DARPAN: Controllability-Aware Residual Filtering for Pretrained Physical Control

Anany Pratap Gupta
University of Oklahoma
Norman, OK, USA
anany.p.gupta-1@ou.edu

Ye Zhou
University of Oklahoma
Norman, OK, USA
ye.zhou-1@ou.edu

Qi Li*
University of Oklahoma
Norman, OK, USA
qili@ou.edu

Abstract

Physical control systems, such as building HVAC systems, thermal processes, and power-grid assets, often involve delayed dynamics, safety constraints, and costly online interaction. Because the effect of a control action may appear after several time steps, effective control often requires reasoning over histories of states, actions, and outcomes. Pretrained sequence controllers, such as Decision Transformers, provide a natural way to model this history-dependent control problem by learning policies from offline trajectories, reducing the need for expensive online retraining. However, their performance can degrade when deployment conditions differ from the training data. In building control, for example, actuator wear, sensor drift, occupancy changes, and weather variation can all affect the observed response of the system to the same control action. Online adaptation is therefore needed, but raw prediction error is not always a reliable signal for updating the controller. The key challenge is to identify which part of the observed mismatch is action-sensitive and should guide policy adaptation. To address this challenge, we propose **DARPAN**, a Drift-Aware Residual Policy Adapter Network for pretrained physical control. DARPAN keeps the pretrained Decision Transformer frozen and performs online adaptation only through lightweight residual adapters, preserving the offline policy and limiting the number of trainable parameters. Instead of updating from raw prediction error, DARPAN uses a controllability filter trained in simulation to extract the action-sensitive part of the residual and filter out mismatch that is weakly related to the controller's actions. During deployment, a frozen predictor estimates the next state, the controllability filter produces a filtered residual, and a short history of filtered residuals forms a drift signature. This signature is used to select an existing adapter or create a new one when a persistent drift pattern appears. We benchmark DARPAN on SustainGym under actuator faults, sensor drift, delays, and external disturbances. In the building control environment (BuildingEnv), DARPAN reduces drift-induced performance loss by $31.7 \pm 1.1\%$ and uses 63% fewer online parameters than Online-head DT. We release our code and experiment configurations at https://huggingface.co/ananypratap007/DARPAN2026.

1 Introduction

Physical control systems such as building HVAC networks, thermal processes, and power-grid assets are difficult to adapt online: actions can take several time steps to show effects, safety constraints limit trial-and-error, and deployment data are expensive to collect [14, 8]. These challenges motivate pretrained controllers that learn reusable behavior from offline trajectories before deployment. Sequence models are a natural fit for this setting because they can condition decisions on histories of states, actions, and outcomes, which is important when physical responses are delayed.

*Corresponding author: qili@ou.edu.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

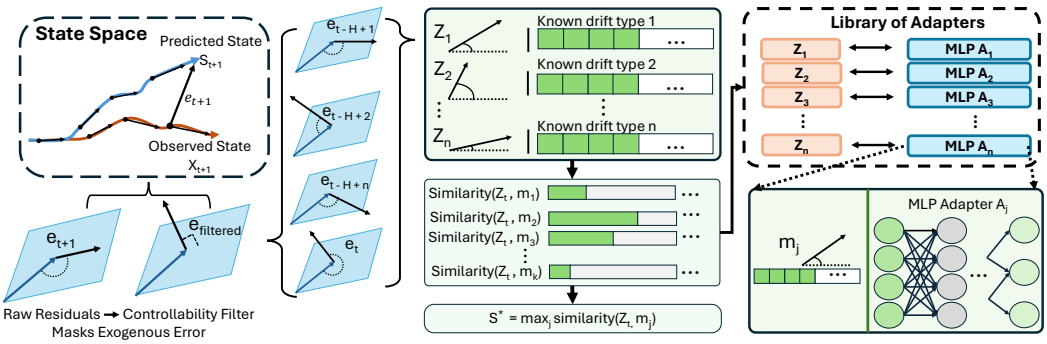

Figure 1: **DARPAN overview: residual filtering and adapter selection.** A frozen predictor forms a prediction residual from the expected and observed next states. DARPAN filters the residual to keep action-related residual components, summarizes recent filtered residuals into a drift signature, and uses this signature to select or create a lightweight residual adapter.

Recent work has explored this idea through Decision Transformers and related pretrained sequence controllers for offline and adaptive control [4, 29, 34, 30, 31, 3]. These methods formulate control as sequence modeling over past states, actions, and returns, making it possible to train controllers from logged trajectories instead of running online reinforcement learning from scratch. Instead of training a new controller for every building, device, or operating condition, a pretrained sequence controller can learn reusable behavior from diverse offline data and then be adapted during deployment [16]. However, reusing a pretrained controller introduces a different problem: the deployed system may not follow the same dynamics as the systems represented in the offline data.

This mismatch is not only an observation problem. In many partially observed control problems, the hidden state is the main source of uncertainty, while the transition dynamics are treated as fixed. In deployed physical systems, however, the system response itself can change over time. In building control, for example, the same heating command may warm a room quickly during training but more slowly after actuator wear, envelope degradation, leakage, shading, or unmodeled airflow changes. Sensors may drift, occupancy patterns may shift, and weather variation may also affect the observed response of the building to the same control action. Because physical control runs in closed loop, small modeling errors can compound along the controller's own rollout, and offline trajectories only cover the operating conditions present during data collection [2, 1, 7]. As a result, offline pretraining alone cannot guarantee reliable behavior under deployment drift.

Prior work has addressed deployment mismatch by continuing to adapt the pretrained controller online [34]. However, full online fine-tuning is risky in slow physical systems: it can require many deployment samples, overwrite useful behavior learned during pretraining, and become unstable when feedback is delayed. To reduce these risks, the pretrained controller can be kept frozen and adapted only through a small residual module. This protects the backbone and reduces the number of trainable parameters, but it raises a key question: what should the adapter learn from?

A straightforward approach is to train the adapter from prediction error. A frozen dynamics predictor can compare the predicted next state with the observed next state and form a residual. However, this raw residual is a mixed signal. A weak heater may create an error that can be reduced by changing the action, but weather shifts, occupancy spikes, or window-opening events may create errors that no action can directly fix. If an adapter learns from every residual, it may spend limited online data adapting to disturbances that should not guide policy updates. Our key insight is that a pretrained physical controller should adapt only to prediction errors that can be influenced by changing the control action. Based on this insight, we introduce **DARPAN**, a Drift-Aware Residual Policy Adapter Network for pretrained physical control. DARPAN keeps the pretrained Decision Transformer frozen and adapts only through lightweight residual adapters. A frozen predictor estimates the expected next state, and a controllability filter keeps the residual components that are most related to the control action. A short history of these filtered residuals forms a drift signature, which is used to select an existing adapter or create a new adapter for the current drift pattern.

During deployment in physical control systems, prediction errors can have different meanings. Some errors indicate that the action channel has changed: for example, an actuator may become weaker, delayed, or partly stuck, so changing the control action can help compensate for the mismatch. Other errors come from factors outside the controller's direct influence, such as weather shifts, occupancy spikes, or transient sensor noise. DARPAN is designed to adapt to the first type of error and avoid false adaptation to the second. By using filtered residuals instead of raw residuals, DARPAN separates two questions: did the model predict the next state incorrectly, and is this error useful for adapting the controller?

We benchmark DARPAN on SustainGym [33] under deployment-time drift in building HVAC control and combined heat-and-power dispatch. The evaluation tests whether DARPAN can improve control under actuator faults and delays, avoid false adaptation under external disturbances, reuse adapters under recurring drift, and reduce online trainable parameters compared with direct online fine-tuning.

Our contributions are summarized as follows:

- We formulate deployment-time adaptation for pretrained physical control as a residual filtering problem, where online updates should use only the part of the prediction error that can be influenced by control actions.
- We introduce DARPAN, a frozen-backbone adaptation method that combines controllability-aware residual filtering, temporal drift signatures, and a lightweight residual-adapter library.
- We design deployment drift protocols that test actuator faults, delays, sensor drift, external disturbances, and recurring drift in physical control benchmarks.
- We show that DARPAN improves control under deployment drift and reduces online trainable parameters compared with direct online fine-tuning.

2 Related Work

We organize related work around three questions: how pretrained sequence controllers are reused in physical control, how frozen controllers can be adapted with small trainable modules, and how deployment-time prediction errors can be filtered before adaptation. These questions correspond to DARPAN's main components: a frozen pretrained Decision Transformer, lightweight residual adapters, and a controllability-aware residual filter.

Pretrained sequence controllers for physical control. Decision Transformers [4] use sequence models to represent policies trained from logged trajectories, conditioning action prediction on histories of states, actions, and returns. This formulation has motivated several ways to reuse pretrained policies at deployment time, including online fine-tuning [34], trajectory stitching [29], prompting [30], and adapter-based Decision Transformers [31]. In building control, HVAC-DPT shows that pretrained transformers can transfer across simulated HVAC systems without changing their main weights [3]. These methods reduce the need for online reinforcement learning by moving most learning into offline pretraining [19, 6]. However, they mainly treat deployment as transfer across tasks, trajectories, or environments. In physical systems, the same deployed system can also change after pretraining because of actuator wear, sensor drift, occupancy changes, or weather effects [28, 32]. DARPAN builds on pretrained sequence control, but focuses on deployment drift with a frozen pretrained policy, where the key issue is deciding which part of the observed mismatch should guide online adaptation.

Parameter-efficient and modular adaptation. A common way to adapt a pretrained controller is to keep the main policy fixed and learn a small correction on top of it. Residual policy learning follows this idea by adding an action correction to an existing controller [22, 12, 1, 2]. Parameter-efficient adaptation methods similarly show that frozen models can be adapted through small trainable modules instead of full-model updates [9, 10, 26]. Hyper-DT applies this idea to Decision Transformers by generating lightweight adaptation modules for a frozen policy [31]. Continual and modular adaptation methods also reduce forgetting by storing knowledge in separate reusable components [17, 21, 35]. These methods show that deployment adaptation can be localized to small modules instead of updating the whole controller. DARPAN follows this principle by keeping the Decision Transformer frozen and updating only a selected residual adapter. The key difference is the update signal: rather than

adapting from the full task loss, raw prediction error, or raw context embedding, DARPAN selects and updates adapters using filtered residuals that reflect mismatch related to the controller's actions.

Residual filtering under deployment shift. A central question in deployment-time adaptation is which online signal should be trusted for updating a pretrained model. Test-time adaptation studies how pretrained models adjust to target data during deployment, often using confidence-based objectives, pseudo-labels, or self-training signals [24, 25, 15]. This work shows that a deployed model does not always need to remain fixed when the target distribution changes. However, most test-time adaptation methods focus on perception tasks, where the update signal is not caused by the model's own control action. Physical control is different: the online signal is often a prediction residual observed after the controller acts. This residual can come from actuator faults, weather shifts, occupancy changes, demand shifts, or sensor noise, and not all of these sources should guide adaptation. A related distinction appears in classical control through matched and unmatched disturbances [5]. Reinforcement learning and representation learning also study controllable or action-dependent factors of variation, encouraging agents to focus on parts of the world that can be influenced by their actions [23, 27, 18, 13, 20]. These works show that control should not treat every observed change as equally useful. DARPAN brings this idea to pretrained physical control by filtering the residual $e_t = x_t - \hat{x}_t$, keeping the part most related to the effect of the controller's action, and using the filtered residual to select and update a lightweight adapter.

3 Method

We study deployment-time adaptation in physical systems with delayed dynamics. At each timestep, the controller observes a state, selects an action, and later observes the next state. The main challenge is that prediction residuals are not automatically reliable update signals. Actuator faults, sensor bias, weather variation, and temporary occupancy changes can all produce large residuals, but only some residuals should change the controller. DARPAN therefore adapts only to residual patterns that are both persistent and related to the effect of the control action.

DARPAN operates in two stages. Offline, we train three components: a Decision Transformer controller, a short-horizon predictor, and a controllability filter. The Decision Transformer learns the base policy from offline trajectories. The predictor estimates the next state expected under the current state and base action. The controllability filter identifies which residual components are related to the effect of the control action. All three components are kept frozen during deployment, so online learning does not overwrite the pretrained controller. The second stage runs online, where DARPAN adapts only through lightweight residual adapters. The frozen Decision Transformer first produces a base action. A selected adapter adds a local correction, and the final action is projected back into the valid action range. After observing the next state, the frozen predictor forms a prediction residual, the controllability filter keeps the action-related part of this residual, and recent filtered residuals are summarized into a drift signature. DARPAN uses this signature to select an existing adapter or create a new one when a persistent drift pattern appears. Figure 2 shows the deployment loop, including residual filtering, drift-signature construction, adapter selection, and adapter update.

Frozen Base Policy. The first component of DARPAN is the pretrained policy that provides the default control behavior. We instantiate this policy as a Decision Transformer [4], denoted by D. At timestep t, the controller receives a recent context window τ_t containing states, past actions, and return-to-go values, and predicts a base action,

$$a_t^{\text{base}} = D(\tau_t).$$

Here, x_t is the current observation and a_t is the control action, normalized to the valid action range. During deployment, D is kept frozen. This preserves the behavior learned from offline trajectories [11], and all online adaptation is handled by lightweight residual adapters.

Prediction Residual. Given the frozen base policy, DARPAN next measures how the deployed system responds to the base action. We use a short-horizon predictor, denoted by F, that is trained offline and kept frozen during deployment. F is a separate one-step prediction model and is not the Decision Transformer's state-prediction head. Given the current state and the base action, the predictor estimates the next state expected under the pretrained dynamics,

$$\hat{x}_{t+1} = F(x_t, a_t^{\text{base}}).$$

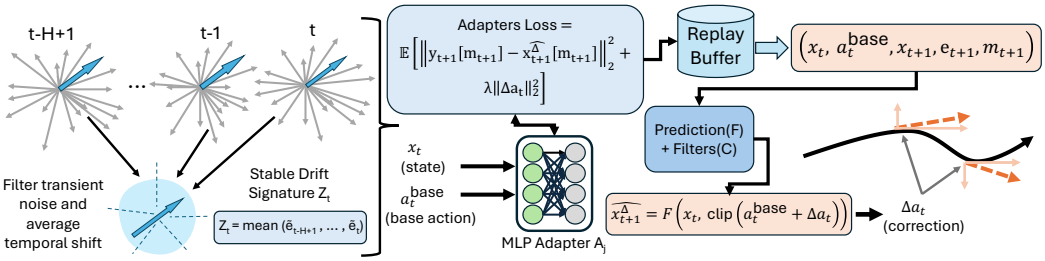

Figure 2: **DARPAN deployment and adapter update.** The frozen predictor produces a raw residual between the predicted and observed next state. The controllability filter keeps only the residual dimensions that are useful for control. A short history from $t - H + 1$ to t is averaged into a drift signature z_t. This signature selects an adapter from the adapter set or creates a new one when the pattern persists. The selected MLP adapter is trained through the frozen predictor using replay-buffer samples and then adds a small correction Δa_t to the frozen Decision Transformer action.

After the action is applied, DARPAN compares the observed next state x_{t+1} with this prediction to form the raw residual,

$$e_{t+1} = x_{t+1} - \hat{x}_{t+1}.$$

This residual measures how the deployed system differs from the frozen predictor's expectation. However, the residual alone does not indicate whether the mismatch is useful for adaptation. For example, actuator degradation may create a residual that can be reduced by changing the action, but a short external disturbance may create a residual that the controller should not learn from.

Controllability-Aware Residual Filtering. The raw residual measures prediction mismatch, but DARPAN should not adapt to every component of this mismatch. We therefore use a frozen controllability filter, denoted by C, to identify residual dimensions that are related to the effect of the control action. Given the current state, the base action, and the raw residual, the filter outputs a score

$$p_{t+1} = C(x_t, a_t^{\text{base}}, e_{t+1}).$$

We convert this score into a binary mask:

$$m_{t+1}[i] = \begin{cases} 1, & \text{if } p_{t+1}[i] > c_{\text{filter}}, \\ 0, & \text{otherwise}, \end{cases}$$

where c_{filter} is a fixed threshold. The filtered residual keeps only the dimensions selected by this mask:

$$\tilde{e}_{t+1}[i] = \begin{cases} e_{t+1}[i], & \text{if } m_{t+1}[i] = 1, \\ 0, & \text{if } m_{t+1}[i] = 0. \end{cases}$$

This filtered residual is the adaptation signal used by DARPAN. Residual components that appear related to the control action, such as a zone-temperature error caused by weak actuation, are kept. Components that appear mainly external, such as weather-driven mismatch, are removed before adapter selection or adapter update.

Filter Training. The controllability filter is trained offline in simulation and then kept frozen during deployment. Its role is to estimate which residual dimensions are likely to change when the control action changes. For each sampled simulator state, we create two types of intervention comparisons. The first changes the control action while keeping outside conditions fixed. If a state variable changes under this intervention, its residual is treated as action-related. The second keeps the action fixed but changes outside conditions, such as weather or occupancy. If a state variable mainly changes under this intervention, its residual is treated as external. The filter learns a score between 0 and 1 for each residual dimension, $p_{t+1} = C(x_t, a_t^{\text{base}}, e_{t+1})$, where higher scores indicate stronger relation to the control action. During deployment, DARPAN keeps residual dimension i only if $p_{t+1}[i] > c_{\text{filter}}$ and sets the remaining dimensions to zero. Additional filter-training details are provided in Appendix C.1.

Temporal Drift Signature. After residual filtering, DARPAN summarizes recent action-related mismatch into a drift signature. A single filtered residual can still be noisy, so the signature is built from a short history rather than only the latest residual. Let H be the history length. At timestep t, DARPAN forms

$$z_t = \mathrm{mean}(\tilde{e}_{t-H+1}, \ldots, \tilde{e}_{t-1}, \tilde{e}_t).$$

The signature z_t summarizes the recent control-relevant mismatch between the frozen predictor and the deployed system. It is not intended to describe the full system state. Instead, it captures whether a similar action-related residual pattern has persisted for several steps. This makes adapter selection more stable because transient disturbances usually disappear quickly, but deployment drift tends to produce repeated residual patterns.

Adapter Selection and Action Correction. The drift signature is used to decide which residual adapter should be active. DARPAN stores a set of residual adapters,

$$\mathcal{A}_t = \{(z_j, A_j)\}_{j=1}^{M_t},$$

where z_j is a stored drift signature and A_j is the adapter associated with that signature. At each timestep, DARPAN compares the current signature z_t with the stored signatures using cosine similarity. A new adapter is created only when the similarity to every stored signature remains below a threshold τ for K_{create} consecutive steps. Otherwise, DARPAN reuses the adapter with the most similar signature. Each adapter is an MLP that predicts a small action correction,

$$\Delta a_t = A_j(x_t).$$

The deployed action is the base action plus the adapter correction, projected back to the valid action range:

$$a_t = \mathrm{Proj}(a_t^{\mathrm{base}} + \Delta a_t).$$

This keeps the frozen Decision Transformer as the main controller, and the adapter only provides a local correction for the current drift pattern.

Online Adapter Update. During deployment, only the selected adapter is updated. The base policy D, predictor F, controllability filter C, and all other adapters remain frozen. This keeps online learning local to the current drift pattern. For a stored transition $(x_t, a_t^{\mathrm{base}}, x_{t+1}, m_{t+1})$, DARPAN first computes the frozen prediction and raw residual,

$$\hat{x}_{t+1} = F(x_t, a_t^{\mathrm{base}}), \qquad e_{t+1} = x_{t+1} - \hat{x}_{t+1}.$$

It then builds a corrective target,

$$y_{t+1} = \hat{x}_{t+1} - \alpha e_{t+1},$$

where $\alpha \in (0, 1)$ controls the correction strength. The adapter predicts a correction $\Delta a_t = A_j(x_t)$, and the frozen predictor estimates the next state under the corrected action:

$$\hat{x}_{t+1}^{\Delta} = F(x_t, \mathrm{Proj}(a_t^{\mathrm{base}} + \Delta a_t)).$$

Let $v[m]$ denote the entries of a vector v whose mask value is one. The selected adapter is trained with

$$\mathcal{L}_{\mathrm{adapter}} = \mathbb{E}\left[\left\|y_{t+1}[m_{t+1}] - \hat{x}_{t+1}^{\Delta}[m_{t+1}]\right\|_2^2 + \lambda \|\Delta a_t\|_2^2\right].$$

The first term trains the adapter only on residual dimensions marked useful for control. The second term keeps the correction small, so the frozen Decision Transformer remains the main controller. Each adapter is updated using small batches from its own replay buffer to avoid large updates from single noisy transitions.

Deployment Procedure. At deployment, DARPAN runs as a closed-loop adaptation procedure. Given the current state x_t, the frozen Decision Transformer predicts a base action $a_t^{\mathrm{base}} = D(\tau_t)$. The current drift signature z_t selects an adapter A_{k_t}, which produces a correction $\Delta a_t = A_{k_t}(x_t)$. The deployed action is

$$a_t = \mathrm{Proj}(a_t^{\mathrm{base}} + \Delta a_t).$$

After observing the next state x_{t+1}, the frozen predictor forms the residual $e_{t+1} = x_{t+1} - F(x_t, a_t^{\mathrm{base}})$. The controllability filter converts this residual into a mask m_{t+1} and filtered residual

Table 1: **Performance and online parameter cost under deployment drift.** Online params count the parameters updated during deployment. Recovery measures the percentage of the Frozen-DT-to-reference physical penalty gap reduced by each method. Higher recovery and lower online parameter count are better.

Method	Building params ↓	Cogen params ↓	Building recovery (%) ↑	Cogen recovery (%) ↑
Online PPO	10,189	8,095	-12.5 ± 6.4	-18.2 ± 7.1
Full online DT	387,503	162,244	-8.7 ± 5.2	-5.4 ± 4.8
Online-head DT	390	975	20.1 ± 2.4	17.8 ± 2.5
LoRA-DT	768	3,388	1.9 ± 1.2	2.4 ± 1.5
Hyper-DT	642	892	13.5 ± 0.9	24.9 ± 1.1
DARPAN	**142**	**495**	$\mathbf{31.7 \pm 1.1}$	$\mathbf{28.5 \pm 1.4}$

$\tilde{e}_{t+1}$. The filtered residual updates the drift signature, the transition is stored in the selected adapter's replay buffer, and only the selected adapter is periodically updated.

This procedure separates deployment adaptation into three roles: the predictor measures mismatch, the filter keeps the action-related part of the mismatch, and the adapter set decides whether to reuse an existing correction or learn a new one.

4 Experimental Evaluation

To evaluate deployment-time adaptation under physical drift, we test four properties: control improvement under action-channel changes, false-adaptation prevention under external disturbances, adapter reuse when similar drift returns, and online parameter cost compared with direct fine-tuning and parameter-efficient baselines.

We evaluate DARPAN on two SustainGym environments: `BuildingEnv` for HVAC comfort control and `CogenEnv` for combined heat-and-power dispatch [33]. These environments differ in dynamics, rewards, control intervals, and action spaces, but require adapting a pretrained controller when deployment dynamics differ from the offline training setting. `BuildingEnv` tests thermal comfort control under building drift, while `CogenEnv` tests energy dispatch under faults and demand shifts.

After defining the deployment-drift protocols in these environments, we compare DARPAN with adaptation methods that update different parts of the pretrained controller. To isolate the effect of deployment-time adaptation, all transformer-based baselines use the same pretrained Decision Transformer backbone [4]. We compare against Frozen DT, full online DT fine-tuning [34], online-head DT, LoRA-DT [10, 26], Hyper-DT [31], and Online PPO. Frozen DT measures performance without deployment adaptation. Full online DT tests whole-controller fine-tuning under delayed physical feedback. Online-head DT, LoRA-DT, and Hyper-DT test parameter-efficient adaptation without DARPAN's filtered residual signature. Online PPO tests whether learning from scratch online is competitive in this setting. We report online parameter count and physical recovery. Given the Frozen DT penalty P_{frozen}, clean reference penalty P_{ref}, and method penalty P_{method}, we compute

$$\text{Recovery} = 100 \cdot \frac{P_{\text{frozen}} - P_{\text{method}}}{P_{\text{frozen}} - P_{\text{ref}}}.$$

Higher recovery means the method closes more of the drift-induced physical penalty gap. For `BuildingEnv`, P is comfort violation; for `CogenEnv`, P is the environment-specific physical penalty defined in Appendix E.

4.1 Performance and Parameter Cost Under Deployment Drift

To benchmark performance and online parameter cost under deployment drift, we compare DARPAN with full online fine-tuning and parameter-efficient adaptation baselines. Table 1 shows that DARPAN gives the best performance in both environments while updating the fewest trainable parameters online. On `BuildingEnv`, DARPAN reduces the drift-induced comfort penalty by $31.7 \pm 1.1\%$ with only 142 online trainable parameters. Online-head DT reduces the penalty by $20.1 \pm 2.4\%$ with 390 parameters, Hyper-DT by $13.5 \pm 0.9\%$ with 642 parameters, and LoRA-DT by $1.9 \pm 1.2\%$ with 768 parameters. On `CogenEnv`, DARPAN also gives the best result, reducing the drift-induced physical penalty by $28.5 \pm 1.4\%$.

The performance gain comes from the filtered adaptation signal rather than from a larger number of trainable parameters. Full online DT has many more trainable parameters, but whole-controller updates can overwrite useful pretrained behavior under delayed physical feedback. LoRA-DT and Hyper-DT are more parameter-efficient, but they still adapt without DARPAN's controllability-aware residual filtering. DARPAN first removes residual dimensions that are unlikely to respond to control actions, then uses the remaining residual pattern to update only a lightweight adapter.

4.2 False Adaptation Under External Disturbances

To test whether DARPAN avoids learning from residuals outside the influence of control, we evaluate disturbance-only phases with weather, occupancy, and noise shifts. In these phases, the controller observes prediction residuals, but there is no actuator fault or delay that should require a new adapter. A reliable method should avoid treating every large residual as a new drift pattern. Table 2 compares DARPAN with a raw-residual adapter and Hyper-DT. The raw-residual adapter creates 30.60 ± 5.59 unnecessary adapters, while DARPAN creates only 1.20 ± 0.45, reducing false creations by 96%. DARPAN also requires fewer adapter updates and achieves the largest mean penalty reduction relative to the frozen controller, measured by ΔP_{dist}.

The result shows why residual filtering is needed. A raw residual only indicates that the predictor was wrong; it does not indicate whether the controller can influence the source of the error. The raw-residual adapter therefore creates many unnecessary adapters during disturbance-only phases. DARPAN filters the residual before adapter selection, so residual components caused mostly by weather, occupancy, or noise are less likely to create new adapters. This keeps the adapter set focused on persistent action-related drift.

Table 2: **False adaptation under external disturbances.** False creations count adapters created during disturbance-only phases with weather, occupancy, or noise shifts. Hyper-DT has no adapter set, so false creations do not apply. We report the penalty difference $\Delta P_{\text{dist}} = P_{\text{frozen}} - P_{\text{method}}$, where positive values indicate a lower penalty than the frozen controller. Results are reported as mean $\pm$ standard deviation across seeds.

Method	False creations ↓	Adapter updates ↓	ΔP_{dist} ↑
Raw residual adapter	30.60 ± 5.59	190 ± 27	$+50.5 \pm 54.4$
Hyper-DT	—	864 ± 0	$+11.3 \pm 2.4$
DARPAN	$\mathbf{1.20 \pm 0.45}$	$\mathbf{102 \pm 1}$	$\mathbf{+95.5 \pm 36.2}$

4.3 Continual Adaptation and Adapter Reuse

To test whether DARPAN can reuse learned corrections, we evaluate it in a 60-day continual drift protocol. The protocol exposes the frozen controller to weak actuators, sensor bias, actuation delay, stuck actuators, repeated faults, and combined faults. This setting tests whether DARPAN can do more than improve control immediately after a drift appears. It also tests whether the adapter set stores reusable correction patterns when similar drift returns later.

Figure 3 shows control improvement and adapter use over time. The top panel reports cumulative comfort-penalty reduction relative to Frozen DT, and the bottom panel shows the active adapter index. DARPAN maintains positive improvement across the 60-day sequence. At the same time, the adapter trace does not change at every new disturbance. Long flat regions indicate that an existing adapter is being reused, while new adapters appear only when the filtered residual signature remains different from stored signatures. In the final combined-fault phase, DARPAN often reuses adapters learned from earlier single-fault phases rather than creating a new adapter for every fault combination. This suggests that the adapter set stores reusable correction patterns in residual space rather than memorizing fault names.

4.4 Ablation Study

To test which components are responsible for DARPAN's behavior, we evaluate four ablations. *Raw residual* removes the controllability filter and adapts from the unfiltered residual. *Single-step signature* removes the temporal residual history and uses only the latest filtered residual. *Naive expansion* removes the conservative adapter-creation rule and creates adapters whenever a new residual pattern

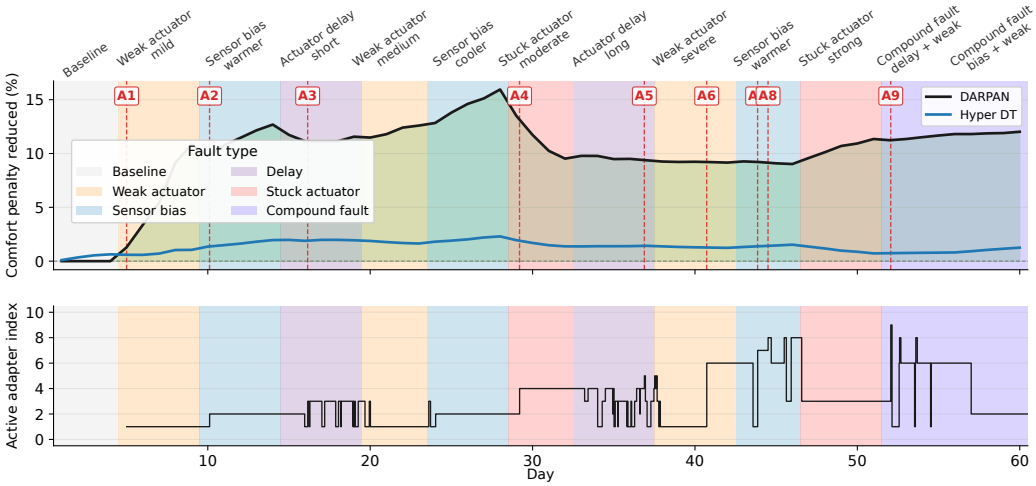

Figure 3: **Continual adaptation and adapter reuse under sequential drift.** Top: cumulative comfort-penalty reduction relative to the frozen Decision Transformer, normalized by the frozen model's total comfort penalty. Bottom: active adapter index over time, with A_1 to A_9 marking adapter creation events. DARPAN maintains positive improvement while reusing existing adapters across recurring and combined faults.

appears. *One global adapter* removes the adapter set and updates a single adapter for all drift patterns. The ablation experiments use a different set of random seeds from the disturbance-only experiments in Section 4.2.

The ablations fail in different ways, showing that each component has a distinct role. Removing the controllability filter reduces recovery from 31.7% to 20.3% and increases false creations from 1.4 to 30.6, showing that raw residuals lead to unnecessary adaptation. Removing the temporal history makes adapter selection sensitive to one-step noise: recovery drops to 18.5%, the adapter set grows to 26 adapters, and reuse falls to 55.2%. Naive expansion keeps recovery relatively high at 28.2%, but grows the adapter set to 78 adapters and reduces reuse to 12.0%. Using one global adapter gives the lowest recovery, 8.4%, because later drift patterns overwrite earlier corrections. Together, these results show that DARPAN needs residual filtering, temporal signatures, conservative adapter creation, and separate adapters for different drift patterns.

Table 3: **Ablation of DARPAN components.** Each variant removes one design choice. Recovery measures the percentage of the Frozen-DT-to-reference physical penalty gap reduced by each method. False creations count unnecessary adapters created during disturbance-only phases.

Variant	Recovery (%) ↑	False creations ↓	Library size ↓	Reuse rate (%) ↑
Full DARPAN	$\mathbf{31.7 \pm 1.1}$	$\mathbf{1.4 \pm 0.9}$	$\mathbf{9 \pm 1.5}$	$\mathbf{92.6 \pm 0.8}$
Raw residual, no filter	20.3 ± 3.4	30.6 ± 5.6	12 ± 1.5	21.5 ± 2.1
Single-step signature	18.5 ± 1.9	14.8 ± 3.2	26 ± 0.0	55.2 ± 3.5
Naive expansion	28.2 ± 3.4	4.1 ± 1.1	78 ± 4.2	12.0 ± 1.8
One global adapter	8.4 ± 1.5	—	1 ± 0.0	—

4.5 Filter Generalization to Held-Out Drift Mechanisms

In `BuildingEnv`, the filter is trained entirely from simulator counterfactuals by varying actions and external conditions, without observing fault identities or using the evaluation fault catalog. `CogenEnv` uses fault-to-dimension supervision only because its ONNX interface does not provide the same counterfactual reset capability. To test whether the `CogenEnv` filter generalizes beyond its training catalog, we ran a leave-one-mechanism-out evaluation over four mechanism families, covering wind shift, sensor drift, GT1 degradation, and multi-turbine degradation. For each condition, we retrained the controllability filter from scratch after removing every filter-training trajectory containing the

held-out mechanism, including all compound trajectories in which that mechanism appeared. Fault identities and classes are not provided as inputs to the filter.

Table 4 shows that exact exposure to the evaluated mechanism is not required. Holding out wind shift increased false adaptation on F5 from 1.3% to 3.4%, while holding out sensor drift increased it on F3/F4 from 2.6% to 4.9% (fault IDs follow Table 11). Even without these mechanisms in training, 96.6% and 95.1% of timesteps did not trigger an adapter update. Compound-fault recovery changed only slightly, from 48.5% to 46.9% on F7 and from 46.0% to 44.3% on F8. Recovery also remained stable for held-out controllable faults, and overall active-fault recovery changed from 38.4% to 37.2%. This aggregate includes only controllable and compound fault segments, but the 28.5% in Table 1 also includes clean and exogenous segments.

Table 4: Leave-one-mechanism-out evaluation of the `CogenEnv` controllability filter. False adaptation rate is the percentage of timesteps in exogenous segments that trigger an adapter update. Recovery is reported for controllable or compound segments.

Held-out mechanism	Evaluation segment	Metric	Full catalog	Held out
Wind shift	F5	False adaptation rate (%)	1.3 ± 0.5	3.4 ± 0.6
Wind shift	F7	Recovery (%)	48.5 ± 3.4	46.9 ± 6.8
Sensor drift	F3/F4	False adaptation rate (%)	2.6 ± 0.2	4.9 ± 0.3
Sensor drift	F8	Recovery (%)	46.0 ± 2.7	44.3 ± 3.2
GT1 degradation	F1 segments	Recovery (%)	43.6 ± 3.3	46.1 ± 3.1
Multi-turbine degradation	F2	Recovery (%)	34.8 ± 2.3	34.2 ± 4.9
Active-fault aggregate	Controllable/compound segments	Recovery (%)	38.4 ± 1.8	37.2 ± 3.9

5 Conclusion

We presented DARPAN, a controllability-aware adaptation method for pretrained physical controllers. DARPAN keeps the Decision Transformer backbone frozen and updates only lightweight residual adapters, allowing the controller to adapt without overwriting behavior learned from offline trajectories. The main insight is that, in physical systems, not every prediction error should trigger learning. DARPAN filters residuals to focus on errors related to the effect of control actions, summarizes recent filtered residuals into drift signatures, and reuses adapters when similar drift patterns return.

The main limitations of DARPAN are its dependence on a reliable controllability filter and the controlled scope of the evaluation. The filter is trained from simulator counterfactuals in `BuildingEnv` and labeled fault trajectories in `CogenEnv`, but real deployments may contain residual patterns not covered by these sources. The current evaluation uses a fixed building configuration and a finite drift catalog, so future work should study filter training from logged interventions, uncertainty-aware prediction, and stronger adapter management for longer real-world deployments.

Acknowledgments and Disclosure of Funding

This work was supported in part by the University of Oklahoma through seed grants from the Institute for Community and Society Transformation (ICAST) and the Data Institute for Societal Challenges (DISC), as well as institutional research support.

References

[1] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting physical simulators with stochastic neural networks: Case study of planar pushing and bouncing. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3066–3073. IEEE, 2018.

[2] Lars Lien Ankile, Anthony Simeonov, Idan Shenfeld, Marcel Torne Villasevil, and Pulkit Agrawal. From imitation to refinement–residual rl for precise visual assembly. In *CoRL 2024 Workshop on Mastering Robot Manipulation in a World of Abundant Data*, 2024.

[3] Anaïs Berkes. Hvac-dpt: A decision pretrained transformer for hvac control. *arXiv preprint arXiv:2411.19746*, 2024.

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[5] Wen-Hua Chen, Jun Yang, Lei Guo, and Shihua Li. Disturbance-observer-based control and related methods—an overview. *IEEE Transactions on industrial electronics*, 63(2):1083–1095, 2015.

[6] Seongwoong Cho, Donggyun Kim, Jinwoo Lee, and Seunghoon Hong. Meta-controller: Few-shot imitation of unseen embodiments and tasks in continuous control. *Advances in Neural Information Processing Systems*, 37:134250–134286, 2024.

[7] Pim De Haan, Dinesh Jayaraman, and Sergey Levine. Causal confusion in imitation learning. *Advances in neural information processing systems*, 32, 2019.

[8] Patrick Emami, Abhijeet Sahu, and Peter Graf. Buildingsbench: A large-scale dataset of 900k buildings and benchmark for short-term load forecasting. *Advances in Neural Information Processing Systems*, 36:19823–19857, 2023.

[9] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR, 2019.

[10] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[11] Michael Janner, Qiyang Li, and Sergey Levine. Offline reinforcement learning as one big sequence modeling problem. *Advances in neural information processing systems*, 34:1273–1286, 2021.

[12] Tobias Johannink, Shikhar Bahl, Ashvin Nair, Jianlan Luo, Avinash Kumar, Matthias Loskyll, Juan Aparicio Ojea, Eugen Solowjow, and Sergey Levine. Residual reinforcement learning for robot control. In *2019 international conference on robotics and automation (ICRA)*, pages 6023–6029. IEEE, 2019.

[13] Alex Lamb, Riashat Islam, Yonathan Efroni, Aniket Rajiv Didolkar, Dipendra Misra, Dylan J Foster, Lekan P Molu, Rajan Chari, Akshay Krishnamurthy, and John Langford. Guaranteed discovery of control-endogenous latent states with multi-step inverse models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=TNocbXm5MZ`.

[14] Nevena Lazic, Craig Boutilier, Tyler Lu, Eehern Wong, Binz Roy, MK Ryu, and Greg Imwalle. Data center cooling using model-predictive control. *Advances in Neural Information Processing Systems*, 31, 2018.

[15] Jian Liang, Ran He, and Tieniu Tan. A comprehensive survey on test-time adaptation under distribution shifts. *International Journal of Computer Vision*, 133(1):31–64, 2025.

[16] Fangchen Liu, Hao Liu, Aditya Grover, and Pieter Abbeel. Masked autoencoding for scalable and generalizable decision making. *Advances in Neural Information Processing Systems*, 35: 12608–12618, 2022.

[17] Jianbiao Mei, Yukai Ma, Xuemeng Yang, Licheng Wen, Xinyu Cai, Xin Li, Daocheng Fu, Bo Zhang, Pinlong Cai, Min Dou, et al. Continuously learning, adapting, and improving: A dual-process approach to autonomous driving. *Advances in Neural Information Processing Systems*, 37:123261–123290, 2024.

[18] Minting Pan, Xiangming Zhu, Yunbo Wang, and Xiaokang Yang. Iso-dream: Isolating and leveraging noncontrollable visual dynamics in world models. *Advances in neural information processing systems*, 35:23178–23191, 2022.

[19] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[20] Maximilian Seitzer, Bernhard Schölkopf, and Georg Martius. Causal influence detection for improving efficiency in reinforcement learning. *Advances in Neural Information Processing Systems*, 34:22905–22918, 2021.

[21] Guangyuan Shi, Jiaxin Chen, Wenlong Zhang, Li-Ming Zhan, and Xiao-Ming Wu. Overcoming catastrophic forgetting in incremental few-shot learning by finding flat minima. *Advances in neural information processing systems*, 34:6747–6761, 2021.

[22] Tom Silver, Kelsey Allen, Josh Tenenbaum, and Leslie Kaelbling. Residual policy learning. *arXiv preprint arXiv:1812.06298*, 2018.

[23] Valentin Thomas, Emmanuel Bengio, William Fedus, Jules Pondard, Philippe Beaudoin, Hugo Larochelle, Joelle Pineau, Doina Precup, and Yoshua Bengio. Disentangling the independently controllable factors of variation by interacting with the world. *arXiv preprint arXiv:1802.09484*, 2018.

[24] Dequan Wang, Evan Shelhamer, Shaoteng Liu, Bruno Olshausen, and Trevor Darrell. Tent: Fully test-time adaptation by entropy minimization. *arXiv preprint arXiv:2006.10726*, 2020.

[25] Qin Wang, Olga Fink, Luc Van Gool, and Dengxin Dai. Continual test-time domain adaptation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7201–7211, 2022.

[26] Shaowen Wang, Linxi Yu, and Jian Li. Lora-ga: Low-rank adaptation with gradient approximation. *Advances in Neural Information Processing Systems*, 37:54905–54931, 2024.

[27] Tongzhou Wang, Simon S Du, Antonio Torralba, Phillip Isola, Amy Zhang, and Yuandong Tian. Denoised mdps: Learning world models better than the world itself. *arXiv preprint arXiv:2206.15477*, 2022.

[28] Long Wei, Peiyan Hu, Ruiqi Feng, Haodong Feng, Yixuan Du, Tao Zhang, Rui Wang, Yue Wang, Zhi-Ming Ma, and Tailin Wu. Diffphycon: a generative approach to control complex physical systems. *Advances in Neural Information Processing Systems*, 37:4090–4147, 2024.

[29] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in neural information processing systems*, 36:18532–18550, 2023.

[30] Mengdi Xu, Yikang Shen, Shun Zhang, Yuchen Lu, Ding Zhao, Joshua Tenenbaum, and Chuang Gan. Prompting decision transformer for few-shot policy generalization. In *international conference on machine learning*, pages 24631–24645. PMLR, 2022.

[31] Mengdi Xu, Yuchen Lu, Yikang Shen, Shun Zhang, Ding Zhao, and Chuang Gan. Hyper-decision transformer for efficient online policy adaptation. In *The Eleventh International Conference on Learning Representations*, 2023.

[32] Shuqi Yang, Xingzhe He, and Bo Zhu. Learning physical constraints with neural projections. *Advances in Neural Information Processing Systems*, 33:5178–5189, 2020.

[33] Christopher Yeh, Victor Li, Rajeev Datta, Julio Arroyo, Nicolas Christianson, Chi Zhang, Yize Chen, Mohammad Mehdi Hosseini, Azarang Golmohammadi, Yuanyuan Shi, et al. Sustaingym: Reinforcement learning environments for sustainable energy systems. *Advances in Neural Information Processing Systems*, 36:59464–59476, 2023.

[34] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *international conference on machine learning*, pages 27042–27059. PMLR, 2022.

[35] Yuxuan Zhou, Xien Liu, Xiao Zhang, Chen Ning, Shijin Wang, Guoping Hu, and Ji Wu. Investigating and mitigating catastrophic forgetting in medical knowledge injection through internal knowledge augmentation learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Appendix A Environment Details

We pretrain and evaluate DARPAN on two SustainGym [33] environments: `BuildingEnv` and `CogenEnv`. SustainGym is publicly available at `https://github.com/chrisyeh96/sustaingym` under the CC BY 4.0 license.

We use both environments because they test DARPAN under two different kinds of physical control problems. `BuildingEnv` is a thermal building-control task with continuous HVAC actions and a short 5-minute control interval. `CogenEnv` is an energy-dispatch task with a fitted nonlinear plant model, mixed action types, and a longer 15-minute control interval. Together, these two settings let us test whether the same adaptation idea works across different dynamics, timescales, action spaces, and deployment shifts.

A.1 `BuildingEnv`

`BuildingEnv` simulates thermal control in a multi-zone building. The building dynamics follow a resistance–capacitance thermal model, with additional nonlinear effects from solar gain and occupancy. At every 5-minute step, the controller chooses one continuous heating/cooling action per zone in $[-1, 1]$. The reward mainly penalizes deviation from the target temperature, with a small penalty on actuation. Each episode lasts one day, so the horizon is $T = 288$ steps.

We use the `OfficeSmall` configuration, which has $n = 6$ zones, a 10-dimensional state, and a 6-dimensional action. This choice keeps the control problem fixed while we study deployment-time adaptation. SustainGym includes many building types with different zone counts, ranging from small buildings to buildings with more than one hundred zones. However, changing the number of zones also changes the input and output dimensions of the Decision Transformer. Handling all building sizes with one model would require extra machinery, such as per-building heads or a hypernetwork over zone count. That is an interesting problem, but it is separate from the question studied in this paper. We therefore fix one building configuration and focus on whether DARPAN can adapt when the same deployed system changes over time.

For pretraining diversity, we vary the `stochastic_summer_percentage` parameter. This changes the weather distribution by mixing standard TMY3 weather with a summer-weighted resampled distribution. Thus, the pretrained controller sees different ambient conditions during data collection. At evaluation time, we introduce stronger deployment shifts using a fault catalog. These shifts include actuator weakening, window-opening disturbances, and sensor drift. The full drift protocol is described in Appendix E.

A.2 `CogenEnv`

`CogenEnv` simulates a combined-cycle gas plant with three gas turbines and one steam turbine. The plant response is given by a fitted ONNX model. At every 15-minute step, the controller chooses plant dispatch actions, and the reward penalizes fuel cost, ramping cost, supply–demand mismatch, and operating-constraint violations. Each episode lasts one day, giving a horizon of $T = 96$ steps.

The native action space is a 15-dimensional `Dict` with 8 continuous dispatch variables, 6 binary unit-level switches, and a 12-level integer cooling-tower-bay count. This mixed structure is difficult to use with a single fixed-shape Decision Transformer action head. We therefore use the project `CogenEnvWrapper`, which converts the native action space into a flat 15-dimensional vector in $[-1, 1]$. Continuous actions are mapped linearly to their physical ranges, binary actions are thresholded at zero, and the bay count is mapped linearly and rounded. The wrapper also provides a flat 44-dimensional observation containing time of day, the previous action, and short forecasts of ambient conditions, demand, prices, and fuel cost. The projection is deterministic; the only loss of precision comes from rounding the bay count.

For pretraining diversity, we vary two parameters. `renewables_magnitude` subtracts a wind-power profile from electricity demand, using 0 MW and 300 MW following SustainGym, which changes the structure of the demand profile. `forecast_noise_std` adds Gaussian noise to future forecast entries, making the planning information less reliable without directly changing the plant physics. At evaluation time, we introduce stronger deployment shifts through a fault catalog with nine plant

faults, including turbine degradation, ambient sensor bias, wind shifts, and compound faults. These faults are evaluated over a 14-day rollout, as described in Appendix E.

Appendix B Decision Transformer and Prediction Model Training

We pretrain the Decision Transformer on offline trajectories collected from two SustainGym [33] environments: `BuildingEnv` and `CogenEnv`. Both environments are simulators, so we generate the offline dataset using scripted behavior policies instead of costly online exploration or expert demonstrations. The goal of pretraining is not to solve every possible deployment shift in advance. Instead, it gives the controller a useful base policy and a short-horizon prediction model before deployment-time adaptation begins.

Across both environments, we follow the original Decision Transformer setup [4] as closely as possible: trajectories are represented as sequences of returns, states, and actions; training samples are formed using fixed-length context windows; and the action prediction objective is trained from offline rollouts. Environment-specific details, such as action discretization, state targets, and reward normalization, are changed only when required by the simulator interface or by empirical stability. This keeps the comparison focused on DARPAN's deployment-time adaptation mechanism rather than on a specially tuned backbone.

Figure 4 summarizes the shared Decision Transformer backbone used for both environments before DARPAN performs deployment-time adaptation.

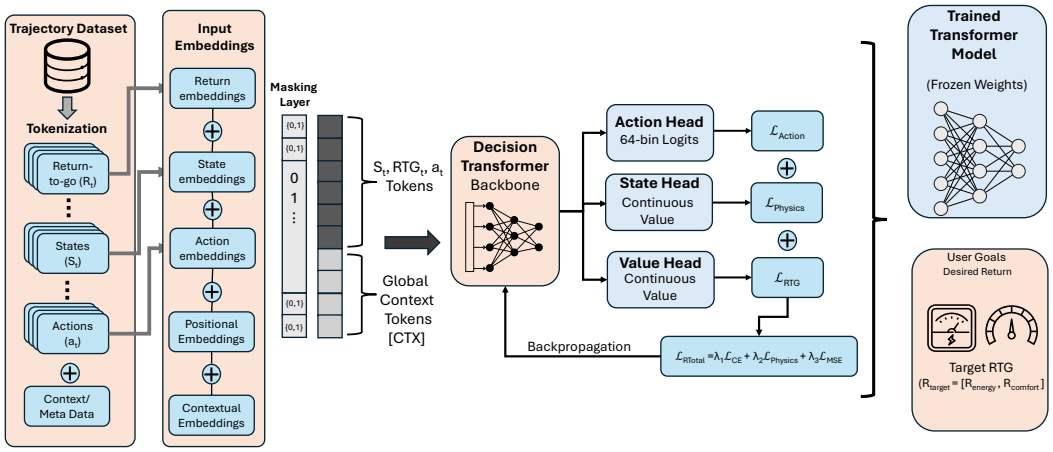

Figure 4: Decision Transformer backbone used during pretraining. Offline trajectories are tokenized into return, state, action, and context tokens, and the model is trained with action, state-prediction, and value objectives. During deployment, the backbone is frozen.

B.1 Offline Data Generation

For each environment, we collect trajectories using a small set of scripted behavior policies. We include a uniform random policy to cover a broad range of states and actions. We also include simple rule-based controllers that reflect reasonable operating behavior, such as thermostat-style control in `BuildingEnv` and demand-tracking dispatch in `CogenEnv`. Finally, we add mildly perturbed versions of the rule-based policies by injecting Gaussian noise into the actions. This gives the offline dataset both low-return exploratory behavior and higher-return structured behavior, which is the setting in which Decision Transformers are commonly trained.

B.2 Trajectory Format

Each trajectory is a sequence $\{(s_t, a_t, r_t)\}_{t=0}^{T-1}$ over one episode. The horizon is $T = 288$ for `BuildingEnv`, corresponding to 24 hours at 5-minute intervals, and $T = 96$ for `CogenEnv`, corresponding to 24 hours at 15-minute intervals. At each timestep, the return-to-go is computed

as

$$R_t = \sum_{t'=t}^{T-1} r_{t'}. \tag{1}$$

Training samples are then created by sliding a context window of length K over each trajectory. The transformer receives the tokenized return, state, and action history inside this window and predicts the next action, together with auxiliary prediction targets described below.

B.3 `BuildingEnv` Pretraining

`BuildingEnv` is a multi-zone HVAC control task. For a building with n zones, the state contains the zone temperatures, outdoor temperature, ground temperature, solar irradiance, and occupancy heat gain:

$$s(t) = \big(T_1(t), \ldots, T_n(t), T_E(t), T_G(t), Q^{\mathrm{GHI}}(t), \bar{Q}^p(t)\big) \in \mathbb{R}^{n+4}.$$

The action $a(t) \in [-1, 1]^n$ gives the normalized heating or cooling command for each zone, where positive values correspond to heating and negative values correspond to cooling. The reward penalizes deviation from the target temperature and includes a small action-cost term:

$$r(t) = -\Big(q_{\mathrm{rate}} \|a(t)\|_p + e_{\mathrm{rate}} \big\|(T(t) - T^{\mathrm{target}}(t)) \odot m_{\mathrm{ac}}\big\|_p\Big). \tag{2}$$

We use the environment default values $q_{\mathrm{rate}} = (1 - \beta) \cdot 24$, $e_{\mathrm{rate}} = \beta$, $p = 2$, and $\beta = 0.999$. This makes comfort the dominant term while still discouraging unnecessary actuation.

We use the `OfficeSmall` configuration, which has $n = 6$ zones, a 10-dimensional state, and a 6-dimensional action. This fixes the input and output dimensions of the Decision Transformer. SustainGym includes buildings with many different zone counts, but training one shared transformer across all of them would require additional architecture choices, such as per-building input/output heads or a zone-count-conditioned hypernetwork. That is a separate problem from the deployment-time adaptation question studied in this paper, so we keep the building configuration fixed.

For data collection, we use eight behavior policies built from four main types. The first samples each zone action uniformly from $[-1, 1]$. The second is a bang-bang thermostat controller that heats when a zone is below the target band, cools when it is above the target band, and otherwise applies no action. The third is a proportional controller,

$$a_i(t) = \mathrm{clip}\big(K_p(T_i^{\mathrm{target}}(t) - T_i(t)), -1, 1\big),$$

using several gain values. The fourth adds Gaussian noise to the proportional controller before clipping. To expose the model to different weather regimes, we also vary `stochastic_summer_percentage`, which changes the ambient distribution between standard TMY3 weather and a summer-weighted resampled distribution.

For tokenization, building-wide variables are represented as global context: outdoor temperature, ground temperature, solar irradiance, occupancy heat gain, and time of day. Each zone is represented by a local token containing its temperature, previous action, and target temperature. The target temperature is included because it is part of the control objective, even though it is provided by the environment configuration rather than by the raw state vector.

We discretize each zone action into $B = 21$ uniform bins over $[-1, 1]$. The action head predicts a categorical distribution over these bins for each zone, and the action loss is the sum of the per-zone classification losses. The auxiliary state head predicts the next-step zone temperatures $\{T_i(t+1)\}_{i=1}^n$. This prediction target encourages the backbone to learn the short-term thermal response of the building, including the effects of HVAC actions, outdoor temperature, solar gain, and occupancy.

B.4 `CogenEnv` Pretraining

`CogenEnv` is a combined-cycle gas plant dispatch task with three gas turbines and one steam turbine. The simulator uses a fitted ONNX model for the plant response. The native action space is mixed: it contains continuous dispatch variables, binary unit switches, and an integer cooling-tower-bay count. Because a standard Decision Transformer expects a fixed-shape action vector, we use the project `CogenEnvWrapper`. The wrapper maps the native action into a 15-dimensional vector in

$[-1, 1]$: continuous dimensions are mapped linearly, binary dimensions are thresholded at zero, and the cooling-tower-bay count is rounded after projection.

The wrapped state is a 44-dimensional vector:

$$s(t) = \big[\tau,\, a(t-1),\, T^{\mathrm{amb}}_{t:t+F},\, P^{\mathrm{amb}}_{t:t+F},\, H^{\mathrm{amb}}_{t:t+F},\, d^{p}_{t:t+F},\, d^{q}_{t:t+F},\, \pi^{p}_{t:t+F},\, \pi^{f}_{t:t+F}\big],$$

where τ is the fraction of the day, $a(t-1)$ is the previous flat action, and the remaining terms are short forecasts of ambient conditions, electricity demand, steam demand, electricity price, and fuel price. The default forecast horizon is $F = 3$.

The per-step reward penalizes fuel cost, ramping, supply–demand mismatch, and operating-constraint violations:

$$r(t) = -\Big(c_{\mathrm{fuel}}(t) + \lambda_r c_{\mathrm{ramp}}(t) + \lambda_s c_{\mathrm{sd}}(t) + \lambda_c c_{\mathrm{dyn}}(t)\Big). \tag{3}$$

We use the environment default weights $\lambda_r = 2$, $\lambda_s = 1000$, and $\lambda_c = 1000$. These defaults strongly penalize infeasible operation, which is important because small dispatch errors can create large supply–demand or constraint violations.

We use three behavior policies in the wrapped action space. The first samples all action dimensions uniformly from $[-1, 1]$. The second is a low-actuation policy that keeps the plant near a conservative operating point. The third is a demand-tracking controller that distributes electricity demand across the units and sets steam flow to follow steam demand. We also add Gaussian noise to the demand-tracking actions to broaden the dataset around realistic dispatch behavior. During data collection, we vary `renewables_magnitude` between 0 MW and 300 MW and vary `forecast_noise_std` in a subset of rollouts. The first changes the demand pattern, while the second makes the forecast information less reliable.

For tokenization, plant-wide variables are treated as global context. These include time of day and the forecast blocks for weather, demand, prices, and fuel cost. Unit-specific information is taken from the previous-action component of the state. Each gas turbine token contains its previous power output, steam flow, evaporative cooler switch, and power-augmentation switch. The steam turbine token contains its previous power output, condenser steam flow, and bay count. This keeps the tokenization aligned with the physical units of the plant.

Because the wrapper represents all actions in the same range, we apply uniform binning to each of the 15 action dimensions. The action head uses 15 parallel classification heads, one per action dimension. At inference time, the predicted bin centers are passed through the wrapper, which clips, thresholds, and rounds them before applying the native plant action.

The auxiliary state head predicts next-step plant outputs that directly affect the reward: fuel cost, net electric power output, and net steam export. These targets are useful because they supervise the model on the short-term input–output behavior of the ONNX plant model. For `CogenEnv`, we also standardize return-to-go values over the training set before using them for value-head supervision and inference-time conditioning. This is needed because the reward components have very different scales: fuel and ramping costs are usually moderate, while supply–demand and constraint violations can create very large one-step penalties.

B.5 Summary of Pretraining Configurations

Table 5 summarizes the main pretraining settings. Unless otherwise stated, environment parameters and penalty weights follow the corresponding SustainGym implementation. We only modify representations or normalization when needed to make the Decision Transformer interface stable across training and deployment.

Appendix C Pretraining Dataset and Filter Training

C.1 Pretraining Dataset and Controllability Filter Training

This appendix describes the offline data used for Decision Transformer pretraining and controllability-filter training. The two datasets have different roles. The pretraining dataset teaches the frozen controller useful behavior before deployment. The filter dataset teaches DARPAN which parts of a

Table 5: Pretraining configurations for the two SustainGym environments.

	`BuildingEnv`	`CogenEnv`
Episode length	288 steps, 5-minute control	96 steps, 15-minute control
State dimension	10	44
Action dimension	6	15
Action range	$[-1, 1]^6$	wrapped to $[-1, 1]^{15}$
Action discretization	21 bins per dimension	uniform bins per dimension
Main behavior policies	random, thermostat, proportional	random, low actuation, demand tracking
State prediction target	next zone temperatures	next fuel, power, and steam outputs
Main pretraining shifts	weather distribution	wind profile and forecast noise

prediction residual are likely to be action-sensitive. The filter is trained separately, frozen, and then used only during deployment-time adapter updates.

Unless otherwise stated, we follow the default SustainGym settings and the standard Decision Transformer setup. Environment-specific changes are made only when required by the simulator interface or for stable training.

C.1.1 BuildingEnv Pretraining Dataset

For `BuildingEnv`, we use a fixed `OfficeSmall` configuration in the `Hot_Dry` climate setting. Each episode contains $T = 288$ steps at 5-minute resolution. Keeping the building configuration fixed ensures that all trajectories share the same state and action dimensions.

The dataset is collected using a mixture of simple behavior policies. These include an all-zero policy, proportional thermostat controllers with different gains, an energy-saving thermostat variant, low-magnitude random exploration, bounded random exploration, and a noisy thermostat controller. The thermostat policies use the zone-temperature error,

$$a_i(t) = \mathrm{clip}\big(K_p(T_i^{\mathrm{target}} - T_i(t)), -1, 1\big),$$

while the exploratory policies broaden the action distribution around low and moderate actuation regimes. This policy mixture gives the Decision Transformer both structured control behavior and exploratory trajectories.

To expose the pretrained controller to mild deployment variation, part of the dataset is collected with a fault wrapper. The wrapper samples from the same broad fault families used later in evaluation, including actuator weakening, sensor drift, window-opening disturbances, and compound cases. The main deployment challenge is still left for the online adaptation stage.

Table 6: `BuildingEnv` pretraining data summary.

Setting	Value
Environment	`BuildingEnv`, `OfficeSmall`
Episode length	288 steps
Control interval	5 minutes
Number of behavior policies	8
Training episodes	3000
Validation episodes	300
Held-out episodes	600
Fault-wrapped trajectories	included during data collection

During Decision Transformer training, we apply trajectory weighting only to the action loss. The weight gives slightly higher influence to stronger control trajectories while retaining lower-return exploratory behavior. The state-prediction and value losses are not reweighted.

C.1.2 CogenEnv Pretraining Dataset

For `CogenEnv`, trajectories are collected through the flat-action wrapper. Each behavior policy outputs a 15-dimensional action in $[-1, 1]$, and the wrapper maps this vector to the native mixed action space.

We use three behavior policies. The first samples random dispatch actions. The second keeps the plant near a low-actuation operating regime. The third is a demand-tracking heuristic that reads the current electricity and steam demand fields and distributes dispatch across the turbines according to their feasible ranges. Small Gaussian noise is added to the demand-tracking actions so that repeated rollouts are not identical.

Trajectories are split by calendar day rather than by random transition sampling. This prevents the same day from appearing in both training and validation. Harder distribution shifts are introduced later through the evaluation fault catalog rather than through a separate out-of-distribution pretraining split.

Table 7: `CogenEnv` pretraining data summary.

Setting	Value
Environment	`CogenEnv`, flat-action wrapper
Episode length	96 steps
Control interval	15 minutes
Number of behavior policies	3
Training trajectories	4000
Validation trajectories	400
Action dimension	15
Validation split	disjoint calendar days

The behavior policies are designed to cover a range of operating regimes, from broad exploration to more structured demand-following behavior. This gives the pretrained controller a useful base policy before deployment-time adaptation begins.

C.1.3 Controllability Filter: Purpose

The controllability filter decides which dimensions of a prediction residual should be used to train the adapter. A residual caused by weak actuation should be kept because changing the action can help correct it. A residual caused by weather, noise, or an external demand shift should usually be suppressed because the adapter should not learn to chase it.

The filter is trained offline and then frozen. At deployment time, it does not choose actions, veto actions, or trigger a fallback policy. It only produces a mask over residual dimensions. The selected adapter is then updated using the masked residual signal.

C.1.4 Filter Training for BuildingEnv

For `BuildingEnv`, the filter input is the concatenation of the current state, current action, and one-step predictor residual. The output is a sigmoid mask with one entry per state dimension.

The filter is a small MLP with hidden size 128, LayerNorm, GELU activations, and a sigmoid output head. Inputs are normalized using statistics computed from the training data.

Training labels are built from simulator counterfactuals. For each sampled transition, we compare two controlled changes from the same simulator state. One changes the action, and the other changes exogenous conditions such as outdoor temperature or occupancy-related heat gain. These comparisons produce action-sensitivity and exogenous-sensitivity scores for each state dimension. The filter is trained to keep dimensions where action changes have stronger influence and suppress dimensions dominated by exogenous changes.

The filter objective is

$$\mathcal{L}_{\text{filter}}^{\text{train}} = \lambda_{\text{sens}}\mathcal{L}_{\text{sens}} + \lambda_{\text{stab}}\mathcal{L}_{\text{stab}} + \lambda_{\text{neutral}}\mathcal{L}_{\text{neutral}}. \tag{4}$$

Here, $\mathcal{L}_{\text{sens}}$ encourages high mask values for action-sensitive dimensions, $\mathcal{L}_{\text{stab}}$ encourages low mask values for exogenous-sensitive dimensions, and $\mathcal{L}_{\text{neutral}}$ weakly prevents the mask from becoming overconfident too early.

C.1.5 Filter Training for CogenEnv

The `CogenEnv` filter has the same role, but its labels are obtained from fault trajectories rather than simulator counterfactuals. This is because `CogenEnv` does not provide the same cheap counterfactual reset interface as `BuildingEnv`.

The input is the concatenation of the state, action, and predictor residual over the plant target subset. The output is a sigmoid mask over the target dimensions. The architecture is the same small MLP used for `BuildingEnv`, with input and output dimensions changed for `CogenEnv`.

Fault trajectories are drawn from a catalog containing turbine degradation, ambient sensor bias, wind-shift demand perturbations, and compound cases. Each fault type is mapped to the plant-output dimensions it can physically affect. Healthy trajectories receive an all-zero label. The filter is trained with positive-weighted binary cross-entropy to handle class imbalance across target dimensions.

C.1.6 Deployment Use of the Filter

During evaluation, the filter is loaded from its checkpoint and kept frozen. At each adapter-update step, DARPAN computes the frozen predictor residual, passes it through the filter, and obtains a detached mask. Only the selected adapter is updated using the masked residual loss. The Decision Transformer, predictor, filter, and all non-selected adapters remain fixed.

For ablations, we replace the learned mask with an all-ones mask. This tests how much of DARPAN's behavior comes from controllability-aware residual filtering rather than from the adapter alone.

Appendix D Implementation Details

D.1 Implementation Details

This section summarizes the implementation used to pretrain the Decision Transformer backbone on `BuildingEnv` and `CogenEnv`. Both environments use the same architecture. They differ only in input/output dimensions, state-prediction targets, and a small number of training settings listed in Table 8. We follow the standard Decision Transformer design as closely as possible and make environment-specific changes only when required by the simulator interface or by empirical stability. All experiments are run on a single workstation with an NVIDIA GeForce RTX 5090 GPU (32 GB VRAM), an Intel Core Ultra 9 285K CPU, and 64 GB of RAM, running Ubuntu 22.04.

D.1.1 Model Architecture

We use a causal transformer decoder with a shared backbone and three prediction heads. Each timestep is represented by return-to-go, state, and action tokens, interleaved into a sequence of length $3K$ with context length $K = 20$. The backbone has two transformer blocks, hidden size $d_{\text{model}} = 64$, four attention heads, pre-LayerNorm, residual connections, GELU activations, and dropout 0.1.

States, actions, and returns-to-go are embedded with separate linear layers and a learned timestep embedding. We also include a small FiLM gate that allows the return-to-go representation to modulate the state-token representation before prediction. The gate is initialized as the identity map, so the model starts as a standard Decision Transformer and learns this conditioning only if useful.

The model has three heads. The action head predicts the normalized action from the state-token hidden state, rather than from the action-token hidden state, to avoid leaking the current action into the prediction. The state head predicts short-horizon future state targets at horizons $\{1, 3, 6, 12\}$. The value head predicts the normalized return-to-go. The backbone, embedding layers, and FiLM gate are shared across both environments; only d_s, d_a, d_{tgt}, and the timestep-table size change.

D.1.2 Training Procedure

We train with AdamW using learning rate 3×10^{-4} and weight decay 10^{-4}. Both environments use cosine learning-rate decay. `BuildingEnv` uses no warmup, while `CogenEnv` uses a 500-step warmup because its reward scale and target dynamics are less smooth at the start of training.

The total loss is

$$\mathcal{L}_{\text{total}} = \lambda_{\text{action}}\mathcal{L}_{\text{action}} + \lambda_{\text{state}}\mathcal{L}_{\text{state}} + \lambda_{\text{value}}\mathcal{L}_{\text{value}}. \tag{5}$$

The action loss matches the recorded normalized action, the state loss trains the multi-horizon prediction head, and the value loss trains return-to-go prediction. For `BuildingEnv`, we additionally apply a per-episode weight to the action loss so that high-return comfort-control trajectories have slightly more influence. This weight only affects the action loss; the state and value losses are not reweighted.

D.1.3 Hyperparameters

Table 8 lists the main implementation settings. Dataset construction and split details are given separately in Appendix C.1.

Table 8: Main implementation settings for Decision Transformer pretraining.

Parameter	`BuildingEnv`	`CogenEnv`
Architecture		
State dimension d_s	10	44
Action dimension d_a	6	15
State target dimension d_{tgt}	10	13
Hidden size d_{model}	64	64
Transformer layers	2	2
Attention heads	4	4
Dropout	0.1	0.1
Context length K	20	20
Timestep table size $L_{\max}$	4096	512
State head horizons	$\{1, 3, 6, 12\}$	$\{1, 3, 6, 12\}$
FiLM hidden size	64	64
Optimization		
Optimizer	AdamW	AdamW
Learning rate	3×10^{-4}	3×10^{-4}
Weight decay	10^{-4}	10^{-4}
LR schedule	cosine, no warmup	cosine, 500-step warmup
LR floor	10^{-6}	0
Gradient clip	1.0	1.0
Loss		
$\lambda_{\mathrm{action}}$	5.0	1.0
λ_{state}	0.5	1.0
λ_{value}	0.2	10^{-3}
Action-loss weighting	enabled, $[0.10, 6.00]$	disabled
Training		
Batch size	1024	64
Maximum epochs	30	200
Early stopping	none	val $\mathcal{L}_{\mathrm{action}}$, patience 10
State normalization	per-dimension z-score	per-dimension z-score
Action normalization	per-dimension z-score	already in $[-1, 1]$
RTG normalization	per-dimension z-score	99th percentile $\lvert\mathrm{RTG}\rvert$ scale

Table 9 lists the deployment hyperparameters of DARPAN, which are shared across both environments. Each adapter has a single hidden layer of width 8, $\mathrm{Linear}(d_x, 8) \to \mathrm{ReLU} \to \mathrm{Linear}(8, d_u)$, so the two environments differ only in the adapter input and output dimensions.

Appendix E Drift Protocol and Evaluation

E.1 Drift Protocol and Evaluation

This appendix describes how we evaluate DARPAN after pretraining. The goal is to test whether a frozen pretrained controller can adapt during a long deployment, where the physical system changes over time. Each controller is evaluated in a continuous rollout. The environment is reset at the start of each day, but DARPAN's adapter library is not reset. Adapters, signatures, optimizers, replay buffers, and usage history all persist across the full evaluation. This is important because a deployment method should remember useful corrections when a similar fault returns later.

Table 9: Deployment hyperparameters of DARPAN.

Symbol	Meaning	`BuildingEnv`	`CogenEnv`
H	Signature history length	12	12
c_{filter}	Controllability filter threshold	0.50	0.50
τ	Cosine similarity threshold (adapter matching)	0.91	0.91
K_{create}	Persistence step count (adapter creation rule)	24	24
α	Corrective strength	0.8	0.8
λ	Correction-size penalty	10^{-3}	10^{-3}
$M_{\max}$	Adapter library capacity	40	40
—	Adapter hidden width	8	8
d_x	Adapter input dimension	10	44
d_u	Adapter output dimension	6	15
—	Adapter parameter count	142	495

The main evaluation in the paper uses a 60 day continual drift protocol. This protocol is the source of the main evaluation figure. The figure shows two things together: the physical recovery over time and the active adapter selected by DARPAN. The recovery curve tells us whether adaptation improves control, while the adapter trace tells us whether DARPAN keeps creating new adapters or reuses earlier ones.

E.1.1 Why the protocol uses several drift types

A single fault is not enough to test DARPAN. The central claim of the paper is not only that the controller can adapt, but that it adapts for the right reason. For this reason, the protocol includes controllable drift, sensor drift, delay, stuck actuators, and compound faults.

At each step, the frozen predictor gives an expected next state $\hat{x}_{t+1}$ and the environment gives the observed next state x_{t+1}. The raw residual is

$$e_{t+1} = x_{t+1} - \hat{x}_{t+1}.$$

DARPAN then applies the controllability mask m_{t+1} and forms the filtered residual

$$\tilde{e}_{t+1} = m_{t+1} \odot e_{t+1}.$$

The adapter library is routed using a short history of these filtered residuals:

$$z_t = \frac{1}{H} \sum_{i=0}^{H-1} \tilde{e}_{t-i}.$$

This vector z_t is the drift signature. It is not meant to identify a fault label. It is meant to summarize the recent action sensitive direction of the mismatch.

Different drift types create different residual directions. A weak actuator creates a residual in the zone temperatures that should respond to stronger actions. A sensor bias creates a persistent offset in the measured state, but it does not always mean the physical room has changed in the same way. An actuation delay creates a time shifted residual, where the action may be correct but its effect arrives late. A stuck actuator creates a stronger failure because part of the action space no longer behaves normally. Compound faults test whether DARPAN can reuse corrections from simpler faults instead of treating every new combination as unrelated.

E.1.2 60 day BuildingEnv drift schedule

The BuildingEnv evaluation is a single 60 day rollout. Each day is one episode of $T = 288$ five minute steps. The schedule begins with clean operation and then moves through repeated and increasingly difficult drift conditions. Some faults return later in the rollout so that we can test adapter reuse.

This schedule is designed to answer four questions. First, the weak actuator segments test whether DARPAN can recover when the same action produces a weaker physical effect. Second, the sensor bias segments test whether the filter prevents the adapter from overreacting to persistent measurement shifts. Third, the delay and stuck actuator segments test harder physical failures where the residual

Table 10: BuildingEnv 60 day continual drift schedule used for the main evaluation figure.

Days	Segment	Fault family	Purpose
1 to 5	Clean baseline	none	measure the frozen controller before drift
6 to 10	Weak actuator, mild	controllable	test first correctable drift
11 to 15	Sensor bias, warmer	sensor drift	test persistent measurement shift
16 to 20	Actuation delay, short	delay	test delayed action response
21 to 25	Weak actuator, medium	controllable	test stronger under response
26 to 30	Sensor bias, cooler	sensor drift	test opposite bias direction
31 to 35	Stuck actuator, moderate	actuator fault	test partial loss of control authority
36 to 40	Actuation delay, long	delay	test stronger temporal mismatch
41 to 45	Weak actuator, severe	controllable	test severe correctable drift
46 to 50	Sensor bias, warmer repeat	sensor drift	test recurrence of a known pattern
51 to 55	Stuck actuator, strong	actuator fault	test strong loss of actuator response
56 to 60	Compound faults	compound	test reuse under combined drift

is not just larger, but has a different temporal or directional structure. Fourth, the final compound segment tests whether the adapter library has learned reusable correction patterns. If DARPAN simply memorized fault names, it would need a new adapter for each compound case. If it learned useful residual directions, it should reuse adapters from earlier isolated faults.

The main evaluation figure visualizes this behavior. The top panel reports cumulative comfort penalty reduction relative to the frozen Decision Transformer. The bottom panel reports the active adapter index over the same 60 day rollout. A good result should show positive recovery without constant adapter creation. Long flat regions in the adapter trace indicate reuse, while occasional changes indicate that DARPAN creates a new adapter only when the filtered residual signature remains different for several steps.

E.1.3 CogenEnv drift schedule

CogenEnv is evaluated with a shorter 14 day protocol because each day represents a full plant dispatch episode with a different calendar day and demand profile. Each episode has $T = 96$ fifteen minute steps. The schedule includes clean days, turbine degradation, sensor bias, wind demand shifts, and compound faults.

Table 11: CogenEnv 14 day drift schedule.

Day	Fault ID	Fault family	Description
0 to 1	none	clean	clean baseline
2	F1 mild	controllable	GT1 power degradation, severity 0.85
3	F1 severe	controllable	GT1 power degradation, severity 0.65
4	F3	sensor drift	ambient temperature sensor bias $+5.0^{\circ}\mathrm{F}$
5	F1 severe	controllable	repeated GT1 degradation
6	none	clean	clean recovery
7	F4	sensor drift	ambient pressure sensor drift
8	F6	controllable	HRSG1 fuel response degradation
9	F5	demand shift	wind shift on demand, -100 MW
10	F2	controllable	GT2 and GT3 multi turbine degradation
11	F7	compound	F1 severe with wind shift
12	F8	compound	multi turbine degradation with sensor bias
13	F1 severe	controllable	final reuse test

The CogenEnv schedule tests the same idea in a different physical system. Turbine degradation is controllable because dispatch changes can partly compensate for reduced output. Sensor bias and wind shifts are more exogenous because they change the information or demand pattern seen by the controller. Compound faults test whether the same residual filtering and adapter routing idea remains useful when multiple changes occur together.

E.1.4 Deployment adapter loop

At deployment, the Decision Transformer, predictor, and controllability filter are frozen. Only the active adapter is updated. At each step, the frozen Decision Transformer predicts a base action a_t^{DT}. The selected adapter predicts a small correction δa_t, and the environment receives

$$a_t = \mathrm{clip}(a_t^{\mathrm{DT}} + \delta a_t, -1, 1).$$

The adapter is a small independent MLP. It does not modify the Decision Transformer weights. A newly created adapter is initialized so that its first correction is close to zero. This means DARPAN starts from the frozen controller and only changes the action after the adapter has evidence that a correction is useful.

The adapter library grows online. At every step, DARPAN compares the current drift signature z_t with the stored signatures in the library using cosine similarity. If a similar signature exists, DARPAN reuses the matching adapter. If the signature remains different for several consecutive steps, DARPAN creates a new adapter. The library has a fixed capacity, so it cannot grow without bound. When needed, similar adapters are merged to free space.

E.1.5 Metrics

We report metrics that separate physical recovery from adaptation behavior. Return alone is not enough because a controller can sometimes improve reward while still behaving poorly under physical drift.

For BuildingEnv, the main physical metric is comfort recovery. It measures how much of the comfort penalty gap between the frozen Decision Transformer and the clean reference behavior is closed by a method. We also report recovery over time in the 60 day graph, so the reader can see when adaptation helps and whether it remains stable.

We also track adapter behavior. The active adapter index shows whether the method is reusing old adapters or creating new ones. The library size measures memory growth. The adapter correction norm $\|\delta a_t\|$ measures how strongly the controller is changing the frozen action. The filter mask mean measures how much of the residual is being treated as useful for adaptation.

Three drift specific metrics are especially important. Recovery time measures how quickly the controller improves after a new correctable fault appears. False adaptation rate measures how often the adapter changes its action during mostly exogenous disturbance. Reuse rate measures how often a later repeated fault selects an adapter that already existed before that segment began. Together, these metrics test the main behavior DARPAN is designed for: adapt to controllable drift, avoid chasing exogenous noise, and reuse useful corrections.

E.1.6 Baselines

We compare DARPAN against frozen and adaptive baselines. The frozen Decision Transformer tests what happens with no deployment adaptation. Online Decision Transformer fine tuning tests whether updating the full model online is helpful or unstable. Online head and LoRA baselines test parameter efficient adaptation without DARPAN's residual filtering and library routing. HyperDT tests a closely related adapter based Decision Transformer baseline. Online PPO tests whether learning from scratch online is competitive in this slow physical setting. Online PPO learns from scratch with a learning rate of 3×10^{-4}, batch size 64, 10 optimization epochs, and $\gamma = 0.99$. Full online DT is initialized from the same pretrained Decision Transformer and updates all of its parameters using AdamW with a learning rate of 10^{-5}. The negative recovery of these two methods indicates failure to recover within the fixed deployment budget, rather than evidence that they cannot eventually adapt.

All transformer based methods start from the same pretrained Decision Transformer. This keeps the comparison focused on the deployment adaptation mechanism. DARPAN differs from the other adaptive methods because it does not update the backbone and does not adapt from the raw residual. It first filters the residual, builds a temporal drift signature, and then updates only the selected adapter.

Appendix F Justification & Walkthrough of DARPAN

This appendix gives a concrete view of how DARPAN runs during deployment. Section 3 describes the method formally; here we walk through one step, connect each component to the evaluation, and summarize the settings where the method is expected to help. The goal is to make the residual filtering and adapter reuse behavior easier to follow.

F.1 Notation Summary

Table 12 summarizes the main symbols used in the method.

Table 12: Main symbols used in DARPAN.

Symbol	Meaning
x_t	Observed state at time t.
a_t^{base}	Action proposed by the frozen Decision Transformer.
Δa_t	Adapter correction added to the base action.
a_t	Final deployed action, $\text{clip}(a_t^{\text{base}} + \Delta a_t)$.
$\hat{x}_{t+1}$	Frozen-predictor estimate of the next state.
e_{t+1}	Raw residual, $x_{t+1} - \hat{x}_{t+1}$.
p_{t+1}	Per-dimension controllability score from the filter.
m_{t+1}	Binary mask produced from p_{t+1}.
$\tilde{e}_{t+1}$	Filtered residual, $m_{t+1} \odot e_{t+1}$.
z_t	Drift signature, computed from a short history of filtered residuals.
$\mathcal{A}_t = \{(z_j, A_j)\}_{j=1}^{M_t}$	Adapter library at time t.
H	History length for the drift signature.
α	Strength of the corrective target.
λ	Weight on the adapter correction size penalty.
$P_{\text{frozen}}, P_{\text{ref}}, P_{\text{method}}$	Physical penalty for Frozen DT, clean reference, and the evaluated method.

F.2 One Deployment Step

We illustrate one step of DARPAN on `BuildingEnv`. For simplicity, we show only one zone temperature and the outdoor-temperature dimension. All other state dimensions are processed in the same way.

At time t, suppose zone 1 is at $19.5°$C and its target temperature is $21.0°$C. The frozen Decision Transformer reads the recent context window and proposes a base heating action, $a_t^{\text{base}}[1] = 0.60$. The current drift signature z_t is then compared with the adapter library. If the closest stored signature corresponds to a weak-actuator pattern, the matching adapter is selected. For example, if it predicts $\Delta a_t[1] = 0.15$, the deployed action becomes $a_t[1] = \text{clip}(0.60 + 0.15) = 0.75$.

Next, the frozen predictor estimates what should happen under the base action. Assume it predicts $\hat{T}_1(t+1) = 20.3°$C and $\hat{T}_E(t+1) = 8.0°$C. After the action is applied, the environment returns $T_1(t+1) = 19.9°$C and $T_E(t+1) = 4.0°$C. The raw residual is therefore

$$e_{t+1} = \begin{bmatrix} T_1(t+1) - \hat{T}_1(t+1) \\ T_E(t+1) - \hat{T}_E(t+1) \end{bmatrix} = \begin{bmatrix} -0.4 \\ -4.0 \end{bmatrix}.$$

The outdoor-temperature error is much larger, but it is not necessarily useful for control. A large residual only says that the predictor was wrong; it does not say whether the controller can correct the cause.

The controllability filter resolves this ambiguity. Suppose it outputs $p_{t+1}[\text{zone } 1] = 0.85$ and $p_{t+1}[\text{outdoor}] = 0.10$. With threshold $c_{\text{filter}} = 0.5$, the mask is $m_{t+1} = (1, 0)^\top$, and the filtered residual becomes

$$\tilde{e}_{t+1} = m_{t+1} \odot e_{t+1} = (-0.4, 0)^\top.$$

The outdoor residual is removed before the adapter is updated. This is the key behavior DARPAN is designed to produce: the method does not adapt to the largest residual, but to the residual dimensions that appear useful for control.

The filtered residual is then added to the running window used to form the next drift signature. If the zone-temperature residual remains negative over several steps, the signature stays close to the weak-actuator pattern and the same adapter is reused. If the residual appears only once, the history window reduces its influence and prevents an immediate adapter switch.

Finally, the transition is stored in the selected adapter's replay buffer. When an update is triggered, DARPAN samples from this buffer and updates only the selected adapter. The Decision Transformer, predictor, controllability filter, and all other adapters remain frozen.

F.3 How Each Component Is Tested

Table 13 connects each part of DARPAN to the experiment that tests it. The ablations are designed so that removing one component produces a different failure mode.

Table 13: DARPAN components and the evidence supporting them.

Component	What can fail without it	Evidence
Frozen DT backbone	Online updates can overwrite useful pretrained behavior.	Full online DT gives negative recovery on BuildingEnv, showing that updating the whole model can hurt under drift.
Controllability filter	Adapters may learn from weather, occupancy, or noise residuals.	The raw-residual variant creates many more false adapters and has lower recovery than full DARPAN.
Drift signature	Single noisy residuals can cause unstable adapter switches.	The single-step variant grows a larger library and has a lower reuse rate.
Conservative creation rule	The library can fill with near-duplicate adapters.	Naive expansion keeps reasonable recovery but creates many adapters and has low reuse.
Adapter library	A single adapter can forget earlier corrections when new drift appears.	The one-global-adapter variant has the lowest recovery among the ablations.
Intervention-trained filter	The method may not separate action-sensitive and outside-driven residuals.	The disturbance-only experiment shows a large reduction in false adapter creation.

The table also explains why the ablation results should be read together. The filter, residual history, creation rule, and adapter library do not solve the same problem. The filter decides what residual information is safe to use. The history makes that information stable over time. The creation rule prevents the library from growing too quickly. The library stores corrections so they can be reused when similar drift returns.

F.4 Why the Baselines Behave Differently

The baselines fail for different reasons. This section gives an intuitive explanation of those differences.

Online PPO. Online PPO learns from scratch using the environment reward. This is difficult in slow physical systems because actions have delayed effects and early exploration can create large comfort or operating penalties. Within the evaluation horizon, PPO pays the cost of exploration before it can learn a useful control policy.

Full online DT. Full online DT starts from the pretrained controller but updates all of its weights during deployment. This gives it a large number of trainable parameters, but also makes it easier to overwrite useful pretrained behavior. Since the online buffer contains both correctable drift and outside disturbances, full fine-tuning can move the controller toward recent but misleading transitions.

Online-head DT. Online-head DT is safer because it updates only the output head. This preserves the backbone and gives positive recovery. Its limitation is that one head must represent all drift patterns. A correction for one type of drift can interfere with a correction for another.

LoRA-DT. LoRA-DT updates low-rank components inside the transformer. This is parameter-efficient, but the update still changes how the backbone processes the full context. In physical control,

this can mix useful action-related residuals with weather, occupancy, or other outside factors. The result is limited recovery in our setting.

Hyper-DT. Hyper-DT is the closest adapter-based baseline. It generates adaptation parameters from the current context, so it can respond to deployment changes. The difference is that it does not select adapters from a library of filtered residual signatures. As a result, recurring drift patterns must be handled again rather than explicitly reused.

DARPAN. DARPAN combines four safeguards. The backbone stays frozen, the residual is filtered before learning, the signature is averaged over time, and adapters are stored for reuse. This is why DARPAN can recover under controllable drift while using fewer active parameters.

F.5 Hyperparameter Behavior

DARPAN is controlled mainly by the filter threshold c_{filter}, the history length H, the corrective-target strength α, and the library capacity $M_{\max}$. We summarize their roles below.

Filter threshold c_{filter}. If the threshold is too low, the filter keeps too many residual dimensions and the adapter may learn from outside disturbances. If it is too high, the filter removes useful residual dimensions and adaptation becomes weak. A useful diagnostic is the average mask value during clean, controllable-drift, and disturbance-only phases.

History length H. If H is too small, adapter selection becomes sensitive to single-step noise. If H is too large, DARPAN reacts slowly when a new drift pattern appears. The goal is to make the signature long enough to smooth noise but short enough to detect real change.

Corrective strength α. If α is too small, the corrective target stays too close to the frozen prediction and the adapter learns little. If it is too large, the adapter can overreact to residuals that are only partly action-sensitive. We therefore use conservative values and keep a penalty on the size of Δa_t.

Library capacity $M_{\max}$. If the library is too small, different drift patterns may be merged too early. If it is too large, the method can keep many near-duplicate adapters. The capacity should reflect the expected number of distinct residual patterns, not the number of named fault labels.

F.6 When DARPAN Is Expected to Help

DARPAN is meant for slow physical systems where the controller's action has a measurable short-term effect, but the deployed system can drift over time. This includes building control, thermal systems, energy dispatch, and related process-control settings.

The method is most useful when controllable drift creates residuals that persist for several steps, and when outside disturbances create residuals that can be separated by the filter. It is also useful when similar drift patterns may return later, because the adapter library can reuse earlier corrections.

DARPAN may be less effective when short-term prediction is not reliable. For example, in fast systems with contact, perception failures, or other agents, the next-state residual may be too noisy to guide an adapter safely. The method may also struggle when a controllable fault and an outside disturbance produce nearly identical residuals in the observed state. In those cases, DARPAN would need richer observations, longer-horizon prediction, uncertainty estimates, or additional safety checks.

F.7 Validity and Limitations

The main limitation is that the controllability filter is trained in simulation. For `BuildingEnv`, the filter uses simulator counterfactuals; for `CogenEnv`, it uses labeled fault trajectories. This gives clean training signals, but real deployments may contain residual patterns that are not covered by these training sources. A real system would likely need logged interventions, system-identification data, or some form of recalibration before deployment.

The BuildingEnv experiments also use a fixed building configuration. This lets us study deployment drift without changing the input and output dimensions of the Decision Transformer. It does not test generalization across buildings with different numbers of zones.

The drift catalog is limited. The evaluation includes actuator weakening, sensor bias, actuation delay, stuck actuators, weather or occupancy shifts, and compound faults. Real systems may contain additional long-tail faults such as valve sticking, filter clogging, humidity effects, maintenance changes, or unmodeled airflow. These may create more residual patterns and may require more adapters or stronger adapter consolidation.

The evaluation horizon is also finite. A 60-day protocol is long for an ML benchmark, but real buildings operate across seasons. The current protocol tests recurring and compound drift, but not full-year seasonal change.

These limitations do not remove the main finding. They define the setting where the claim is currently supported. Within that setting, the experiments show that filtered residuals are a more reliable update signal than raw residuals for adapting pretrained physical controllers.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction accurately describe the paper's main contributions: DARPAN, a frozen-backbone adaptation method that uses controllability-aware residual filtering and residual-based adapter reuse for pretrained physical control. The claims are scoped to deployment-time drift in simulated building-control and power-plant dispatch benchmarks and are supported by the experimental comparisons and ablation studies reported in the paper.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a Limitations section discussing the scope of the evaluation and the main assumptions behind the method. In particular, we discuss that DARPAN is evaluated on simulated building-control and power-plant dispatch benchmarks with controlled drift protocols, depends on the frozen predictor and controllability filter, and requires further validation in longer deployments and real physical-control systems.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present formal theoretical results, theorems, or proofs. The equations in the paper define the DARPAN method, residual filtering procedure, adapter routing, and training objectives, and the claims are evaluated empirically.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the benchmark environment, offline pretraining data, deployment drift protocol, drift types, evaluated baselines, model components, adapter update procedure, and evaluation metrics. Additional implementation details, including data generation, model training, and hyperparameter settings, are provided in the appendix to support reproduction of the main experimental results.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.

(c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer:[Yes]

   Justification: We provide code and configuration files in the supplemental material, including scripts for data generation, model training, drift evaluation, and baseline comparison. The experiments use the publicly available SustainGym benchmark, and the supplemental material includes instructions for reproducing the main results.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specifies the SustainGym benchmark setting, offline pretraining data, train/test separation, drift protocol, drift severities, baselines, evaluation metrics, and adapter update procedure. Model architecture, optimizer settings, context length, batch size, and other hyperparameters are reported in the experimental setup and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We report results over multiple random seeds and include the standard deviation for the main experimental metrics. The paper states that the error bars capture variability across repeated runs with different seeds under the same drift protocol.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: The paper reports the hardware used for training and evaluation, including GPU type, CPU, and memory, in Appendix D.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

Answer: [Yes]

Justification: The research uses benchmark simulation environments and does not involve human subjects, private data, sensitive personal data, or deceptive data collection. The research follows the NeurIPS Code of Ethics.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses potential positive impacts for energy-efficient and adaptive building control, as well as possible risks from unsafe online adaptation, comfort violations, and over-reliance on simulation-trained controllers. The discussion also notes that real deployments should include safety constraints, monitoring, and human/operator oversight.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk models, scraped datasets, or dual-use generative systems. The experiments use benchmark physical-control environments and do not involve private, sensitive, or scraped human data.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.

- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper uses the publicly available SustainGym benchmark and cites the original SustainGym paper. We state the benchmark version and license information in the experimental setup or appendix, and we use the benchmark only according to its terms. Other existing methods used as baselines are cited to their original papers.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The supplemental material includes code and configuration files for reproducing the main experiments. The released materials are documented with setup instructions, experiment scripts, and descriptions of the benchmark environments and drift protocols.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject experiments, or participant data collection. The experiments are conducted using benchmark physical-control environments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing, human-subject experiments, or participant data collection. The experiments are conducted using benchmark physical-control environments, so IRB approval or equivalent human-subject review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: The core method development in this paper does not involve LLMs as an important, original, or non-standard component. Any use of LLMs was limited to writing, editing, or formatting assistance and did not affect the methodology, experiments, scientific claims, or originality of the research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.