A GENERATING CORRECTIVE LABELS WITH KNOWN DYNAMICS FUNCTION

With a known dynamics function, we show an example algorithm that one can use to generate N corrective labels (as defined in Sec 4.1). The algorithm first trains a behavior cloning policy, samples test time roll out trajectories from the learned policy, and then derives labels using a root finding solver.

Algorithm 2 Generating Corrective Labels using Dynamics Function

Input Expert Data $\mathcal{D}* = (s_i^*, a_i^*, s_{i+1}^*)$.
Input Dynamics function $f(s^{i+1}|s^i, a^i)$.
Input Parameter N.
Initialize $\mathcal{D}^{\mathcal{G}} \leftarrow \varnothing, \mathcal{S}' \leftarrow \varnothing$
$\hat{\pi} = \arg\min_{\hat{\pi}} -\mathbb{E}_{s_i^*, a_i^*, s_{i+1}^* \sim \mathcal{D}^*} \log(\hat{\pi}(a_i^* \mid s_i^*))$
for $i \in 1..N$ **do**
    $s_0^{\mathcal{G}} \sim P_0, s_{j+1}^{\mathcal{G}} \sim f(s_j^{\mathcal{G}}, \pi(s_j^{\mathcal{G}})), \mathcal{S}' \leftarrow \mathcal{S}' \cup \{s_j^{\mathcal{G}}\}$
end for
for $i \in 1..N$ **do**
    $a_i^{\mathcal{G}} \leftarrow \arg\min_{a^{\mathcal{G}}} ||[s_i^{\mathcal{G}} + f(s_i^{\mathcal{G}}, a_i^{\mathcal{G}})] - s_k^*||$
    $\mathcal{D}^{\mathcal{G}} \leftarrow \mathcal{D}^{\mathcal{G}} \cup (s_i^{\mathcal{G}}, a_i^{\mathcal{G}})$
end for
return $\mathcal{D}^{\mathcal{G}}$

B PROOFS

B.1 PROOF OF THEOREM 4.2

Notation. Let f be the ground truth 1-step residual dynamics model, and let $\hat{f}$ be the learned approximation of f.

Assumptions:

1. The estimation error of the learned dynamics model *at the training data* is bounded. $\left\| f(s_{t+1}^*, a_{t+1}^*) - \hat{f}(s_{t+1}^*, a_{t+1}^*) \right\| \leq \epsilon$.
2. $\hat{f}$ is locally K_1-Lipschitz in state around data points: $\left\| \hat{f}(s_t^{\mathcal{G}}, a_{t+1}^*) - \hat{f}(s_{t+1}^*, a_{t+1}^*) \right\| \leq K_1 \left\| s_t^{\mathcal{G}} - s_{t+1}^* \right\|$.
3. f is locally K_2-Lipschitz in state around data points: $\left\| f(s_t^{\mathcal{G}}, a_{t+1}^*) - f(s_{t+1}^*, a_{t+1}^*) \right\| \leq K_2 \left\| s_t^{\mathcal{G}} - s_{t+1}^* \right\|$.

Proof:

$$
\begin{aligned}
&\left\| f(s_t^{\mathcal{G}}, a_{t+1}^*) - \hat{f}(s_t^{\mathcal{G}}, a_{t+1}^*) \right\| \\
&= \left\| f(s_t^{\mathcal{G}}, a_{t+1}^*) - f(s_{t+1}^*, a_{t+1}^*) + f(s_{t+1}^*, a_{t+1}^*) - \hat{f}(s_{t+1}^*, a_{t+1}^*) + \hat{f}(s_{t+1}^*, a_{t+1}^*) - \hat{f}(s_t^{\mathcal{G}}, a_{t+1}^*) \right\| \\
&\leq \left\| f(s_t^{\mathcal{G}}, a_{t+1}^*) - f(s_{t+1}^*, a_{t+1}^*) \right\| + \left\| f(s_{t+1}^*, a_{t+1}^*) - \hat{f}(s_{t+1}^*, a_{t+1}^*) \right\| + \left\| \hat{f}(s_{t+1}^*, a_{t+1}^*) - \hat{f}(s_t^{\mathcal{G}}, a_{t+1}^*) \right\| \\
&\leq \epsilon + (K_1 + K_2) \left\| s_t^{\mathcal{G}} - s_{t+1}^* \right\|
\end{aligned}
$$

Remark Our assumption about the error of the learned dynamics model is not a global constraint but simply requires the model to have prediction small error *on the training data*. Our proof leverages simple triangle inequality and is valid only near the expert data support.

B.2 PROOF OF THEOREM 4.3

Notation. Let f be the ground truth 1-step residual dynamics model, and let $\hat{f}$ be the learned approximation of f.

Assumptions

1. f is locally K_1-Lipschitz in *action*: $\left\|f(s_t^{\mathcal{G}}, a_t^*) - f(s_t^{\mathcal{G}}, a_t^* + \Delta)\right\| \leq K_1 \left\|\Delta\right\|$.
2. f is locally K_2-Lipschitz in *state*: $\left\|f(s_t^{\mathcal{G}}, a_t^*) - f(s_t^*, a_t^*)\right\| \leq K_2 \left\|s_t^{\mathcal{G}} - s_t^*\right\|$.
3. Using rejection sampling, we can enforce $\left\|s_t^{\mathcal{G}} - s_t^*\right\| \leq \epsilon_{rej}$.
4. Given that the generated labels come from a root solver:

$$\begin{aligned} s_{t+1}^* - \hat{f}(s_t^{\mathcal{G}}, a_t^* + \Delta) - s_t^{\mathcal{G}} &= \epsilon_{opt} \text{ where } \epsilon_{opt} \to 0 \\ s_t^* + f(s_t^*, a_t^*) - \hat{f}(s_t^{\mathcal{G}}, a_t^* + \Delta) - s_t^{\mathcal{G}} &= \epsilon_{opt} \\ f(s_t^*, a_t^*) - \hat{f}(s_t^{\mathcal{G}}, a_t^* + \Delta) &= s_t^{\mathcal{G}} - s_t^* + \epsilon_{opt} \end{aligned}$$

Proof

$$\begin{aligned} &\left\|f(s_t^{\mathcal{G}}, a_t^* + \Delta) - \hat{f}(s_t, a_t^* + \Delta)\right\| \\ &= \left\|f(s_t^{\mathcal{G}}, a_t^* + \Delta) - f(s_t^{\mathcal{G}}, a_t^*) + f(s_t^{\mathcal{G}}, a_t^*) - f(s_t^*, a_t^*) + f(s_t^*, a_t^*) - \hat{f}(s_t, a_t^* + \Delta)\right\| \\ &\leq \left\|f(s_t^{\mathcal{G}}, a_t^* + \Delta) - f(s_t^{\mathcal{G}}, a_t^*)\right\| + \left\|f(s_t^{\mathcal{G}}, a_t^*) - f(s_t^*, a_t^*)\right\| + \left\|f(s_t^*, a_t^*) - \hat{f}(s_t, a_t^* + \Delta)\right\| \\ &\leq K_1 \left\|\Delta\right\| + K_2 \left\|s_t^{\mathcal{G}} - s_t^*\right\| + \left\|s_t^{\mathcal{G}} - s_t^*\right\| + ||\epsilon_{opt}|| \\ &\leq K_1 \left\|\Delta\right\| + (1 + K_2) \cdot \epsilon_{rej} + ||\epsilon_{opt}|| \end{aligned}$$

When the root solver yields a solution with $||\epsilon_{opt}|| = 0$, we have $\leq K_1 \left\|\Delta\right\| + (1 + K_2) \cdot \epsilon_{rej}$

C Details for CCIL

Our proposed framework for generating corrective labels, **CCIL**, takes three steps:

1. **Learn a dynamics model**: fit a dynamics model $\hat{f}$ that is locally Lipschitz continuous.
2. **Generate labels**: solve a root-finding equation in Sec. 4.3 to generate labels.
3. **Augment the dataset and train a policy**: We use behavior cloning for simplicity to train a policy.

C.1 Learning a locally Lipschitz continuous dynamics model

There are many function approximators to learn a model. For example, using Gaussian process can produce smooth dynamics model but might have limited scalability when dealing with large amount of data. In this paper we demonstrate examples of using a neural network to learn the dynamics model. There are multiple ways to enforce Lipschitz continuity on the learned dynamics function, with varying levels of strength that trade off theoretical guarantees and learning ability. In Sec. 4.2 we discusses using penalty or slack variables to enforce local Lipschitz continuity, here we provide an alternative method used by Shi et al. (2019).

Global Lipschitz Continuity via Spectral Normalization. Using spectral norm with coefficient L provides the strongest guarantee that the dynamic model is globally L-Lipschitz. Concretely, spectral normalizat Miyato et al. (2018) normalizes the weights of the neural network following each gradient update.

$$\arg\min_{\hat{f}} E_{s_j^*, a_j^*, s_{j+1}^* \sim \mathcal{D}^*} \left[\text{MSE} \quad \text{while } W \to W / \max(\frac{||W||_2}{\lambda}, 1)\right] \tag{8}$$

However, spectral normalization enforces *global* Lipschitz bound. It may hinder the model's ability to learn the true dynamics.

Local Lipschitz Continuity via Sampling-based Penalty. Following Gulrajani et al. (2017), a simple way to relax the global Lipschitz continuity constraint is by penalizing any violation of local Lipschitz constraint

$$\arg\min_{\hat{f}} E_{s_j^*, a_j^*, s_{j+1}^* \sim \mathcal{D}^*} \left[\text{MSE} + \lambda \cdot \mathbb{E}_{\Delta_s \sim \mathcal{N}} \max\left(\hat{f}'(s_j^* + \Delta_s, a_j^*) - L, 0\right) \right]. \tag{9}$$

Doing so ensures that the approximate model is mostly L Lipschitz-bounded while being predictive of the transitions in the expert data. This approximate dynamics model can then be used to generate corrective labels. The sampling procedure, $\mathbb{E}_{\Delta_s \sim \mathcal{N}} \max\left(\hat{f}'(s_j^* + \Delta_s, a_j^*) - L, 0\right)$, is indicative of whether the local continuity constraint is violated for a given state-action pair.

The sampling-based penalty perturbs the data points by some sampled noise and enforces the Lipschitz constraint between the perturbed data and the original using a penalty term in the loss function. In Sec. 4.2, we showed an alternative ways to enforce *local* Lipschitz continuity. i,e, The slack-based penalty method.

We here discuss another approach for local continuity: enforce $\|J_f\|_2 \leq L$, where J_f is the Jacobian of f, when evaluated at the data points using a penalty in the loss function, which essentially adds a soft constraint that the model remain L-Lipschitz in the neighborhood of the data distribution.

Intuitively, we expect that spectral normalization tends to work better for simpler environments where the ground truth dynamics are global Lipschitz with some reasonable L, whereas the soft constraint should be better suited for data regimes with more complicated dynamics and discontinuity.

C.2 Generating corrective labels

In Sec. 4.3 we discuss two techniques to generate corrective labels. Depending on the structure of the application domain, one can choose to generate labels either by backtrack or by disturbed actions. Both techniques require solving root-finding equations (Eq. 4 and Eq. 6). To solve them, we can transform the objective to an optimization problem and apply gradient descent.

Eq. 4 specifies the root finding problem for backtrack labels.

$$s_t^* - \hat{f}(s_{t-1}^{\mathcal{G}}, a_t^*) - s_{t-1}^{\mathcal{G}} = 0.$$

Given s_t^*, a_t^* and the learned dynamics function $\hat{f}$, we need to solve for $s_t^{\mathcal{G}}$ that satisfies the equation. We can instead optimize for

$$\arg\min_{s_{t-1}^{\mathcal{G}}} ||s_t^* - \hat{f}(s_{t-1}^{\mathcal{G}}, a_t^*) - s_{t-1}^{\mathcal{G}}|| \tag{10}$$

Similarly, we can transform Eq. 6 to become

$$\arg\min_{s_t^{\mathcal{G}}} ||s_t^{\mathcal{G}} + \hat{f}(s_t^{\mathcal{G}}, a_t^* + \Delta) - s_{t+1}^*|| \tag{11}$$

With access to the trained model $\hat{f}$ and its gradient $\frac{\partial \hat{f}}{\partial s_t^{\mathcal{G}}}$, one can use any optimizer. For simplicity, we use the Backward Euler solver that apply an iterative update $s_t^{\mathcal{G}} \leftarrow s_t^{\mathcal{G}} - s \cdot \frac{\partial \hat{f}}{\partial s_t^{\mathcal{G}}}$ where s is a step size. We repeat the update until the objective is within a threshold $||s_t^{\mathcal{G}} + \hat{f}(s_t^{\mathcal{G}}, a_t^* + \Delta) - s_{t+1}^*|| \leq \epsilon_{opt}$.

C.3 Using the generated labels

There are multiple ways to use the generated corrective labels. We can augment the dataset with the generated labels and treat them as if they are expert demonstrations. For example, for all experiments conducted in this paper, we train behavior cloning agent using the augmented dataset. Optionally, one can favor the original expert demonstrations by assigning higher weights to their training loss. We omit this step for simplicity in this paper.

Alternatively, one can query a trained imitation learning policy with the generated labels and measure the difference between our generated action and the policy output. This difference can be used as an alternative metrics to evaluate the robustness of imitation learning agents when encountered a subset of out-of-distribution states. However, for a query state, our proposal does not necessarily recover *all* possible corrective actions. We defer exploring alternatives way of leveraging the generated labels to future work.

D EXPERIMENTAL DETAILS

We provide details to reproduce our experiments, including environment specification, expert data, parameter tuning for our proposal and details about the baselines. We will also open source the code and the configuration we use for each experiment, once the proposal is published.

D.1 ENVIRONMENT AND TASK DESIGN

We conduct experiments on 4 different domains and 8 robots, including the pendulum, a drone, a car, four robots for locomotion and one robot arm for manipulation. We consider 18 tasks: the pendulum, a modified pendulum swing task with discontinuity, three drone navigation tasks (fly-through, circle, hover), one LiDar racing task on F1tenth, four MuJoCo tasks (Hopper, HalfCheetah, Ant, Walker2D) and 8 MetaWorld tasks (coffee-pull, coffee-push, button-press-topdown, drawer-close, drawer-open, window-close, push, soccer). The drone, F1tenth, MuJoCo and Metaworld environments are from open source implementations. We will describe how we set up the Pendulum environment and how we modify it for testing our method with discontinuity.

Pendulum Formulation. The pendulum environment asks a policy to swing a pendulum up to the vertical position by applying torque. The properties of the system are controlled by the constants g, the gravitational acceleration, and l, the length of the pendulum. In all experiments we take $g = 9.81$ and $l = 1$.

A pendulum's state is characterized by θ, the current angle, and $\dot{\theta}$, the current angular velocity. To avoid any issues regarding angle representation, we do not directly store θ in the state representation; instead, we parameterize the state as $s = \begin{bmatrix}\sin\theta & \cos\theta & \dot{\theta}\end{bmatrix}^T$. A policy can control the system by applying torque to the pendulum, which we represent as a scalar a, which is clamped to the range $[-3, 3]$.

The continuous time dynamics function is given by:

$$\frac{ds}{dt} = f(s, a) = \begin{bmatrix} \dot{\theta}\cos\theta \\ -\dot{\theta}\sin\theta \\ -\frac{g}{l}\sin\theta + a \end{bmatrix}.$$

This continuous dynamics model is then discretized to a timestep of 0.02 seconds using RK4. Additionally, although not required by the algorithms we study, we create the following reward function. where θ is the normalized pendulum angle in the range $[0, 2\pi)$ to metricize policy performance:

$$r(s, a) = -\frac{1}{2}\left\|\begin{bmatrix}\theta - \pi \\ \dot{\theta}\end{bmatrix}\right\|_2^2 - \frac{1}{2}a^2.$$

Expert Formulation for Pendulum. We formulate the expert policy using a combination of LQR and energy shaping control, where LQR is applied when the pendulum is near the top and energy-shaping is applied everywhere else. Note that the LQR gains were calculated by linearizing the dynamics around $\theta = \pi$, along with the cost function $c(s, a) = -r(s, a)$. So, the expert policy has the form:

$$\pi_e(s) = \begin{cases} -20.11(\theta - \pi) - 7.08\dot{\theta} & \text{if } |\theta - \pi| < 0.1 \\ -\dot{\theta}\left(\frac{1}{2}\dot{\theta}^2 - 9.81\cos\theta - 9.81\right) & \text{otherwise.} \end{cases}$$

Discontinuous Pendulum . We create a fixed wall in the Pendulum environment to create local discontinuity. When the ball hits the wall, we *revert* the sign of its velocity, creating a discontinuity in the dynamics.

D.2 DEMONSTRATION DATA

To feed expert data to train imitation learning agents, we design expert policies for the pendulum. For all other environments, we use the expert data from the D4RL dataset Fu et al. (2020). For drone

Environment	Trajectories
Pendulum	50
Discontinuous Pendulum	500
F1tenth Racing	1
Drone, all three tasks	5000
MuJoCo - Ant	10
MuJoCo - Walker2D	20
MuJoCo - Hopper	25
MuJoCo - HalfCheetah	50
MetaWorld, all tasks	50

Table 3: Number of expert demonstration trajectories used in our experiments. We limit the amount of expert data to avoid making the task trivially solvable by naive behavior cloning.

environments, we first generate a bunch of via points alongside the target trajectories and then use a low-level PID controller to hit the via points one by one.

We note that it is possible to solve most tasks with naive behavior cloning if we feed them with a sufficient number of demonstrations. We thus limit the number of demonstrations we use for all tasks, as shown in Table. 3.

D.3 Parameter Tuning.

Our proposal first trains a dynamics model and has hyperparameters: $\bar{L}$ (desired local Lipschitz smoothness per NN layer), λ (weight of the Lipschitz penalty) and σ (size of perturbation for estimating local Lipschitz continuity). We first fit a dynamics model without enforcing any Lipschitz smoothness, to obtain an average prediction error for reference. To enforce local Lipschitz continuity, we then adopt the sampling-based penalty and train a series of dynamics models by sweeping parameters, $\bar{L} = [2, 3, 5, 10]$, soft dynamics $\lambda = [0.3, 0.5]$ and $\sigma = [0.0001, 0.0003, 0.0005]$. In environments with many discontinuities (MuJoCo and MetaWorld tasks), we use the slack variable to train the dynamics model. Following Eq. 3, we introduce an additional learnable variable, the slack variable. We choose a relatively smaller β (in our case we choose 0.1), to avoid the explosion on gradients of zero norm estimation. The slack variable is a state-conditioned network that could be easily modeled as a two-layer MLPs(64,32). As to the baseline number $\bar{\lambda}$, we choose it to be 0.1.

To choose a dynamics model for generating labels, we will follow Theorem. 4.2 and Theorem. 4.3. We note that the local Lipschitz bound of a neural network is the product of the Lipschitz bound of each layer. Given that we are using two-layer NN to train our dynamics model, $L = \bar{L}^2$. We denote the empirical prediction error on the validation set for each trained dynamics model as ϵ. We choose the best dynamics model that has the smallest error bound. For Theorem. 4.2, we optimize for $\epsilon + 2L \cdot ||C||$ where C is a constant that we pick to be the average $s^*_{t+1} - s^*_t$ across the training data. For Theorem. 4.3, we optimize for $0.001 \cdot L + (1 + L)\epsilon$.

To generate corrective labels, we pick the dynamics model, the size of the perturbation (Sigma) and the rejection threshold (Epsilon). Empirically, Sigma = 0.00001 and Epsilon = 0.01 worked for all our environments. It is also possible to fine-tune Epsilon, the rejection threshold, for each task and each trained dynamics model to optimize the error bound. In our experiments, we omit this step for simplicity.

After generating corrective labels, we use two-layer MLPs (64,64) plus ReLu activation to train a Behavior Cloning agent with both original and augmented data.

D.4 Baselines

We evaluate the following algorithms to gain a thorough understanding of how our proposal compares to relevant baselines.

- EXPERT: For reference we plot the theoretical upper bound of our performance, which is the score achieved by an expert during data collection.
- BC: a naive behavior cloning agent that minimizes the KL divergence on a given dataset.
- NOISEBC: a modification to naive behavior cloning that injects a small disturbance noise to the input state and reuses the action label, as described in Ke et al. (2021b).
- CCIL: our proposal to generate high-quality corrective labels.