. Then at test time, HYDRA predicts the mode m_t and follows the controller T until reaching the waypoint during predicted sparse periods, and follows low-level actions at each step during predicted dense periods. See Appendix C for more details.

4.1 Data Processing: Mode Labeling and Action Consistency

To dynamically switch action abstractions, we need labeled modes m_t, waypoints w_t, and actions a_t at each time step. We first obtain binary mode labels m_t from humans, and then use the mode labels to extract waypoints and to relabel low-level actions. Importantly, modes can be labeled either online (during demonstration collection, e.g. with a simple button interface), or entirely offline (after demonstration collection, e.g., labeling each frame with its mode). With modes labeled, we can segment each demonstration into sparse waypoint and dense action phases. We provide the details of the labeling and segmentation process in Appendix B. For each sparse phase, we can extract the desired future waypoint w_t at o_t: if $m_t = 0$ (sparse), the future waypoint is final proprioceptive state $w_t = p_{t'}$ in that sparse segment, where $t' > t$. But if $m_t = 1$ (dense), the waypoint is the next proprioceptive state $w_t = p_{t+1}$. This yields a dataset of $\hat{\mathcal{D}}$ of (o, a, w, m) tuples. Now the policy has full supervision to learn the modes, waypoints, and actions.

Mode Labeling Strategy: Since waypoints will be reached online with controller T, the main requirement for labeling modes is that during sparse phases ($m_t = 0$), the labeled waypoint w_t should be reachable via T starting from o_t (i.e., without collision): for example, if the demonstrator starts in free space and labels a waypoint close to coffee K-pod, and if the policy uses a linear P-controller as T, then the K-pod waypoint should be reachable from the initial pose in a straight-line path. Otherwise, the learned policy might collide when it tries to reach similar waypoints. We do not assume access to a collision-avoidance planner as T in this work, but if one has access to a planner then T can always reach the desired waypoint, so this reachability requirement can be ignored. Other considerations for mode labeling and a discussion of mode sensitivity is provided in Appendix B. We specifically show that our method is not overly sensitive to mode labeling strategies outside of the collision-free requirement above. Furthermore, we show that mode labels can be learned from substantially fewer examples without a major effect on performance Appendix D.3.

Relabeling Low-Level Actions: As discussed in Section 3, action consistency can improve policy performance by simplifying the BC objective in Eq. (1) and thus reducing $||\epsilon_t||$, provided the data has enough state coverage. However, making actions consistent during data collection is challenging and can often reduce state coverage [22], so instead HYDRA performs *offline* action relabeling, i.e., after collection. To relabel human actions a_t during the sparse periods, HYDRA uses waypoint controller T to "imagine" a new action at each demonstrated robot state s_t^p based on the waypoint w_t. We lack a consistent relabeling strategy for dense periods, so we leave this to future work.

However, a subtle challenge with offline relabeling is that changing the actions in the data can put the policy out of distribution at test time, since new actions can lead to new states online. For example, if an arc path was demonstrated to get to a waypoint, but a linear controller is used for relabeling, the linear action will take us off that path. HYDRA avoids this problem by using a waypoint controller T online during sparse periods, meaning relabeled actions will not be deployed online. Rather, this action relabeling serves primarily to simplify the dense action learning objective of HYDRA and increase action consistency in the overall dataset.

A natural question arises: since sparse actions will be executed with T online, could we instead further simplify learning by avoiding training on dense actions during sparse periods? If HYDRA mispredicts a sparse mode as dense, then the dense actions will still be executed online, so HYDRA should still be trained on dense actions during sparse periods as a back-up. We show that reducing the training weight of dense actions during sparse periods hurts performance in Appendix D.5.

4.2 Training and Evaluation

Training: HYDRA is trained to both imitate low-level actions a with policy π_θ^A, high-level waypoints w with π_θ^W, and the mode m with π_θ^M at each time step. To balance the waypoint and action losses, we use a mode-specific loss at each time step that weighs the current mode's loss with $(1-\gamma)$, and the other mode's loss with γ. Given a processed dataset $\hat{\mathcal{D}}$ consisting of tuples of (o, a, w, m), we modify the loss in Eq. (1) with the new heads of HYDRA (mode, action, and waypoint):

$$\mathcal{L}_a(\theta) = -\mathbb{E}_{(o,a,w,m)\sim p_{\hat{\mathcal{D}}}} \left[(1-\alpha_m)\log \pi_\theta^A(a|o) + \alpha_m \log \pi_\theta^W(w|o)\right] \tag{2}$$

$$\mathcal{L}_m(\theta) = -\mathbb{E}_{(o,a,w,m)\sim p_{\hat{\mathcal{D}}}} \left[m \log \pi_\theta^M(m=1|o) + (1-m)\log \pi_\theta^M(m=0|o)\right] \tag{3}$$

$\mathcal{L}_a$ weighs the BC loss for waypoints and actions by the current mode: $\alpha_m = m\gamma + (1-m)(1-\gamma)$ is the mode-specific weight for the sparse waypoint part of $\mathcal{L}_a$. If we are in sparse mode ($m = 0$), then $\alpha_m = 1-\gamma$, but in dense mode, $\alpha_m = \gamma$. Thus, a low gamma encourages the model to fit the loss for the current mode *more* than the loss for the other mode, and $\gamma = 0.5$ will be a mode-agnostic weighting. See Appendix D.5 for results of ablating γ. $\mathcal{L}_m$ is the mode cross entropy classification loss. Combining these terms with mode loss weight β, we get the full HYDRA objective:

$$\mathcal{L}(\theta) = \mathcal{L}_a(\theta) + \beta\mathcal{L}_m(\theta) \tag{4}$$

Evaluation: During evaluation, the policy chooses the mode using $\tilde{m}_t$. If $\tilde{m}_t = 0$, the model will servo in a closed-loop fashion to the predicted waypoint $\tilde{w}_t$ using controller T. The policy is queried at every step to continually update the policy hidden state, but importantly its outputs are ignored until we reach the waypoint to avoid action prediction errors. If $\tilde{m}_t = 1$, the model will execute just the next step using the predicted dense action $\tilde{a}_t$.

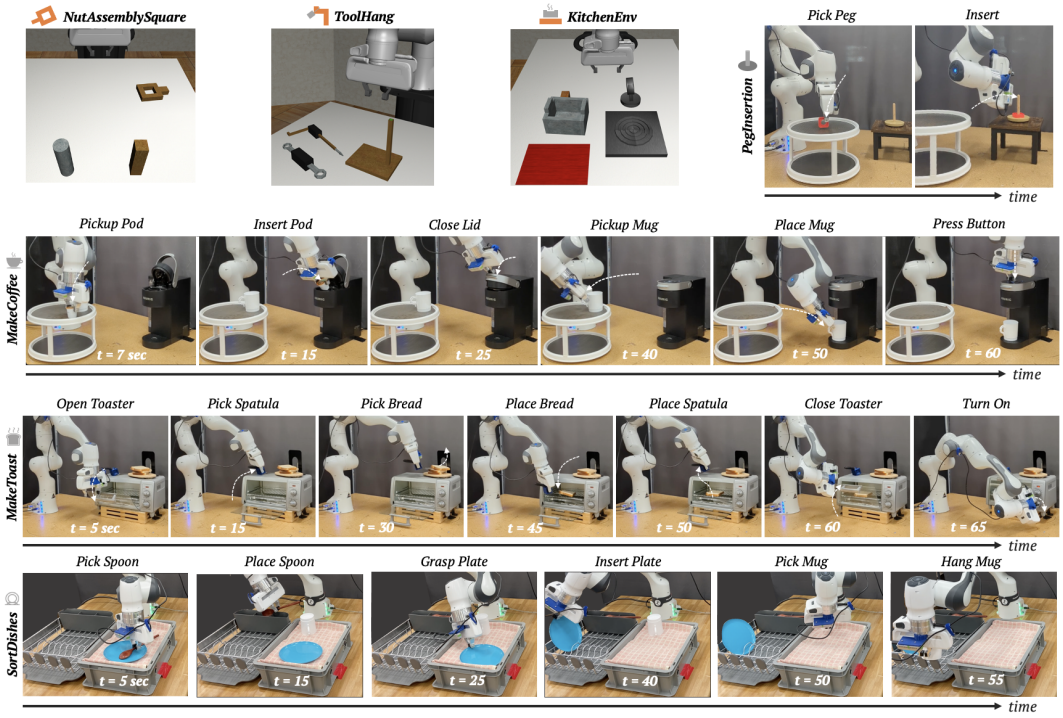

Figure 2: Simulation & Real-world environments, with task stages shown for real world tasks. **Simulation**: In *NutAssemblySquare*, the task is to pick up a square nut at various positions and orientations and insert it onto a vertical square peg. In *ToolHang*, a hanging frame is inserted onto a fixed stand, followed by placing a tool on the frame. Both the frame and tool poses are randomized. Frame insertion is challenging due to the small insertion area. *KitchenEnv* involves turning on a stove, moving a pot onto the stove, putting an object in the pot, then moving the pot to a serving area. **Real World**: *PegInsertion* involves inserting a peg with a hole in the center onto a round insertion rod (top right); the peg location and geometry are varied. *MakeCoffee* is a 6-step task (top middle row) involving picking up a K-pod, inserting it into a Keurig machine, closing the lid of the Keurig, positioning a mug, and then pressing start on the Keurig; the K-pod location and mug orientations are varied. This is a more challenging version of the task used in prior work [13], which did not include the mug component. *MakeToast* has 7-steps (bottom middle row): a hinged toaster oven is opened, a spatula is picked up, bread is placed inside the toaster, the toaster is closed, and the timer dial is turned to start. Both bread and spatula initial poses vary. *SortDishes* (bottom row) has 6 stages: pick up spoon, place spoon in rack, grasp plate and insert it into rack, and grasp mug and hang the mug. All objects vary in initial pose.

5 Experiments

We evaluate the performance of HYDRA in 3 challenging simulation environments and 4 complex real world tasks, shown in Fig. 2. These tasks cover a wide range of affordances and levels of precision, from precisely inserting a coffee pod to picking up bread with a spatula. See Appendix C for model hyperparameters, data collection, and training details. Videos can be found on our website.

Data Collection: We leverage proficient human demonstration data for simulated tasks from robomimic [19]. Mode labels and waypoints were annotated offline for simulation datasets as described in Appendix B. Demonstrations for real world tasks were collected by a proficient user using VR teleoperation using an Oculus Quest 2. Mode labels and waypoints were provided during data collection (online) using the side button on the Quest VR controller with no added collection time.

Simulation: In Fig. 3 (top row), we compare our method to BC and BC-RNN for the *NutAssemblySquare* and *ToolHang* tasks (state-based), as well as the *KitchenEnv* task (vision-based) from robosuite (see top row in Fig. 2). Our method improves performance on the *NutAssemblySquare* task, where baselines are already quite strong. We also ablate the data size from 200 demos to 100 and 50 in Fig. 3, illustrating that HYDRA is more sample efficient than baselines, with the gap growing as data size decreases. HYDRA-NR in Fig. 3 removes action relabeling and drops performance by 8%, which we attribute to high action multi-modality in non-relabeled sparse periods.

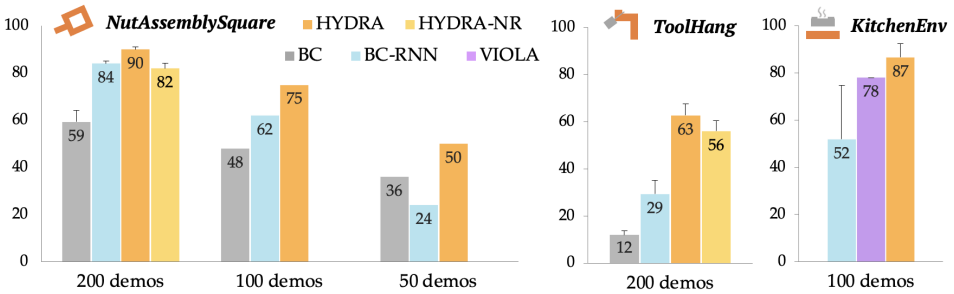

Figure 3: Sim Results for HYDRA vs. BC, BC-RNN, and VIOLA: best checkpoint success rate averaged over three seeds. **Left to Right**: *NutAssemblySquare* (state), *ToolHang* (state), and *KitchenEnv* (vision) tasks. HYDRA beats baselines on all of these tasks, and even beats VIOLA [13] on the kitchen task despite using a much smaller and simpler model. We also show a comparison for BC-RNN and HYDRA with decreasing data sizes for *NutAssemblySquare*, showing that our method is more sample efficient than BC-RNN. HYDRA without action relabeling (HYDRA-NR, *NutAssemblySquare* and *ToolHang*) drops performance by 7-8%.

For the *Tool Hang* task (top middle in Fig. 3), which is long horizon and consists of many waypoint and dense periods and requiring much higher precision, our method has an even bigger gap in performance with BC and BC-RNN. While the best baseline gets 29%, our method reaches 63% with the same inputs. Once again, removing action relabeling (HYDRA-NR) drops performance by 7% but is still substantially better than baseline.

For *KitchenEnv* (vision-based), we also compare to VIOLA [13], an image-based model that uses bounding box features and a large transformer architecture to predict actions. Once again, HYDRA is able to outperform BC-RNN by 35% on this long horizon task. HYDRA also outperforms VIOLA by 9%, despite using a simpler and smaller model.

In Appendix D, we show a waypoint-only baseline, mode labeling strategy ablations, and a relabeling-only ablation where action consistency is improved but the waypoint controller is not used online. In Appendix D.3, we show that mode labels can be learned with fewer examples without a large drop in performance (e.g., using 25% of mode labels drops performance by 10%).

Real World: In Fig. 4, we compare our method to BC-RNN (vision-based) for four high precision tasks: *PegInsertion*, *MakeCoffee*, *MakeToast*, and *SortDishes*. The latter three are long-horizon, and Fig. 4 shows cumulative success per task stage. In *PegInsertion*, our method substantially outperforms BC-RNN at both peg grasping and precise insertion portions of the task, thanks to combining precise waypoints with flexible low level actions where necessary.

For *MakeCoffee*, HYDRA once again beats BC-RNN and VIOLA by a substantial margin at all stages of the task. Although all methods perform well in grasping the K-pod, the performance of the baselines declines rapidly in the following phases. While BC-RNN failed to do this task in prior work, we see that with a bit of parameter tuning, BC-RNN is a strong baseline, achieving 20% performance [13]. The reported performance of VIOLA in prior work for coffee pod insertion and closing the lid is 60%, which matches with the performance we observe for the corresponding stage of our coffee task. Our task adds two more stages (picking up and placing a mug before pressing the button), interestingly causing the final success rate of VIOLA to drop to 20%, the same as BC-RNN. Using the same parameters and model size as BC-RNN, HYDRA achieves 80% final success at this task with the same underlying dataset.

For *MakeToast* and *SortDishes*, HYDRA performs better on all stages of the task as compared to BC-RNN. We omit VIOLA in these tasks since, as seen in the coffee task, BC-RNN is a competitive baseline. Both tasks consists of several bottleneck stages where performance drops sharply. In *MakeToast*, for picking up bread, the spatula must slide underneath a bread slice – HYDRA passes this stage 70% of the time, beating BC-RNN by 30%. The last stage (turning the toaster on) is particularly challenging for all methods, but HYDRA completes it 20% of the time compared to 0% for BC-RNN. In *SortDishes*, the final hang-mug stage similarly requires high precision with randomized objects. Not including the challenging last stage, HYDRA beats BC-RNN by 40% on this task. See Appendix D.1 for rollouts of each task for each model.

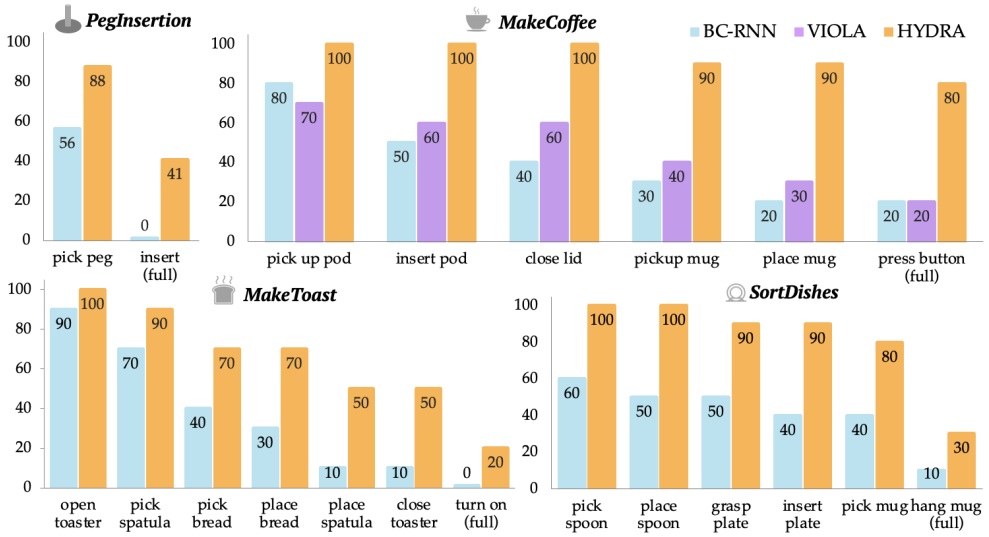

Figure 4: Real Results for HYDRA vs. BC, BC-RNN, and VIOLA. The x-axis denotes each stage (right-most value is the final success rate). **Top Left**: HYDRA vs. BC-RNN on the real *PegInsertion* task for 50 demos under 32 rollouts across 4 different nuts. This task requires very precise grasping and insertion of multiple types of nuts, which our method does with high success. While baseline is unable to perform insertion, HYDRA gets 41% success. **Top Right**: *MakeCoffee* long-horizon task for 100 demos under 10 rollouts. Our method beats baseline by 60%. **Bottom Left**: *MakeToast* long-horizon task for 100 demos under 10 rollouts. While both methods struggle to turn the toaster on, HYDRA is able to reach 50% success for 6/7 stages compared to 10% for baseline. **Bottom Right**: *SortDishes* for 100 demos under 10 rollouts. Waypoints in HYDRA precisely capture the diverse poses in this task, beating BC-RNN by 40% and 20% for the last two stages.

We observe that the performance gain for HYDRA in our real world experiments is notably higher than in simulation. We hypothesize this is due to (1) higher variance in action playback on the real robot setup, which HYDRA mitigates during sparse periods using the closed-loop waypoint controller, and (2) increased potential for compounding errors in longer tasks. Overall, HYDRA is well-suited to long horizon tasks even with many high-precision bottleneck stages, due to its ability to switch between waypoints and dense actions and its ability to increase action consistency offline. We also observed that in our real world tasks, HYDRA exhibits emergent retrying behavior, often re-servoing to a consistent and in-distribution waypoint to retry a failed dense period.

6 Discussion

Summary: In this work, we propose HYDRA, which uses a flexible action abstraction to reduce compounding errors, and improves action consistency while maintaining the state diversity present in uncurated human demonstrations. HYDRA learns to dynamically switch between following waypoints and taking low level actions with a small amount of added mode label supervision that can be provided either online or offline. HYDRA substantially outperforms baselines on three simulation tasks and four real world tasks that involve long horizon manipulation with many bottleneck states.

Limitations & Future Work: While only a minor amount of added supervision, HYDRA relies on having expert-collected mode labels. We show that mode labels can be learned from much less data in Appendix D.3, but future work might consider using unsupervised methods for mode labeling, e.g., skill segmentation [37] or automatically extracting "linear" portions of a demonstration. We also hypothesize multi-task datasets can help learn a general mode-predictor that can be fine-tuned or deployed zero-shot on novel tasks. Furthermore, when mode labels are collected online, mode labeling can add a mental load for the demonstrator and might also influence the quality of the data on its own. Future work might conduct more extensive user studies to better understand the effect of providing mode labels for both the demonstrator and the final learning performance.

Despite these limitations, HYDRA is a simple and easy-to-implement method, and it is exciting that it shows substantial improvement over state-of-the-art imitation learning techniques and significant promise in solving challenging manipulation tasks in the real world.

References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

[2] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[3] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[5] R. Thoppilan, D. De Freitas, J. Hall, N. Shazeer, A. Kulshreshtha, H.-T. Cheng, A. Jin, T. Bos, L. Baker, Y. Du, et al. Lamda: Language models for dialog applications. *arXiv preprint arXiv:2201.08239*, 2022.

[6] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.

[7] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[8] S. Ross and D. Bagnell. Efficient reductions for imitation learning. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 661–668. JMLR Workshop and Conference Proceedings, 2010.

[9] S. Ross, G. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011.

[10] J. Spencer, S. Choudhury, A. Venkatraman, B. Ziebart, and J. A. Bagnell. Feedback in imitation learning: The three regimes of covariate shift. *arXiv preprint arXiv:2102.02872*, 2021.

[11] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. Palm-e: An embodied multimodal language model. In *arXiv preprint arXiv:2303.03378*, 2023.

[12] S. Karamcheti, S. Nair, A. S. Chen, T. Kollar, C. Finn, D. Sadigh, and P. Liang. Language-driven representation learning for robotics. 2023.

[13] Y. Zhu, A. Joshi, P. Stone, and Y. Zhu. Viola: Imitation learning for vision-based manipulation with object proposal priors. *6th Annual Conference on Robot Learning (CoRL)*, 2022.

[14] S. Belkhale and D. Sadigh. Plato: Predicting latent affordances through object-centric play. In *6th Annual Conference on Robot Learning*, 2022.

[15] P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[16] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *arXiv preprint arXiv:2303.04137*, 2023.

[17] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *6th Annual Conference on Robot Learning*, 2022.

[18] R. P. Joshi, N. Koganti, and T. Shibata. A framework for robotic clothing assistance by imitation learning. *Advanced Robotics*, 33(22):1156–1174, 2019.

[19] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. *arXiv preprint arXiv:2108.03298*, 2021.

[20] P. Sundaresan, S. Belkhale, and D. Sadigh. Learning visuo-haptic skewering strategies for robot-assisted feeding. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=lLq09gVoaTE.

[21] J. Silvério, Y. Huang, F. J. Abu-Dakka, L. Rozo, and D. G. Caldwell. Uncertainty-aware imitation learning using kernelized movement primitives. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 90–97. IEEE, 2019.

[22] K. Gandhi, S. Karamcheti, M. Liao, and D. Sadigh. Eliciting compatible demonstrations for multi-human imitation learning. In *Proceedings of the 6th Conference on Robot Learning (CoRL)*, 2022.

[23] S. Belkhale, Y. Cui, and D. Sadigh. Data quality in imitation learning, 2023.

[24] L. L. Richmond and J. M. Zacks. Constructing experience: Event models from perception to action. *Trends in Cognitive Sciences*, 21(12):962–980, 2017. ISSN 1364-6613. doi:https://doi.org/10.1016/j.tics.2017.08.005. URL https://www.sciencedirect.com/science/article/pii/S1364661317301857.

[25] C. Baldassano, J. Chen, A. Zadbood, J. Pillow, U. Hasson, and K. Norman. Discovering event structure in continuous narrative perception and memory, 2016.

[26] Y. Cui, D. Isele, S. Niekum, and K. Fujimura. Uncertainty-aware data aggregation for deep imitation learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 761–767. IEEE, 2019.

[27] R. Hoque, A. Balakrishna, E. Novoseller, A. Wilcox, D. S. Brown, and K. Goldberg. Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning. In *Conference on Robot Learning*, pages 598–608. PMLR, 2022.

[28] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

[29] A. Mandlekar, D. Xu, R. Martín-Martín, Y. Zhu, L. Fei-Fei, and S. Savarese. Human-in-the-loop imitation learning using remote teleoperation. *arXiv preprint arXiv:2012.06733*, 2020.

[30] M. L. Schrum, E. Hedlund-Botti, N. Moorman, and M. C. Gombolay. Mind meld: Personalized meta-learning for robot-centric imitation learning. In *2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 157–165. IEEE, 2022.

[31] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg. Dart: Noise injection for robust imitation learning. In *Conference on robot learning*, pages 143–156. PMLR, 2017.

[32] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta. R3m: A universal visual representation for robot manipulation. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=tGbpgz6yOrI.

[33] K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, 2020.

[34] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on robot learning*, pages 1113–1132. PMLR, 2020.

[35] M. Shridhar, L. Manuelli, and D. Fox. Perceiver-actor: A multi-task transformer for robotic manipulation. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=PS_eCS_WCvD.

[36] E. Johns. Coarse-to-fine imitation learning: Robot manipulation from a single demonstration. In *2021 IEEE international conference on robotics and automation (ICRA)*, pages 4613–4619. IEEE, 2021.

[37] T. Kipf, Y. Li, H. Dai, V. Zambaldi, A. Sanchez-Gonzalez, E. Grefenstette, P. Kohli, and P. Battaglia. CompILE: Compositional imitation learning and execution. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3418–3428. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/kipf19a.html.

[38] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

[39] L. Zhang and B. C. Stadie. Understanding hindsight goal relabeling requires rethinking divergence minimization. In *Deep Reinforcement Learning Workshop NeurIPS 2022*.