STABLEBFM: EXTENDING BEHAVIOR FOUNDATION MODELS TO ROBUST BEHAVIOR

Anonymous authors
Paper under double-blind review

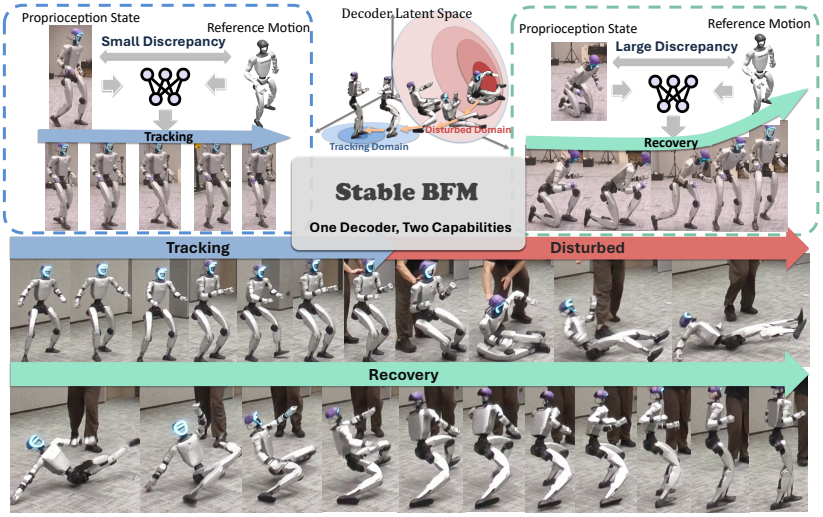

Figure 1: **One decoder, two behaviors.** The same state–reference decoder tracks the reference at small discrepancy and closes the loop back into the tracking region after pushes and falls (top); real-robot push–fall and get-up sequence (bottom).

ABSTRACT

Pretrained behavior foundation models (BFMs) provide a reusable foundation for whole-body humanoid control by conditioning on large-scale reference motions and decoding actions, enabling high-fidelity tracking of diverse behaviors. However, real-world deployment also requires reliable closed-loop recovery from off-nominal states such as falls. How to extend the capabilities of a pretrained BFM to recovery while preserving its existing tracking ability and keeping training overhead low remains a key challenge. Given the current robot state and a reference motion, we define the state–reference discrepancy as the deviation between the current state and the target state implied by the reference. Standard tracking pretraining mainly constrains behavior in the small-discrepancy regime, whereas fall recovery requires the decoder to start from large-discrepancy states, reduce the discrepancy through closed-loop control, and return to the nominal tracking region. From this perspective, we formulate recovery as extending the effective closed-loop control range of a pretrained BFM, and unify tracking and recovery as the same state–reference conditional decoding problem across discrepancy regimes. To this end, we treat recovery as a capability extension of a pretrained motion-tracking BFM and perform simple, low-overhead post-training without adding model parameters. We retain the original tracking proximal policy optimization (PPO) objective, allocate a subset of parallel environments to recovery, and apply DAgger-style expert supervision to states visited by the learner, thereby extending recovery supervision to the large-discrepancy regime. We further design corresponding recovery-injection mechanisms for two decoder parameterizations. Experiments show that the proposed framework significantly improves recovery while maintaining tracking performance, and supports deployment on a real humanoid robot.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

1 INTRODUCTION

Humanoid robots are widely regarded as general-purpose embodied platforms for performing tasks in human environments. In recent years, Behavior Foundation Models (BFMs) (Zeng et al., 2025; 2026; Luo et al., 2026) have emerged as a unified paradigm for whole-body humanoid control. Pretrained on large-scale retargeted human motion data, these models learn to decode control actions from the current state and reference motion, enabling general tracking across diverse behaviors and providing a reusable foundation for whole-body control.

However, during BFM tracking pretraining, rollouts are typically terminated early when the robot deviates substantially from the reference and continued tracking is deemed unsuccessful, so pretrained decoders mainly learn control in nominal tracking states, while large-deviation states remain insufficiently covered. To characterize the deviation between the current robot state and the target state implied by the reference motion, we define the **state–reference discrepancy**. Small discrepancies correspond to nominal tracking states, whereas disturbances, accumulated errors, or falls may drive the robot into large-discrepancy states. This reveals a gap in the decoder's effective control range: pretraining covers the small-discrepancy regime, while recovery requires operating in the large-discrepancy regime and returning to tracking. Can a pretrained BFM extend its effective closed-loop control range from small- to large-discrepancy states through low-overhead post-training, without a separate recovery policy, while preserving its tracking capability? From this perspective, tracking and recovery are not two independent control problems, but two capabilities of the same state–reference conditional decoder in different discrepancy regimes. We therefore view fall recovery as an extension of the **effective closed-loop control range** of a pretrained BFM, rather than independent recovery-policy training.

Based on this perspective, we propose STABLEBFM, a **fixed-capacity recovery-injection framework** (Fig. 1). We retain the original tracking PPO objective while allocating a subset of parallel environments to recovery. Recovery rollouts start from diverse fallen configurations and stand-up references, are generated by the current learner, and are supervised by a recovery expert on learner-visited states, forming a DAgger-style data aggregation process (Ross et al., 2011). The two branches share the same decoder and are trained with PPO and behavior cloning (BC), respectively. For a Markov decoder, recovery supervision directly modifies the state–reference-to-action mapping, whereas for the context-conditioned decoder we introduce context management to handle historical information during recovery. Experiments show that the proposed framework substantially improves fall recovery while largely preserving tracking performance, and supports deployment on a real humanoid robot.

Our main contributions are:

- We view fall recovery as an extension of the effective closed-loop control range of pretrained BFMs, enabling a shared state–reference-conditioned decoder to cover both nominal tracking and large-discrepancy recovery.
- We develop STABLEBFM, a joint post-training framework that preserves the tracking PPO objective, applies DAgger-style expert supervision to learner-visited recovery states, and explores asynchronous PPO–BC updates.
- We instantiate the framework with both Markov and context-conditioned decoders and validate its real-world deployment on the Unitree G1.

2 RELATED WORK

2.1 FALL RECOVERY FOR HUMANOID ROBOTS

Fall recovery is commonly learned as a task-specific skill. HumanUP separates motion discovery under relaxed constraints from deployment-oriented trajectory tracking, whereas HoST uses multi-critic RL and curricula for posture-adaptive standing-up (He et al., 2025b; Huang et al., 2025). HiFAR progressively expands constrained recovery to full-body motion (Chen et al., 2025). Broader policies cover the complete fall sequence (Xu et al., 2025; Azulay et al., 2026), distill expert banks through a mode field (He et al., 2025a), or widen the training distribution (Zhang et al., 2025); UniReLo further learns terrain-aware recovery-to-locomotion transitions within one

policy (Xu et al., 2026). Closest to capability extension, BFM freezes its generative base and learns a residual decoder (Zeng et al., 2025), while continual learning appends skills while constraining forgetting (Ma et al., 2026a). In contrast, we achieve recovery through lightweight post-training of an existing motion-tracking BFM without structural expansion.

2.2 Behavior Foundation Models

The decoder we inject into comes from a line of humanoid behavior foundation models that cast whole-body control as decoding an action from the current state and a motion reference: a conditional variational decoder trained by masked online distillation (Zeng et al., 2025), scaled to a transformer over larger retargeted corpora (Zeng et al., 2026), recast so that tracking itself is the general objective (Luo et al., 2026), or made promptable by unsupervised reinforcement learning that embeds motions, goals, and rewards in one latent space (Li et al., 2025); guided diffusion and masked conditioning of a partially observed reference reach comparable versatility (Liao et al., 2026; Tessler et al., 2024). These models mainly emphasize high-fidelity tracking near the reference, while behavior under large state–reference discrepancies remains less explored. In contrast, we extend one such decoder to the large-discrepancy regime through lightweight post-training with DAgger-style expert supervision, without a separate recovery policy. A broader landscape of related work is discussed in Appendix A.

3 Method

We propose a recovery-injection post-training framework that injects fall-recovery supervision into a shared decoder while preserving pretrained tracking. Section 3.1 defines goal-conditioned tracking and state–reference discrepancy; Section 3.2 introduces two-branch tracking–recovery training with DAgger-style supervision; and Section 3.3 presents decoder-specific recovery-injection mechanisms.

3.1 Problem Formulation

3.1.1 Goal-Conditioned Motion Tracking

We formulate humanoid motion tracking as a goal-conditioned Markov decision process (MDP)

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{T}, R, \gamma \rangle. \tag{1}$$

At control step t, the robot is in a proprioceptive state $\boldsymbol{s}_t \in \mathcal{S}$, receives the goal $\boldsymbol{g}_t \in \mathcal{G}$ corresponding to the reference motion, and the decoder outputs a control action

$$\boldsymbol{a}_t \sim \pi_\theta(\cdot \mid \boldsymbol{s}_t, \boldsymbol{g}_t) \in \mathcal{A}, \tag{2}$$

where $\boldsymbol{a}_t$ denotes the target joint positions tracked by the joint-level PD controllers. The pretrained tracker maximizes the cumulative return with PPO (Schulman et al., 2017):

$$\max_\theta \ \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=1}^{T} \gamma^{t-1} R_t \right]. \tag{3}$$

Here R_t consists of tracking rewards and regularization terms. During recovery injection, we keep the reward design, control interface, and PPO configuration of the pretrained tracker unchanged.

3.1.2 State–Reference Discrepancy

The current robot state and the reference motion come from different sources and use different representations, but both describe the motion state of the same robot: the former is the state actually realized under the robot's own dynamics, contacts, and actuation constraints, while the latter is the target state prescribed by the reference motion. We therefore define the state–reference discrepancy in a unified robot state representation space:

$$\boldsymbol{d}_t \ = \ \phi(\boldsymbol{s}_t) - \phi(\boldsymbol{r}_t), \tag{4}$$

where $\boldsymbol{r}_t$ denotes the state of the reference motion at time t, and $\phi(\cdot)$ denotes the unified state representation used to compare the current state with the reference state. $\boldsymbol{d}_t$ thus describes the

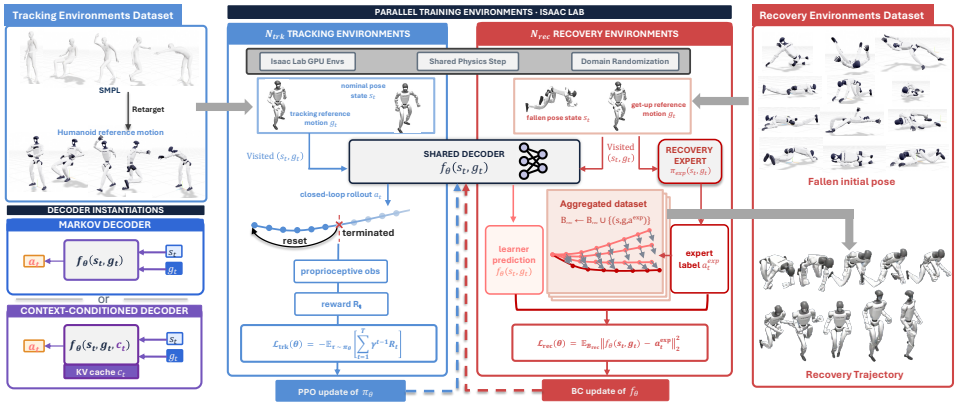

Figure 2: **Recovery injection with a shared decoder.** A subset of environments starts from fallen poses; a recovery expert labels learner-visited states (BC branch), while the tracking branch keeps PPO. Both branches update the same decoder, Markov or context-conditioned.

deviation of the current robot state from the reference state; its concrete instantiation is deferred to Section 4.3. The discrepancy is only used to describe the state deviation and for later analysis, and is not used as an additional input to the decoder.

Standard tracking pretraining usually terminates early once the robot deviates visibly from the reference motion, and therefore mainly covers the states with small discrepancy. In contrast, fall recovery starts from states that deviate substantially from the reference, and progressively reduces the orientation discrepancy through closed-loop control until the robot returns to the tracking regime. We therefore view recovery injection as an extension of the decoder's supervised region:

$$\mathcal{D}_{\text{trk}} \longrightarrow \mathcal{D}_{\text{rec}}, \tag{5}$$

where $\mathcal{D}_{\text{trk}}$ denotes the state region covered by tracking pretraining, and $\mathcal{D}_{\text{rec}}$ denotes the states with large discrepancy involved in recovery.

3.2 Stable Tracking with Recovery Injection

Rather than modeling recovery as a separate policy or control module, we let a single decoder handle both tracking and recovery without added capacity or a changed control interface. To this end, we adopt a two-branch training framework with a shared decoder (Figure 2): the tracking branch retains the pretrained tracking objective, while the recovery branch introduces expert recovery supervision targeting states with large state–reference discrepancy. The two branches share the same decoder parameters and are optimized with PPO and imitation learning, respectively.

3.2.1 Tracking–Recovery Two-Branch Training

Let $N = N_{\text{trk}} + N_{\text{rec}}$ denote the total number of parallel training environments, with N_{trk} for tracking and N_{rec} for recovery.

Tracking branch. The tracking branch retains the reference sampling, tracking reward, termination rule, domain randomization, and rollout horizon of pretraining, and keeps optimizing the decoder with PPO:

$$\mathcal{L}_{\text{trk}}(\theta) = -\frac{1}{N_{\text{trk}}} \sum_{i=1}^{N_{\text{trk}}} \mathbb{E}_{\tau_i \sim \pi_\theta} \left[\sum_{t=1}^{T} \gamma^{t-1} R_t \right]. \tag{6}$$

Here τ_i is the rollout of the i-th of the N_{trk} tracking environments. PPO uses the standard clipped surrogate objective; the clipping range, entropy coefficient, and target KL are kept the same as the pretrained tracker, with detailed settings in Appendix C.

Recovery branch. The recovery branch initializes from fallen or perturbed states and executes the corresponding get-up reference $\boldsymbol{g}_{1:T}^{\text{rec}}$. At each recovery state $\boldsymbol{s}_t$, a goal-conditioned recovery expert provides the action label

$$\boldsymbol{s}_1 \sim \mu_{\text{fall}}, \qquad \boldsymbol{a}_t^{\text{exp}} = \pi_{\text{exp}}(\boldsymbol{s}_t, \boldsymbol{g}_t). \tag{7}$$

The expert is our Isaac Lab reproduction of the BFM-Zero algorithm (Li et al., 2025), which provides reliable closed-loop recovery actions that drive the robot from fallen or perturbed states toward the current reference state $\boldsymbol{g}_t$. The decoder f_θ receives the current state and goal $(\boldsymbol{s}_t, \boldsymbol{g}_t)$ and learns to fit the expert actions. The recovery imitation loss is defined as

$$\mathcal{L}_{\text{rec}}(\theta) = \frac{1}{N_{\text{rec}}} \sum_{i=1}^{N_{\text{rec}}} \mathbb{E}_{(\boldsymbol{s}_t, \boldsymbol{g}_t, \boldsymbol{a}_t^{\text{exp}}) \sim \mathcal{B}_{\text{rec}}} \left[\| f_\theta(\boldsymbol{s}_t, \boldsymbol{g}_t) - \boldsymbol{a}_t^{\text{exp}} \|_2^2 \right], \tag{8}$$

where the average is over the N_{rec} recovery environments, and $\mathcal{B}_{\text{rec}}$ denotes a training batch sampled from the aggregated recovery dataset. Unlike a fixed expert recovery trajectory, we use DAgger-style online data aggregation. The k-th recovery window is executed by the current learner, and the states it actually visits are recorded; expert actions are then queried on these states and added to the aggregated recovery dataset:

$$\mathcal{B}_{\text{rec}}^{k+1} = \mathcal{B}_{\text{rec}}^{k} \cup \{ (\boldsymbol{s}_t, \boldsymbol{g}_t, \boldsymbol{a}_t^{\text{exp}}) : t \in \mathcal{W}_k \}, \tag{9}$$

where $\mathcal{W}_k$ denotes the recovery states visited during the k-th recovery window. The recovery batch $\mathcal{B}_{\text{rec}}$ is sampled from the current aggregated dataset $\mathcal{B}_{\text{rec}}^k$. Thus, the supervision combines newly collected learner-visited states with states accumulated from previous recovery windows, rather than being limited to fixed expert-generated trajectories. The two branches always act on the same decoder. The tracking branch updates θ with PPO, and the recovery branch updates θ with the imitation loss.

3.2.2 Asynchronous Updates

We introduce an update interval K to control the recovery imitation frequency relative to PPO: tracking PPO updates every $n = 24$ control steps, while recovery imitation updates every Kn control steps:

$$\theta \leftarrow \text{PPOUpdate}(\mathcal{B}_{\text{trk}}) \;\; t \equiv 0 \pmod{n}, \qquad \theta \leftarrow \theta - \eta_{\text{rec}} \nabla_\theta \mathcal{L}_{\text{rec}} \;\; t \equiv 0 \pmod{Kn}. \tag{10}$$

When both conditions hold, the PPO update is applied before the recovery BC update.

A smaller K provides more frequent recovery supervision but covers shorter recovery trajectories per update, while a larger K uses longer trajectories whose older segments were rolled out by older parameter versions (Figure 3, left). We study the effect of K on recovery and tracking performance in Section 4.4.

3.3 Decoder Instantiations

We use the motion-tracking decoder of SONIC (Luo et al., 2026) as the base and post-train only the decoder, keeping all other pretrained components unchanged. Recovery injection depends on whether the decoder explicitly models historical context. We consider two parameterizations: a Markov decoder and a context-conditioned decoder.

3.3.1 Markov Decoder

A Markov decoder directly models the mapping from the current state and reference goal to an action:

$$\boldsymbol{a}_t = f_\theta(\boldsymbol{s}_t, \boldsymbol{g}_t), \tag{11}$$

where f_θ denotes the Markov decoder parameterized by θ, which we instantiate as an MLP. Since the decoder does not explicitly use historical context, both tracking and recovery supervision act directly on the same state–goal-to-action mapping, and recovery injection can directly adopt the joint training objective of Section 3.2.

3.3.2 Context-Conditioned Decoder

A context-conditioned decoder further models historical information:

$$\boldsymbol{a}_t = f_\theta(\boldsymbol{s}_t, \boldsymbol{g}_t, \boldsymbol{c}_t), \qquad \boldsymbol{c}_t = \text{KV}_\theta(\xi^s_{t-C:t-1}, \xi^g_{t-C:t-1}), \quad C = 24. \tag{12}$$

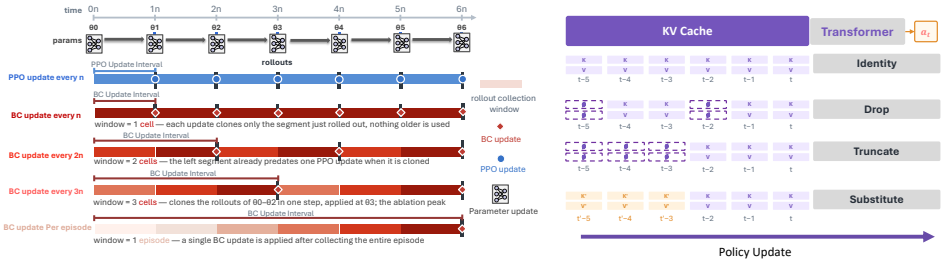

Figure 3: **The two mechanisms of recovery injection.** Left: PPO updates yield parameter versions θ_i and each rollout segment keeps the version that generated it; a recovery update clones the K segments of its window, with recovery peaking at a moderate K (Table 3(a)). The bottom row uses per-episode updates. Right: the transformations $\mathcal{K}_{\text{drop}}$, $\mathcal{K}_{\text{trunc}}$, $\mathcal{K}_{\text{subst}}$ applied to the cached key–value pairs.

where $\boldsymbol{c}_t$ denotes the historical context; historical information is provided by the key–value cache of a causal Transformer (Dai et al., 2019). Here $\text{KV}_\theta(\cdot)$ denotes the key–value cache computed under Transformer parameters θ from the tokenized state and goal inputs $\xi^s_{t-C:t-1}, \xi^g_{t-C:t-1}$ (FSQ tokenization (Mentzer et al., 2024) detailed in Appendix C), and C denotes the number of cached historical control steps. We use a grouped-query causal Transformer with RoPE and cache the keys and values of the most recent C control steps.

For the context decoder, training rebuilds the KV cache $\widetilde{\boldsymbol{c}}_t$ at each gradient update by replaying the historical prefix without gradients (burn-in; Appendix D) to keep it consistent with the current parameters.

However, in recovery, maintaining cache consistency alone is insufficient: the history in a recovery rollout usually comes from fallen states and is inconsistent with the current stand-up reference, which degrades the response of the recovery policy to the reference goal. We therefore further apply context management to the cache (Figure 3, right):

$$\widehat{\boldsymbol{c}}_t = \mathcal{K}(\widetilde{\boldsymbol{c}}_t), \qquad \mathcal{K} \in \{\mathcal{K}_{\text{drop}}, \mathcal{K}_{\text{trunc}}, \mathcal{K}_{\text{subst}}\}, \tag{13}$$

where $\widetilde{\boldsymbol{c}}_t$ is the cache rebuilt with the current parameters, $\mathcal{K}(\cdot)$ is a context transformation, and $\widehat{\boldsymbol{c}}_t$ is the transformed cache: $\mathcal{K}_{\text{drop}}$ randomly removes historical tokens, $\mathcal{K}_{\text{trunc}}$ keeps only the most recent m tokens, and $\mathcal{K}_{\text{subst}}$ replaces the historical context with that of another rollout. We evaluate the individual context management strategies in Section 4.4.

4 Experiments

We compare post-training methods (Section 4.2), analyze closed-loop behavior in state–reference discrepancy space (Section 4.3), ablate recovery updates and context management (Section 4.4), and evaluate the recovery-injected decoder on a real Unitree G1 (Section 4.5).

4.1 Experimental Setup

Platform and training. We use the 29-DoF Unitree G1, train in Isaac Lab, and evaluate in MuJoCo. Motions are retargeted from the Bonestudio Seed dataset. The exploration baselines train for 4,000 iterations, while the DAgger runs use 1,000 optimization steps. Markov experiments initialize from the official released SONIC MLP, whereas context-decoder experiments initialize from our Transformer reproduction; all remaining pretrained modules are frozen.

Compared approaches. We organize the post-training comparison as a roadmap: free recovery exploration, a 1:3 recovery-to-tracking split, and DAgger supervision on the split (ours). We additionally reproduce Ren et al. (2026); implementation details of these baselines are provided in Appendix E. The official SONIC MLP, our reproduced SONIC Transformer, and BFM-ZERO serve as capability references.

Metrics. All tracking evaluations use subsets sampled from AMASS and Motion Decode; the subset construction and split are detailed in Appendix B. Following Luo et al. (2026), we report rollout

success, MPJPE under local (L), Procrustes-aligned (PA), and global (G) alignments, velocity and acceleration errors, and root-height error. The recovery benchmark contains disturbance recovery and fall-to-stand recovery. Disturbance recovery applies 0.12 s pushes of varying magnitude, application point, and direction while the robot tracks a reference motion. Motions completed by all compared models in unpushed rollouts enter the common scored set. The knockdown rate is the fraction of pushed trials in this set that fall below the root-height threshold. On the resulting true-knockdown subset, root recovery requires $z_{\mathrm{root}} \geq 0.65$ m over a common 50-frame (1 s) window, upright recovery additionally requires upright ≥ 0.8, and motion recovery further requires tracking the same still-advancing reference within the root-height, orientation, end-effector-height, and local-MPJPE thresholds of Appendix B.2. Fall-to-stand starts from prone or supine states and commands a static upright posture over 10 s. We report *clean* trials that stand and hold after one attempt, *retried* trials that eventually succeed after an additional attempt or relapse, and *eventual* success, the sum of clean and retried outcomes, over 192 trials.

4.2 Main Results

To jointly evaluate tracking and recovery, we define the **Stability Score**:

$$\mathcal{S} = \frac{2TR}{T+R}, \tag{14}$$

where $T = (T_{\mathrm{A}} + T_{\mathrm{M}})/2$ is mean tracking success on AMASS and Motion Decode, and R averages fall-to-stand eventual success and disturbance motion recovery.

Table 1 presents the post-training roadmap. The official SONIC MLP and our reproduced SONIC Transformer retain strong tracking but cannot recover after a fall. Assigning all environments to unconstrained recovery exploration enables recovery but reduces tracking to 78.4%/72.4% on AMASS/Motion Decode. Introducing a 1:3 recovery-to-tracking environment split restores these rates to 82.0%/76.9% and raises motion recovery from 53.7% to 61.8%, yielding $\mathcal{S} = 77.5$. This split-exploration run is nevertheless expensive, requiring 8.4 GPU-hours versus 2 GPU-hours for our DAgger run in our setup. Replacing recovery exploration with expert supervision gives the Markov decoder 82.8% and 76.9% tracking success, 93.2% eventual stand-up success, and 73.6% motion recovery, increasing $\mathcal{S}$ to 81.6; this run fits on a single RTX 4090. The context decoder reaches 74.0% motion recovery but lower tracking and eventual success, giving $\mathcal{S} = 78.5$. We prioritize motion recovery because returning to the same still-advancing reference, rather than merely standing up, is the capability targeted here. The reproduced STUBBORN reaches 83.9% eventual stand-up success and 42.1% motion recovery, yielding $\mathcal{S} = 68.9$. The reproduction and exploration details are deferred to Appendix E. Gated systems retain a separate recovery policy at inference and higher-level replanning changes the reference target, so neither is evaluated in this fixed-capacity study; separately trained get-up policies serve only as capability references.

Table 1: **Post-training roadmap.** T_{A}/T_{M}: tracking success (%) on AMASS/Motion Decode; R: mean of eventual stand-up success and motion recovery; $\mathcal{S}$: Eq. (14). GPU-hours follow the measurement setup of Appendix E; "–" denotes an unmeasured reference or baseline. Bold marks the best value per column.

Stage	Added component	T_{A}	T_{M}	R	$\mathcal{S}$	GPU-h
Reference	SONIC (official MLP)	**83.0**	**82.4**	0.0	0.0	–
	SONIC (reprod. TF)	81.7	79.5	0.0	0.0	–
	BFM-ZERO (recovery expert)	46.5	40.1	67.9	52.9	–
Roadmap	Free exploration; all envs recovery	78.4	72.4	73.2	74.3	5.4
	+ Recovery:tracking env split 1:3	82.0	76.9	75.7	77.5	8.4
	+ DAgger (ours, MLP)	82.8	76.9	**83.4**	**81.6**	2
	→ Context decoder (ours, TF)	78.9	71.3	82.3	78.5	11.7
Alternative	STUBBORN (reprod.)	78.0	73.9	63.0	68.9	7.5

4.3 Decoder Behavior Analysis

Instantiating the discrepancy. To analyze decoder behavior across state–reference discrepancies, we instantiate the readout ϕ of Section 3.1.2 as three matched kinematic readouts, all evaluated in the robot body frame. Let Δq_t, $\Delta \dot{q}_t$, and $\Delta \omega_t$ denote the residuals of joint positions, joint velocities,

Table 2: **Per-metric tracking and recovery results.** *(a)* Tracking uses sampled AMASS and Motion Decode subsets; MPJPE-L/PA/G are local, Procrustes-aligned, and global errors (mm), and velocity/acceleration errors are per frame at 50 Hz. *(b)* Fall-to-stand uses 192 trials, and disturbance recovery uses the AMASS subset. Bold marks the best value per column within each block (lower is better for error metrics and knockdown).

(a) Motion tracking.

Model	Success (%)	MPJPE-L (mm)	MPJPE-PA (mm)	MPJPE-G (mm)	Vel. (mm/frame)	Accel. (mm/frame^2)	Root-h (mm)
Motion Decode							
Sonic Release (official MLP)	**82.38**	**53.35**	**32.82**	**547.30**	**3.30**	1.20	60.51
Sonic base (reprod. TF)	79.51	60.21	33.36	772.00	3.64	1.20	67.48
Explore, all Recovery envs	72.39	77.12	45.87	1397.10	6.33	1.85	68.73
Explore, split envs	76.91	79.18	44.45	832.80	4.33	1.28	80.86
BFM-Zero	40.12	117.50	51.20	1928.30	6.98	1.64	63.40
Stubborn (reprod.)	73.92	61.53	39.12	931.70	3.79	**1.14**	61.89
Ours (Markov/MLP)	76.85	63.82	38.25	1181.90	4.96	1.53	**58.02**
Ours (context/Transformer)	71.32	81.11	45.74	1343.30	6.34	1.89	71.67
AMASS							
Sonic Release (official MLP)	**83.01**	52.64	**32.19**	606.50	**7.14**	2.26	62.73
Sonic base (reprod. TF)	81.73	59.39	34.99	589.90	7.52	2.34	66.38
Explore, all Recovery envs	78.40	55.31	37.62	**557.50**	7.20	**2.03**	51.80
Explore, split envs	82.00	56.31	34.74	639.90	7.55	2.47	65.69
BFM-Zero	46.53	103.10	49.60	1377.50	11.90	2.82	59.20
Stubborn (reprod.)	78.00	71.86	42.54	579.50	8.16	2.44	73.99
Ours (Markov/MLP)	82.80	**52.43**	34.03	682.60	7.87	2.39	**47.60**
Ours (context/Transformer)	78.93	64.04	40.15	795.90	9.15	2.78	57.77

(b) Recovery benchmark.

Model	*Fall-to-stand recovery*			*Disturbance recovery*			
	Clean	Retried	Eventual	Knockdown	Root	Upright	Motion
Sonic Release (official MLP)	0/192 (0.00%)	0/192 (0.00%)	0/192 (0.00%)	68.60%	0.00%	0.00%	0.00%
Sonic base (reprod. TF)	0/192 (0.00%)	0/192 (0.00%)	0/192 (0.00%)	76.70%	0.00%	0.00%	0.00%
Explore, all Recovery envs	120/192 (62.50%)	58/192 (30.21%)	178/192 (92.71%)	69.40%	63.10%	60.00%	53.70%
Explore, split envs	117/192 (60.94%)	55/192 (28.65%)	172/192 (89.58%)	67.80%	65.50%	63.90%	61.80%
BFM-Zero	**152/192 (79.17%)**	38/192 (19.79%)	**190/192 (98.96%)**	**41.90%**	**94.70%**	**94.70%**	36.80%
Stubborn (reprod.)	77/192 (40.10%)	84/192 (43.75%)	161/192 (83.85%)	69.80%	49.80%	48.60%	42.10%
Ours (Markov/MLP)	116/192 (60.42%)	63/192 (32.81%)	179/192 (93.23%)	69.00%	75.80%	74.70%	73.60%
Ours (context/Transformer)	104/192 (54.17%)	70/192 (36.46%)	174/192 (90.63%)	80.60%	77.40%	76.40%	**74.00%**

and root angular velocity between $\boldsymbol{s}_t$ and $\boldsymbol{g}_t$, respectively, and let $\hat{\boldsymbol{g}}_t$ and $\hat{\boldsymbol{g}}_t^{\mathrm{ref}}$ denote the unit gravity directions in the current and reference root frames. We define

$$d_t^q = \left(\tfrac{1}{29}\sum_j \Delta q_{t,j}^2\right)^{1/2}, \quad d_t^{\mathrm{tilt}} = \arccos\langle \hat{\boldsymbol{g}}_t, \hat{\boldsymbol{g}}_t^{\mathrm{ref}}\rangle, \quad d_t^{\mathrm{vel}} = \left(\tfrac{1}{29}\sum_j \Delta \dot{q}_{t,j}^2 + \tfrac{1}{3}\sum_i \Delta \omega_{t,i}^2\right)^{1/2}, \tag{15}$$

measuring posture, uprightness, and motion deviations, respectively. Figure 4 illustrates the readouts (top) and the resulting discrepancy space (bottom).

Behavior in discrepancy space. Figure 4(a) shows the training inputs. Original tracking inputs concentrate near the origin, indicating that pretraining mainly covers small deviations; recovery supplementation extends them into the large-deviation region with little overlap. Starting from the same large-deviation region, the pretrained tracker remains trapped, whereas the recovery-injected decoder progressively reduces the deviation and returns near the origin (Figure 4(b,c)). Test starting points are sampled with a different random seed and are disjoint from the supplemented training states. Thus, recovery requires not exhaustive state-space coverage, but a closed-loop path from large-deviation states back to the nominal tracking region.

4.4 Ablation Studies

Recovery update interval. We ablate K (Section 3.2.2) with the Markov decoder (Table 3(a)). Tracking success stays within 82.07–83.13%, while $K = 3$ maximizes CLEAN (67.7%) and eventual success (98.4%), and $K = 6$ maximizes motion recovery (73.6%). This may reflect fresher supervision favoring stand-up at $K = 3$, while longer windows at $K = 6$ better support recovery that transfers to motion resumption. Per-episode updates, however, reduce motion recovery to 60.6%, so longer windows do not uniformly help.

Context management. Drop gives the best CLEAN (65.1%) and eventual success (93.2%), but reduces motion recovery from 68.5% to 65.3%; Subst instead achieves the best motion recovery (74.0%) (Table 3(b)). Drop may ease stand-up by removing conflicting history, whereas Subst retains temporal context while discouraging reliance on a specific history, potentially benefiting motion resumption. Trunc lowers eventual success to 86.5%, and all transformations slightly lower tracking success (at most 0.80 percentage points).

Table 3: **Ablation studies.** Tracking and disturbance recovery use the AMASS evaluation subset; recovery follows Table 2. *(a)* Markov decoder update interval; Episode denotes one recovery update per episode. *(b)* Context management for the context decoder. CLEAN measures single-attempt fall-to-stand success; bold marks the best value per row.

(a) Recovery update interval

Metric	$K = 1$	$K = 2$	$K = 3$	$K = 6$	Episode
Success (%)	82.67	**83.13**	82.33	82.80	82.07
MPJPE-L (mm)	54.50	53.50	**51.40**	52.40	53.20
MPJPE-G (mm)	666.10	**652.00**	696.70	682.60	676.50
CLEAN (%)	36.50	49.00	**67.70**	60.40	60.40
Eventual (%)	79.70	83.90	**98.40**	93.20	92.70
Motion recovery (%)	54.90	64.80	69.80	**73.60**	60.60

(b) Context management

Metric	Default	Drop	Trunc	Subst
Success (%)	**79.73**	79.13	79.07	78.93
MPJPE-L (mm)	64.80	**63.60**	66.80	64.00
MPJPE-G (mm)	757.50	**754.70**	771.80	795.90
CLEAN (%)	35.90	**65.10**	56.80	54.20
Eventual (%)	89.10	**93.20**	86.50	90.60
Motion recovery (%)	68.50	65.30	68.60	**74.00**

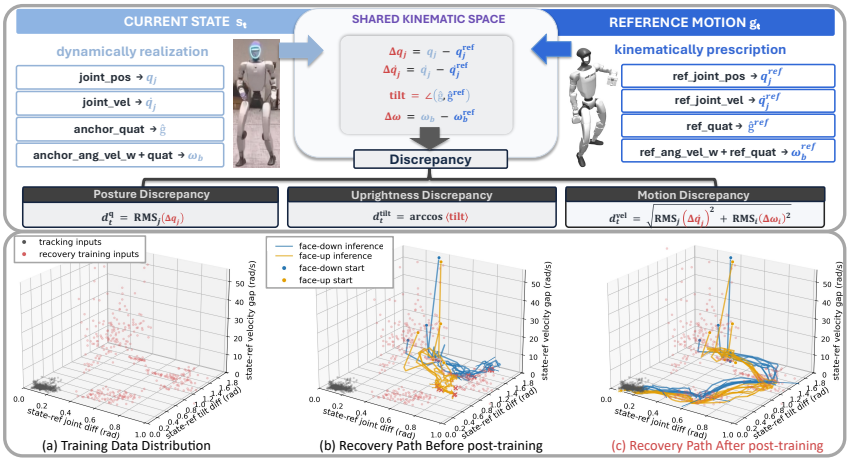

Figure 4: **State–reference discrepancy and recovery paths.** Top: the three readouts of Eq. (15). Bottom: the resulting discrepancy space; (a) training inputs, (b) pretrained tracker, (c) recovery-injected decoder.

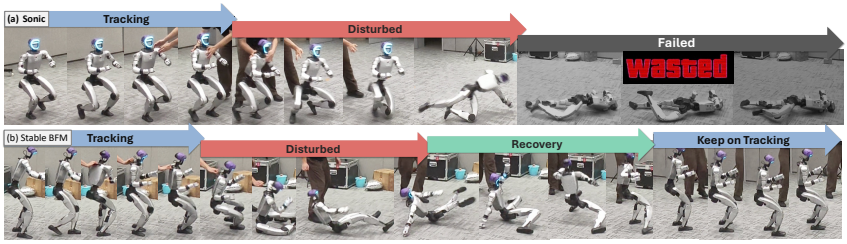

Figure 5: **Real-robot push–recovery–resume sequence on a Unitree G1.** *(a)* The pretrained decoder falls and does not rise. *(b)* The injected decoder stands up and resumes reference tracking.

4.5 Real-Robot Observation

As shown in Figure 5, under external pushes on a Unitree G1, the pretrained decoder falls and fails to rise, whereas the recovery-injected decoder stands up and resumes reference tracking.

5 Discussion

We propose StableBFM, which reframes fall recovery as large-discrepancy decoding on the same state–reference spectrum as tracking, and injects the capability into a pretrained decoder without adding parameters, recovering most of the gap on one consumer GPU and transferring to a real humanoid. The lower recovery reliability of the context decoder suggests that cached history can become a liability under large state–reference discrepancies; isolating the contribution of cache initialization, optimization, and attention allocation remains future work.

AI USE STATEMENT

This paper was prepared with the assistance of AI-based tools for draft edit, and LaTeX typesetting. All research ideas, experimental design, implementation, data analysis, and scientific claims are the work of the authors; we reviewed and edited all AI-assisted output and take full responsibility for the content of this paper. No AI tool was used to generate experimental results.

REPRODUCIBILITY STATEMENT

The full training configurations and hyperparameters are provided in Appendix C, and the standardized recovery-benchmark protocol with its thresholds in Appendix B. All experiments build on open-source simulation platforms (Isaac Lab and MuJoCo), and we plan to release the code and configurations upon publication.

ETHICS STATEMENT

This work aims to improve the safety and reliability of humanoid robots by enabling reliable fall recovery. All real-robot experiments were conducted in a controlled laboratory environment. The authors adhere to the ICLR Code of Ethics.

REFERENCES

Osher Azulay, Zhengjie Xu, Andrew Scheffer, and Stella X. Yu. Vigor: Visual goal-in-context inference for unified humanoid fall safety. *arXiv preprint arXiv:2602.16511*, 2026.

Penghui Chen, Yushi Wang, Changsheng Luo, Wenhan Cai, and Mingguo Zhao. HiFAR: Multi-stage curriculum learning for high-dynamics humanoid fall recovery. *arXiv preprint arXiv:2502.20061*, 2025.

Xiao Chen, Weishuai Zeng, Xiaojie Niu, Zirui Wang, Jianan Li, Huayi Wang, Furui Xu, Jiahe Chen, Weixiang Zhong, Lihe Ding, Kailin Li, Jiangmiao Pang, Tai Wang, Tianfan Xue, and Jingbo Wang. Reactivebfm: Reactive closed-loop motion planning towards universal humanoid whole-body control. *arXiv preprint arXiv:2606.30362*, 2026.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019.

Tairan He, Wenli Xiao, Toru Lin, Zhengyi Luo, Zhenjia Xu, Zhenyu Jiang, Jan Kautz, Changliu Liu, Guanya Shi, Xiaolong Wang, et al. Hover: Versatile neural whole-body controller for humanoid robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2025a.

Xialin He, Runpei Dong, Zixuan Chen, and Saurabh Gupta. Learning getting-up policies for real-world humanoid robots. In *Robotics: Science and Systems (RSS)*, 2025b. arXiv preprint arXiv:2502.12152.

Tao Huang, Junli Ren, Huayi Wang, Zirui Wang, Qingwei Ben, Muning Wen, Xiao Chen, Jianan Li, and Jiangmiao Pang. Learning humanoid standing-up control across diverse postures. In *Robotics: Science and Systems (RSS)*, 2025. arXiv preprint arXiv:2502.08378.

Yitang Li, Zhengyi Luo, Tonghe Zhang, Cunxi Dai, Anssi Kanervisto, Andrea Tirinzoni, Haoyang Weng, Kris Kitani, Mateusz Guzek, Ahmed Touati, et al. Bfm-zero: A promptable behavioral foundation model for humanoid control using unsupervised reinforcement learning. *arXiv preprint arXiv:2511.04131*, 2025.

Qiayuan Liao, Takara E Truong, Xiaoyu Huang, Yuman Gao, Guy Tevet, Koushil Sreenath, and C Karen Liu. Beyondmimic: From motion tracking to versatile humanoid control via guided diffusion. *Science Robotics*, 11(117):eadx8924, 2026. arXiv preprint arXiv:2508.08241.

Zeyu Ling, Xinyao Yu, Renye Yan, Jikang Cheng, Zhanke Wang, Qing Shuai, and Changqing Zou. GenTrack: Physical alignment for robot-native motion generation and zero-shot humanoid tracking. *arXiv preprint arXiv:2608.01410*, 2026.

Min Liu, Deepak Pathak, and Ananye Agarwal. Locoformer: Generalist locomotion via long-context adaptation. In *Conference on Robot Learning (CoRL)*, 2025. arXiv preprint arXiv:2509.23745.

Zhengyi Luo, Ye Yuan, Tingwu Wang, Chenran Li, Sirui Chen, Fernando Castaneda, Zi-Ang Cao, Jiefeng Li, David Minor, Qingwei Ben, et al. Sonic: Supersizing motion tracking for natural humanoid whole-body control. *Science Robotics*, 11(117):eaed4592, 2026.

Yubiao Ma, Han Yu, Kai Guo, Changtai Lv, Zhengquan Mao, Boyang Xing, Xuemei Ren, and Dongdong Zheng. Extreme-rgmt: Continual learning of highly dynamic skills for robust generalist humanoid control. *arXiv preprint arXiv:2607.20110*, 2026a.

Yubiao Ma, Han Yu, Jiayin Xie, Changtai Lv, Qiang Luo, Chi Zhang, Yunpeng Yin, Boyang Xing, Xuemei Ren, and Dongdong Zheng. Robust and generalized humanoid motion tracking. *arXiv preprint arXiv:2601.23080*, 2026b.

Fabian Mentzer, David Minnen, Eirikur Agustsson, and Michael Tschannen. Finite scalar quantization: VQ-VAE made simple. In *International Conference on Learning Representations*, 2024.

Zekun Qi, Xuchuan Chen, Dairu Liu, Chenghuai Lin, Yunrui Lian, Sikai Liang, Zhikai Zhang, Yu Guan, Jilong Wang, Wenyao Zhang, Xinqiang Yu, He Wang, and Li Yi. Humanoid-GPT: Scaling data and structure for zero-shot motion tracking. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2026. arXiv preprint arXiv:2606.03985.

Xiao Ren, Yuhui Yang, Zongbiao Weng, Zhijie Liu, and He Kong. Stubborn: A streamlined and unified reinforcement learning framework for robust motion tracking and fall recovery for humanoids. *arXiv preprint arXiv:2606.12814*, 2026.

Stéphane Ross, Geoffrey J Gordon, and J Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Chen Tessler, Yunrong Guo, Ofir Nabati, Gal Chechik, and Xue Bin Peng. Maskedmimic: Unified physics-based character control through masked motion inpainting. *ACM Transactions on Graphics*, 43(6), 2024.

Yunshen Wang, Shaohang Zhu, Peiyuan Zhi, Yuhan Li, Jiaxin Li, Yong-Lu Li, Yuchen Xiao, Xingxing Wang, Baoxiong Jia, and Siyuan Huang. OmniXtreme: Breaking the generality barrier in high-dynamic humanoid control. In *Robotics: Science and Systems (RSS)*, 2026a. arXiv preprint arXiv:2602.23843.

Zifan Wang, Yizhao Li, Teli Ma, Qiang Zhang, Yudong Fan, Hao Xu, Shuo Yang, and Junwei Liang. Perceptive Behavior Foundation Model: Adapting human motion priors to robot-centric terrain. *arXiv preprint arXiv:2606.08059*, 2026b.

Weihao Wu, Ming Huang, Ruofei Liu, Jinglei Nie, Shuxiang Guo, and Chunying Li. StableMimic: Smooth human-like recovery for humanoid motion tracking—learning beyond the tracking distribution for structured post-fall behavior. *arXiv preprint arXiv:2608.02385*, 2026.

Xiaoyu Xu, Zhiming Chen, Yuenan Zhao, Ran Song, and Wei Zhang. UniReLo: Learning a unified humanoid policy from fall recovery to locomotion across diverse terrains. *arXiv preprint arXiv:2606.08922*, 2026.

Zhengjie Xu, Ye Li, Kwan-yee Lin, and Stella X. Yu. Unified humanoid fall-safety policy from a few demonstrations. *arXiv preprint arXiv:2511.07407*, 2025.

Yifei Yao, Chengyuan Luo, Jiaheng Du, Wentao He, and Jun-Guo Lu. GBC: Generalized behavior-cloning framework for whole-body humanoid imitation. *arXiv preprint arXiv:2508.09960*, 2025.

Weishuai Zeng, Shunlin Lu, Kangning Yin, Xiaojie Niu, Minyue Dai, Jingbo Wang, and Jiangmiao Pang. Behavior foundation model for humanoid robots. *arXiv preprint arXiv:2509.13780*, 2025.

Weishuai Zeng, Kangning Yin, Xiaojie Niu, Shunlin Lu, Weixiang Zhong, Jiahe Chen, Feiyu Jia, Xiao Chen, Zirui Wang, Furui Xu, Ming Zhou, Kailin Li, Weinan Zhang, He Wang, Li Yi, Dahua Lin, Jiangmiao Pang, and Jingbo Wang. Scaling behavior foundation model for humanoid robots. *arXiv preprint arXiv:2607.15163*, 2026.

Tong Zhang, Boyuan Zheng, Ruiqian Nai, Yingdong Hu, Yen-Jen Wang, Geng Chen, Fanqi Lin, Jiongye Li, Chuye Hong, Koushil Sreenath, and Yang Gao. Hub: Learning extreme humanoid balance. In *Conference on Robot Learning (CoRL)*, 2025. Oral. arXiv preprint arXiv:2505.07294.

A Further Related Work

Section 2 summarizes the closest fall-recovery and behavior foundation model literature. Here we place our setting in a broader landscape of unified recovery, capability extension, and history-dependent control.

A.1 Recovery without Abandoning Tracking

Humanoid recovery methods differ in how recovery is integrated with nominal control. Dedicated getting-up policies learn a specialized skill that is invoked after failure (He et al., 2025b; Huang et al., 2025; Chen et al., 2025), whereas unified fall-safety policies cover falling, impact mitigation, and standing up within a single policy (Xu et al., 2025; Azulay et al., 2026; Xu et al., 2026). Recent work moves closer to preserving an ongoing tracking task. Stubborn retains one reinforcement learning policy and exposes it to unstable states through probabilistic termination and tracking-error-driven resampling (Ren et al., 2026). StableMimic instead trains dedicated tracking and recovery experts and blends their actions through a proprioceptive gate; its deployed actor resumes an advancing command without receiving a get-up reference (Wu et al., 2026). Robust and Generalized Humanoid Motion Tracking combines dynamics-conditioned command aggregation with a causal temporal encoder and a curriculum of unstable initial states and annealed upward assistance (Ma et al., 2026b).

Our problem setting imposes a different constraint: recovery must be added to an existing tracker without a second deployed policy, expert, gate, or modified reference generator. The expert is used only to supervise learner-visited recovery states during post-training, while the same action decoder continues to optimize the original tracking objective. This distinction also motivates our motion-recovery criterion, which requires returning to the still-advancing reference rather than only reaching an upright pose.

A.2 Capability Extension of Pretrained Humanoid Controllers

A second line of work expands a pretrained controller by changing its interface, architecture, or training distribution. BFM freezes a generative base and adds a residual decoder for downstream control (Zeng et al., 2025). Perceptive BFM preserves a raw-motion interface while injecting terrain observations through identity-gated residual pathways, trained by transferring a terrain-adapted teacher to a raw-reference student (Wang et al., 2026b). ReactiveBFM couples a BFM controller to an autoregressive motion planner, using scheduled prefix sampling and asynchronous replanning to make generated references responsive to physical execution errors (Chen et al., 2026). These methods extend the inputs or planning stack around the controller.

Other approaches adapt the training process itself. GenTrack alternates execution-grounded generator alignment with tracker training on newly generated references, using anchoring and rehearsal to limit drift (Ling et al., 2026). OmniXtreme separates broad flow-matching pretraining from actuation-aware refinement for high-dynamic deployment (Wang et al., 2026a). Extreme-RGMT first learns a generalist tracker and then constrains policy drift while emphasizing difficult motion segments during continual skill acquisition (Ma et al., 2026a). Our method likewise treats recovery

as capability extension, but keeps the deployed architecture, state–reference interface, and reference stream fixed; only the existing decoder is post-trained with joint tracking and recovery supervision.

A.3 Temporal Context and Learner-Visited Supervision

Scaling and temporal context provide another route to generality. ScaleBFM coordinates motion diversity, on-policy rollout scale, and a Humanoid Transformer architecture (Zeng et al., 2026), while Humanoid-GPT trains a causal Transformer on a two-billion-frame retargeted corpus for zero-shot motion tracking (Qi et al., 2026). LocoFormer extends context across episode boundaries and exhibits test-time adaptation to morphology, dynamics, and failures (Liu et al., 2025). Robust and Generalized Humanoid Motion Tracking instead uses recent proprioception to select among commands in a temporal window (Ma et al., 2026b). Collectively, these results show that history can improve adaptation and command interpretation.

Our results expose a complementary failure mode: after a fall, a coherent history can describe states that are incompatible with the current stand-up reference, so additional context may reinforce the wrong conditional mapping. This issue is distinct from stale-cache computation, which burn-in addresses. It is also distinct from the distribution shift addressed by classical DAgger, which queries an expert on learner-visited states (Ross et al., 2011). GBC combines a DAgger-MMPPO procedure with a Transformer for cross-morphology whole-body imitation (Yao et al., 2025); in our setting, learner-visited supervision is combined with selective perturbation of the recovery cache. Thus context management does not enlarge the model or replace temporal conditioning; it prevents recovery BC from relying on history alone while preserving context for nominal tracking.

B Benchmark Protocol

B.1 Tracking Benchmark Construction

Motion Decode OmniEval-1503 contains 1,503 clips constructed from 501 leaf motions in the ChingMu 1000-hour collection, with short-, medium-, and long-horizon variants for each leaf. The AMASS benchmark contains 1,500 clips sampled from 17,710 candidates with seed 20260902. Sampling is stratified over 25 source datasets in proportion to their candidate counts, assigns at least one clip to each source, and distributes the remainder by the largest-remainder rule. Both sets are held out from training, which instead uses the Bonestudio Seed motion pool; motion names are deduplicated across the training and evaluation pools.

B.2 Disturbance-Recovery Criteria

Each motion is first rolled out without a push. Only motions whose no-push control runs complete for every model enter the common scored set, so all recovery methods are compared on identical motions. A pushed trial is a *knockdown* if the root height satisfies $z < 0.55$ m at any frame after the push. Trials that resist the push are reported separately and are never counted as recoveries; all recovery rates are conditioned on the true-knockdown subset. *Root recovery* requires, after the first knockdown, root height $z \geq 0.65$ m for 50 consecutive frames (1 s), with no 3-s deadline. *Upright recovery* additionally requires

$$\text{upright} = 1 - 2(q_x^2 + q_y^2) \geq 0.8 \tag{16}$$

throughout the same window. *Motion recovery* ("continue tracking", our primary capability metric) further requires all conditions in Table 4 throughout that window.

B.3 Fall Taxonomy and Initialization

Fall-to-stand postures are generated in MuJoCo by applying a 128–550 N horizontal push for 0.12 s to a standing robot and then allowing it to fall for 3 s. We retain grounded, non-ballistic states whose root six-dimensional speed is below 4.0, root height is at most 0.40 m, uprightness is at most 0.5, and $|f_{\text{wd},z}| > 0.8$. The sealed benchmark contains 96 prone states (41 dynamic and 55 settled) and 96 supine states (35 dynamic and 61 settled), generated with seeds 101 and 202, respectively. Each trial commands a static standing posture for 500 control frames (10 s), aligns the target yaw with the robot heading, and uses neither early termination nor trial teleportation.

Table 4: **Motion-recovery criteria.** Conditions that the same 50 consecutive frames must all satisfy, in addition to root and upright recovery.

Condition	Threshold	Meaning
Root height error	≤ 0.25 m	Body height keeps up with the still-advancing reference
Root full orientation angle^2 (incl. yaw)	≤ 1.0 rad^2	Orientation keeps up with the reference
Max limb-endpoint height error	≤ 0.25 m	Both ankles / both wrists return in height
Local MPJPE (14 keypoints)	≤ 200 mm	Whole-body pose recovery after removing root translation

A trial is *clean* if it makes exactly one upward crossing of 0.45 m, reaches 0.65 m without subsequently falling below 0.60 m, and ends stably. A trial is *retried* if it eventually succeeds after multiple attempts or a relapse. *Eventual* success requires the mean root height over the final 50 frames and the final-frame root height both to be at least 0.65 m. Disturbance episodes start from nominal tracking on the reference set of §4.1.

B.4 Disturbance Matrix

From the 1,500-motion AMASS evaluation set, we retain the 337 motions with at least 700 frames and evaluate the first 700 frames (14 s). Each matched push is sampled once with seed 20260903 and released between frames 75 and 194, leaving at least 10 s for post-push observation. Table 5 summarizes the factors.

Table 5: **Disturbance matrix.** The common scored set is the intersection of motions whose no-push control rollout succeeds for every compared model.

Factor	Values
Magnitude	500 / 700 / 900 N
Application point	pelvis center / torso center / torso $+0.18$ m
Direction	forward / backward / left / right
Duration	0.12 s
Release window	frames 75–194 (1.50–3.88 s)

B.5 Statistical Protocol

Tracking metrics are computed by `eval_unify.py`; fall-to-stand and disturbance recovery use `run_recovery_benchmark.py` and `run_push_matrix.py`, respectively.

C Training Details

Tracking PPO. The tracking branch keeps the PPO configuration of the pretrained tracker (Luo et al., 2026) unchanged: $n = 24$ environment steps per update, 5 learning epochs over 4 mini-batches, discount $\gamma = 0.99$, GAE $\lambda = 0.95$, clip range 0.2, entropy coefficient 0.013, value-loss coefficient 1.0, and a gradient-norm clip of 0.1. The learning rate is adapted online so that the measured policy KL stays near a target of 0.01, within $[1 \times 10^{-5}, 2 \times 10^{-4}]$. The KL therefore acts as a step-size controller, not as a penalty term in the objective of Eq. (3).

Recovery supervision and context management. The recovery expert is our Isaac Lab reproduction of BFM-Zero (Li et al., 2025) and is used only to label states visited by the learner. Recovery BC samples batches from the aggregated dataset of learner-visited states accumulated across recovery windows; when PPO and BC are scheduled at the same step, PPO is applied first. For context management, Drop independently removes historical tokens with probability 0.5, Trunc retains the most recent 12 tokens, and Subst draws the replacement history from another rollout in the same recovery mini-batch.

Architecture. The context decoder is a 2-layer grouped-query causal Transformer with model width 512, 8 query heads, 4 key–value heads, a 2,464-dimensional SwiGLU feed-forward block, RoPE positional encoding, and a ring KV cache of $C = 24$ control steps. The current state and reference goal are encoded as two 64-dimensional tokens using an FSQ tokenizer (Mentzer et al., 2024) with 32 levels over 32 quantized dimensions. Its action head is a 512–512–29 MLP with a SiLU hidden activation. The Markov decoder uses hidden sizes 2048–2048–1024–1024–512–512 with SiLU activations. Both decoders output 29 actions through the same PD interface.

Recovery hyperparameters. Table 6 lists the recovery-injection settings.

Table 6: **Recovery-injection hyperparameters.**

Symbol	Meaning	Value
$\rho = N_{\mathrm{rec}}/N$	recovery-environment fraction	$1024/4096 = 1/4$
K	recovery update multiplier	6
η_{rec}	recovery decoder learning rate	2×10^{-4}
β_{rec}	recovery BC coefficient	2.0
μ_{fall}	fallen-posture initialization	353 states (213 falls + 140 push-up-phase frames)
Drop prob.	context management $\mathcal{K}_{\mathrm{drop}}$	0.5
Trunc. length m	context management $\mathcal{K}_{\mathrm{trunc}}$	12

The fallen-posture pool contains 353 states: 213 MuJoCo push-generated falls binned by root height (48/42/49/45/29, from heavy lean to fully down) and 140 push-up-phase frames of the BFM-Zero teacher. All states are used, assigned cyclically across recovery environments, with ± 0.05 heading noise and goal-heading alignment.

The recovery expert is our Isaac Lab reproduction of BFM-Zero (Li et al., 2025), trained strictly following the official implementation; it labels learner-visited states only and is not deployed.

Compute budget. The exploration baselines train for 4,000 iterations, while the DAgger runs use 1,000 optimization steps. Our DAgger implementation requires 2 GPU-hours with the Markov decoder and 11.7 GPU-hours with the context decoder on a single RTX 4090, while the split-environment exploration run requires 8.4 GPU-hours under the same measurement setup. The full accounting protocol is deferred to Appendix E.

D Cache Warm-Up and Context Management

A causal Transformer trained in fixed-length chunks has an effective receptive field of

$$H = 1 + L(C - 1) \tag{17}$$

control steps for L layers and a cache of C steps. For our two-layer decoder with $C = 24$, this yields $H = 47$. The cache saved with a rollout is generated by the parameters before its PPO updates; reusing it after those updates conditions the new decoder on stale keys and values. Context burn-in addresses this numerical inconsistency by replaying the preceding $H - 1 = 46$ steps with the current weights and no gradients before evaluating the loss on a chunk. Equation (18) gives the resulting cache:

$$\widetilde{\boldsymbol{c}}_t = \mathrm{sg}\big[\mathrm{KV}_\theta(\xi^s_{t-B:t-1}, \xi^g_{t-B:t-1})\big], \qquad B = L(C-1), \tag{18}$$

where $\mathrm{sg}[\cdot]$ denotes stop-gradient and L denotes the number of Transformer layers.

Context management addresses a distinct issue. The burn-in cache is coherent with the trajectory and therefore remains a strong shortcut for the behavioral-cloning loss. In recovery, a coherent nominal history cannot replace identifying the current posture from proprioception. Table 7 contrasts the two operations: burn-in makes the conditioning current; context management makes the target unpredictable from history alone.

Table 7: **Burn-in and context management.** The operations are complementary: burn-in reconstructs a cache under the current parameters, whereas context management perturbs an otherwise coherent cache to enforce state-conditioned recovery.

	Burn-in	Context management
Applied during	PPO/BC optimization	recovery BC optimization
Operation	replay coherent prefix with current θ	drop / truncate / substitute cache
Conditioning	current, coherent KV entries	absent or contradictory history
Gradient purpose	avoid stale-cache updates	prevent history-only recovery
Supervision	unchanged PPO or BC target	same expert target under perturbation
Does not solve	context over-reliance	numerical cache staleness
Extra token-forward cost	$(B + C)/C$	none

E Post-Training Baseline Reproductions

All comparative post-training runs share the same data and control interface, and differ only in the training objective. Every run initializes the decoder from a pretrained tracking decoder (the released tracking checkpoint `model_step_041550` for the Markov runs, and our Transformer reproduction for the context-decoder run; Section 4.1) with a non-strict load (minimum key match 0.98), keeps the robot, the `boneseed130k` motion pool, the PD interface, and the observation/tokenizer path of the pretrained tracker unchanged, and freezes every parameter except the deployed action decoder (`g1_dyn`). Each run uses one GPU and is evaluated at the end of its training budget; no checkpoint selection is performed. The DAgger runs use $1{,}000$ optimization steps; the three baselines are given $4{,}000$ steps because they must discover recovery behavior rather than imitate it.

The recovery-exploration environments (in both exploration baselines) disable early termination based on tracking discrepancy: during a recovery episode the tracking-discrepancy termination terms are masked out, and the episode ends only at the recovery timeout of 23 s ($1{,}150$ control steps, with the replay row truncated at $1{,}200$ steps), matching the setting used in the main comparison. The comparison runs train on the flat plane, which the fall-posture initialization requires: the fall poses are generated on flat ground, and on uneven terrain they clip into or float above the surface. The Stubborn reproduction inherits the released tracker's rough-terrain configuration instead, since it never initializes from a stored fall pose.

E.1 Stubborn Reproduction

We reproduce the two algorithmic mechanisms of Ren et al. (2026) and apply them to post-train the released tracking decoder, rather than re-running the original from-scratch recipe; the trainable scope and capacity therefore match the recovery experiments rather than the paper's own training run.

Probabilistic termination (Eq. 3). A row becomes eligible for failure termination once the absolute root-height error exceeds 0.25 m or the full root-orientation error exceeds $\pi/2$. Eligible rows then terminate independently with probability $p = 0.005$ per control step, so a failing row continues with probability 0.995 and the expected exploration window is $1/p = 200$ steps (4 s); all other termination terms are removed and failures are never hard-terminated. The episode boundary is the ordinary time limit $T_{\max} = 1{,}000$ steps (20 s at 50 Hz), which preserves ordinary PPO bootstrapping.

Reference reweighting (Eqs. 4–6). The motion library is partitioned into bins of 50 frames. Each bin carries a weight, initialized to 1.0 and clipped to $[0.05, 1.0]$; a bin's sampling probability is directly proportional to its weight (masses are renormalized per source dataset). At every episode end, the weight of every bin touched by the episode is updated from the episode's mean keypoint error and outcome:

- mean error ≥ 0.06 (failed): every touched bin receives $+0.10$;
- the episode survived to $T_{\max}$: its bins receive -0.10 in the first half of the episode segment and -0.05 in the second half;

- the episode was ended by the probabilistic failure term with mean error < 0.06: bins in the first third receive -0.10, the first-to-middle segment -0.05, and the final third $+0.10$.

This is the bin-level equivalent of Eqs. 4–6 of Ren et al. (2026): the unreliable early portion of a surviving episode is downweighted, while the tail of an early-terminated episode is upweighted. Weights are additionally decayed by 0.8 at each resampling interval.

Training settings. The run keeps the tracking reward and the PPO configuration of the pretrained tracker (no recovery reward term): $n = 24$ steps per environment per update, 5 epochs, 4 mini-batches, $\gamma = 0.99$, $\lambda = 0.95$, clip range 0.2, value-loss coefficient 1.0, entropy coefficient 0.01, gradient-norm clip 0.1, and separate actor/critic learning rates of 2×10^{-4} and 1×10^{-3} with an adaptive-KL step-size controller targeting 0.01. The run uses $4{,}096$ environments for $4{,}000$ optimization steps; PPO applies to all rows (no environment masking) and no recovery-specific data or loss is used. It requires 7.5 GPU-hours (§E.4).

E.2 Free Recovery Exploration (Shared Environment Pool)

The first exploration baseline converts the entire environment pool to recovery: all environments start from fallen postures, no environment follows a tracking reference, and no teacher label is used. PPO is disabled on these rows (the valid mask is empty and the PPO coefficient is zero), so the only learning signal is reward-weighted self-imitation of the student's own discovered recovery trajectories.

Recovery reward. Each recovery row is commanded a *static* standing reference aligned to the robot's own heading, routed through the native G1 encoder path. The reward is computed against this static goal only and never against a randomly sampled tracking motion. With root height z, unit gravity projection g_z, and body-frame root velocities v_b, ω_b:

$$r \;=\; h\,u\,\big(0.5 + 0.5\,s\big) \;+\; \mathbb{1}[z \geq 0.65 \;\wedge\; u \geq 0.8]\,, \tag{19}$$

where

$$h = \sigma\!\left(\frac{z - \min(z^\star, 0.65)}{0.08}\right), \quad u = \mathrm{clamp}(-g_z, 0, 1), \quad s = \exp\!\big(-\tfrac{1}{2}\|v_b\|^2 - 0.2\|\omega_b\|^2\big)\,, \tag{20}$$

and $z^\star$ is the commanded standing height. The height and upright terms reward progress out of fallen states; the stillness term distinguishes a stable stand from briefly crossing the height threshold, and the indicator is the bonus for the standing condition.

Self-imitation updates. Completed recovery episodes enter a CPU reservoir of 256 episodes; an episode is eligible only if its peak root height reaches 0.45 m, and once the reservoir is full, eligible episodes replace a uniformly chosen resident with probability $256/(\text{eligible so far})$. Each stored transition receives a discounted return-to-go with $\gamma = 0.99$, normalized by the geometric horizon. Every optimization step samples $2{,}048$ transitions (episode chosen uniformly, then step uniformly within it), standardizes the sampled returns across the batch, converts them into exponential weights $\exp(\hat{R}/T)$ with temperature $T = 1.0$ clamped at 20 and renormalized to mean 1, and minimizes the weighted action MSE to the student's own recorded action under its mean output, at a dedicated learning rate of 2×10^{-5}, 2 updates per iteration, training only the action decoder. This is a capacity-matched teacher-free control, not full BFM-Zero FB-CPR.

Exploration and domain randomization. The exploration environments retain the pretrained tracker's domain randomization, applied at startup or at resampling intervals: (i) surface properties of all robot bodies with static friction $\mathcal{U}(0.3, 1.6)$, dynamic friction $\mathcal{U}(0.3, 1.2)$, and restitution $\mathcal{U}(0, 0.5)$; (ii) joint default positions offset by $\mathcal{U}(-0.01, 0.01)$ rad; (iii) torso center-of-mass offset ± 0.025 m (x) and ± 0.05 m (y, z); (iv) mass scaling $\mathcal{U}(0.8, 2.5)$ on the wrists and torso; and (v) interval velocity pushes every 4–6 s, which are training-only events and are removed for evaluation. Fallen postures are drawn from the same 353-pose mixture as the DAgger runs (Appendix C), cycled sequentially across environments with ± 0.05 rad anchor-orientation noise.

Budget. The run uses $1{,}024$ environments (all allocated to recovery) for $4{,}000$ optimization steps and requires 5.4 GPU-hours (§E.4).

E.3 SPLIT-ENVIRONMENT EXPLORATION

The split baseline keeps a tracking branch and allocates environments between the two tasks in a 1:3 ratio: 3,072 environments retain the pretrained tracking task and its PPO objective, while 1,024 recovery environments use exactly the free-exploration objective of Appendix E.2 (same static standing reward, same reservoir, same $2 \times 2{,}048$-transition self-imitation schedule every iteration, same initialization mixture, same domain randomization, same timeout and normalization statistics). A valid mask routes each row to exactly one objective: tracking rows contribute PPO loss only, recovery rows contribute self-imitation updates only; no teacher labels, auxiliary losses, or DAgger replay are enabled in this run. This isolates the effect of restricting exploration to the recovery environments while leaving the tracking data distribution untouched.

The run trains for 4,000 optimization steps on 4,096 environments and requires 8.4 GPU-hours; a second repeat with identical settings took 8.7 GPU-hours. A 1:7 variant (3,584 tracking + 512 recovery environments) was also trained for completeness.

E.4 COMPUTE ACCOUNTING

All GPU-hour figures below are single-GPU wall-clock measured from launcher start to the final checkpoint of the run, and therefore include rollout collection, all optimization steps, and checkpoint writing; evaluation and dataset preprocessing are excluded, since evaluations are launched as separate jobs after training. Rollout data collection is inside the training loop in all cases (all methods are on-policy or online), so it is included by construction. Table 8 reports the parallel environment counts and the measured cost of every compared run.

Table 8: **Compute accounting.** Parallel environment counts and the single-GPU wall-clock cost of every compared run.

Run	Environments	Optimization steps	GPU-h
DAgger (Markov)	4096	1,000	2.00
DAgger (context)	4096	1,000	11.70
STUBBORN (reprod.)	4096	4,000	7.50
Free exploration	1024	4,000	5.40
1:3 split	3072 + 1024	4,000	8.40

E.5 CONFIGURATION SUMMARY

Table 9 consolidates the training and recovery configuration for quick reference.

Table 9: **Consolidated post-training configuration.**

Setting	Value
Tracking PPO (unchanged from Luo et al. (2026))	
Update scheme	$n = 24$ environment steps per update; 5 epochs over 4 minibatches
Objective	$\gamma = 0.99$; GAE $\lambda = 0.95$; clip range 0.2; entropy coefficient 0.013; value loss 1.0; gradient-norm clip 0.1
Learning rate	KL-targeted 0.01, within $[1 \times 10^{-5}, 2 \times 10^{-4}]$
Recovery injection (ours)	
Environment fraction	$\rho = N_{\text{rec}}/N = 1024/4096 = 1/4$
Update interval	$K = 6$
Recovery BC	learning rate 2×10^{-4}; coefficient $\beta_{\text{rec}} = 2.0$
Fall initialization	353 states (213 falls + 140 push-up-phase frames)
Context management	Drop $p = 0.5$; Trunc $m = 12$; Subst from another rollout in the same minibatch
Decoders	
Context decoder	2-layer grouped-query Transformer, width 512, 8 query / 4 KV heads, SwiGLU 2464, RoPE, ring cache $C = 24$
State and goal tokens	two 64-dimensional tokens, FSQ tokenizer (32 levels over 32 quantized dims)
Action heads	512–512–29 (context); 2048–2048–1024–1024–512–512 (Markov)
Baseline reproductions	
Stubborn	4096 environments; 4000 iterations; termination probability $p = 0.005$; 50-frame reference bins; init. from the step-41,550 checkpoint
Free exploration	1024 recovery environments; 4000 iterations; RL without the PPO loss; two reward-weighted self-imitation updates per iteration (batch 2048, 256-episode reservoir)
1:3 split	3072 tracking + 1024 recovery environments; 4000 iterations; recovery rows masked from PPO and updated by two self-imitation updates per iteration
Compute (single RTX 4090)	
Our runs	2.00 GPU-h (Markov); 11.70 GPU-h (context)
Baselines	7.50 (Stubborn); 5.40 (free exploration); 8.40 (1:3 split)