A AWE Pseudocode

We provide the complete pseudocode for AWE in Algorithm 1.

Algorithm 1 Automatic Waypoint Extraction (AWE)

```
input: 𝒟; // expert demonstrations
input: ℒ, f, η;
// waypoint selection via dynamic programming
def get_waypoints(τ, η, ℳ):
    if τ ∉ ℳ then
        // check if the endpoints are valid waypoints
        if ℒ(f({τ.start, τ.end}), τ) ≤ η then
            ℳ[τ] = {τ.start, τ.end};
        // try all intermediate states as waypoints, and return the smallest set
        else
            // initialize length of current shortest subsequence
            m ← ∞;
            // loop over all intermediate states as waypoints
            for w ∈ τ.mid do
                𝒲_before ← get_waypoints(τ.before(w), η);
                𝒲_after ← get_waypoints(τ.after(w), η);
                // dedupe w, as it is in both of them
                𝒲 ← (𝒲_before \ {w}) ∪ 𝒲_after;
                if |𝒲| < m then
                    m ← |𝒲|;
                    ℳ[τ] ← 𝒲;
    return ℳ[τ];

// construct dataset for next waypoint prediction
def preprocess_traj(𝒲, τ):
    𝒟_aug ← {};
    for (o_t, x_t) ∈ τ do
        // select the nearest future waypoint in 𝒲
        w ← 𝒲.next_waypoint(t);
        𝒟_aug ← 𝒟_aug ∪ {(o_t, x_t, w)};
    return 𝒟_aug;

𝒟_new ← {};
for τ ∈ 𝒟 do
    ℳ ← {}; // memoize waypoints for efficient dynamic programming
    𝒟_new ← 𝒟_new ∪ preprocess_traj(get_waypoints(τ, η, ℳ), τ)
output: 𝒟_new
```

B Hyperparameters

B.1 Error Budget Threshold

The only hyperparameter we need for waypoint selection is η, the error threshold (Table 4). η is the same for all data sizes $\{30, 50, 100, 200\}$ across all tasks on RoboMimic, i.e. $\eta = 0.005$. We also use a consistent η for both scripted data and human data on both tasks in the Bimanual Manipulation benchmark, i.e. $\eta = 0.01$. Two out of three real-world tasks also use the same η; however, on the Coffee Making task, we opt for a lower η to select more waypoints due to the high-precision nature of the task.

Table 4: Hyperparameter for waypoint selection.

Task	Error thresholod (η)
Lift	0.005
Can	0.005
Square	0.005
Cube Transfer	0.01
Bimanual Insertion	0.01
Screwdriver Handover	0.01
Wiping Table	0.01
Coffee Making	0.008

B.2 ACT in Bimanual Simulation Suite

We use the same hyperparameters as the ACT paper [6], shown in Table 5, except reducing the chunk size from 100 to 50. Intuitively, as the length of trajectories reduces after running AWE, the chunk size can also be reduced to represent the same wall-clock time.

Hyperparameter	ACT	AWE +ACT
learning rate	1e-5	1e-5
batch size	8	8
# encoder layers	4	4
# decoder layers	7	7
feedforward dimension	3200	3200
hidden dimension	512	512
# heads	8	8
chunk size	100	**50**
beta	10	10
dropout	0.1	0.1

Table 5: Hyperparameters of AWE +ACT and ACT. The only difference is reduction in chunk size

B.3 Diffusion Policy in RoboMimic

We use the exact same set of training hyperparameters as Diffusion Policy [5] (Table 6). The only additional hyperparameter we added is the "control multiplier" (bottom row), which allows the low-level controller to take more steps to reach the target position at the inference time. This can be useful when predicted waypoints are far apart.

C Implementation and Experiment Details

C.1 Controller

We use an Operation Space Controller (OSC) in RoboMimic, which allows position and orientation control of the robot's end effector. It takes in the desired absolute position and orientation of the end-effector, and computes the necessary torques and velocities.

We use the default joint position controller in the Bimanual Manipulation benchmark. On real-world tasks, we made no change to the controller except for the Coffee Making task, where we increased DT from 0.02 to 0.1. This allows the controller operate similarly to a blocking controller, which continues to execute low-level actions until reaching the desired joint position.

C.2 Loss Function

To determine the distance between potential waypoints and the ground truth trajectory, we project the ground truth state onto the linearly interpolated waypoint trajectory and compute the L2 distance for xyz position. For orientation, we convert the axis angles to quaternions and slerp two ground truth

H-Param	Lift	Can	Square
Ctrl	Pos	Pos	Pos
To	2	2	2
Ta	8	8	8
Tp	10	10	10
#D-params	9	9	9
#V-params	22	22	22
#Layers	8	8	8
Emb Dim	256	256	256
Attn Dropout	0.3	0.3	0.3
Lr	1e-4	1e-4	1e-4
WDecay	1e-3	1e-3	1e-3
D-Iters Train	100	100	100
D-Iters Eval	100	100	100
Control Multiplier	10	1	10

Table 6: Hyperparameters for Diffusion Policy. Ctrl: position or velocity control To: observation horizon Ta: action horizon Tp: action prediction horizon #D-Params: diffusion network number of parameters in millions #V-Params: vision encoder number of parameters in millions Emb Dim: transformer token embedding dimension Attn Dropout: transformer attention dropout probability Lr: learining rate WDecay: weight decay (for transformer only) D-Iters Train: number of training diffusion iterations D-Iters Eval: number of inference diffusion iterations Control Multiplier: multiplier for the low-level control steps.

quaternions to determine the projection. Then we sum the position and orientation distances as the state loss. For the trajectory loss, we take a max over all states.

C.3 Computation Cost

Computing waypoints is inexpensive, especially compared to the training budget. The wall clock time for labeling one trajectory in Lift is 0.8 seconds on average.

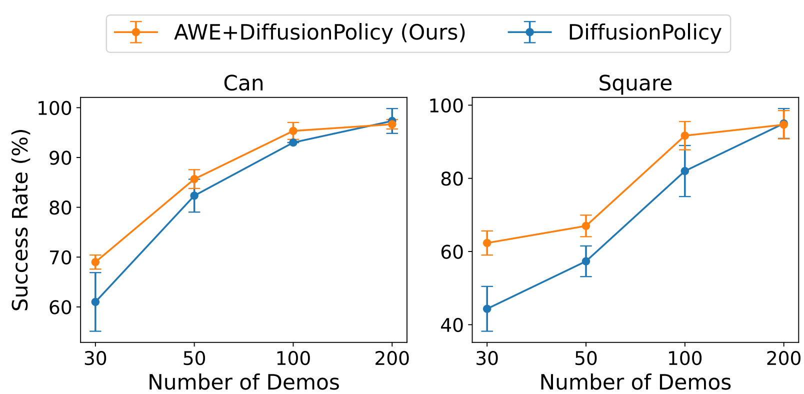

Figure 8: **Performance scaling with demonstrations.** We compare how the performance scale for diffusion policy [5] with and ithout AWE. Training on waypoints generated by AWE consistently improves the performance, with improvements being larger on the harder task (Square).