Residual MPC: Blending Reinforcement Learning with GPU-Parallelized Model Predictive Control

Se Hwan Jeon[1], Ho Jae Lee[1], Seungwoo Hong[2], Sangbae Kim[1]

***Abstract*—Model Predictive Control (MPC) provides interpretable, tunable locomotion controllers grounded in physical models, but its robustness depends on frequent replanning and is limited by model mismatch and real-time computational constraints. Reinforcement Learning (RL), by contrast, can produce highly robust behaviors through stochastic training but often lacks interpretability, suffers from out-of-distribution failures, and requires intensive reward engineering. This work presents a GPU-parallelized residual architecture that tightly integrates MPC and RL by blending their outputs at the torque-control level. We develop a kinodynamic whole-body MPC formulation evaluated across thousands of agents in parallel at 100 Hz for RL training. The residual policy learns to make targeted corrections to the MPC outputs, combining the interpretability and constraint handling of model-based control with the adaptability of RL. The model-based control prior acts as a strong bias, initializing and guiding the policy towards desirable behavior with a simple set of rewards. Compared to standalone MPC or end-to-end RL, our approach achieves higher sample efficiency, converges to greater asymptotic rewards, expands the range of trackable velocity commands, and enables zero-shot adaptation to unseen gaits and uneven terrain.**

I. Introduction

MODEL predictive control (MPC) has been widely used for robotic locomotion control due to its grounding in physical models of the robot's dynamics and constraints [2–7]. By explicitly predicting the evolution of the system over a finite horizon, MPC can generate control inputs that optimize performance while respecting physical limits such as joint torques, contact forces, and stability margins. These models are valid throughout the system's state space and, because the underlying optimization problem is explicit, they provide interpretable predictions and tunable parameters for transparent controller design. The designer can readily inspect internal quantities such as predicted trajectories, constraint satisfaction, or cost terms, which makes MPC attractive not only for deployment but also for debugging and analysis. However, the performance of MPC derives primarily from frequent replanning and online optimization, which makes it vulnerable to inaccuracies in the assumed model. Even small discrepancies between the modeled and actual system, such as uncertainties in terrain shape, variable ground friction, actuator nonlinearities, or unmodeled compliance, can accumulate between replanning steps and destabilize the controller, particularly with computation or actuation delays. This sensitivity to modeling errors and environmental disturbances remains a key challenge for pure model-based locomotion control. Moreover, because MPC relies on solving a structured optimization problem at high bandwidth, the need for real-time, gradient-based computation also restricts the expressiveness of the models and constraints that can be incorporated in practice [4, 5, 8].

In contrast, reinforcement learning (RL) provides a complementary paradigm that bypasses the need for explicit modeling by directly optimizing control policies from simulated data. By training stochastically under diverse simulated conditions, RL can expose policies to a broad spectrum of uncertainties, perturbations, and noise, producing behaviors that often exhibit remarkable robustness to parameter uncertainty and the ability to generalize to unseen terrains [9–14]. Policies learned through RL can, in principle, adapt to phenomena that are difficult to model, such as unexpected contact events or nonlinear actuator dynamics [15, 16]. Nonetheless, RL methods are hampered by the black-box nature of neural networks, which limits interpretability and complicates debugging or analysis: it is often unclear why a given action was chosen, or how a policy will behave outside the training distribution. Furthermore, the design of rewards, penalties, or curricula to steer learning toward a desired behavior can often be unintuitive and time-consuming. Small changes in reward weighting or environment randomization can lead to drastically different policies, sometimes exploiting numerical simulation artifacts rather than learning physically plausible locomotion behavior [17, 18].

Blending MPC and RL seeks to combine the interpretability, tunability, and predictive guarantees of MPC with the robustness and adaptability of RL. The intuition is to let MPC provide a structured, physically consistent baseline while allowing a learned policy to make targeted corrections that account for unmodeled effects or constraints difficult to include in the formulation. This integration offers the potential to reduce the dependence on reward engineering, manual curriculum design, and hyperparameter tuning that is characteristic of end-to-end RL. Various architectures have been proposed to achieve such a synthesis, including hierarchical structures where a high-level policy outputs MPC parameters (e.g., [19]), learning to track predicted states and commands from MPC predictions (e.g., [20]), or embedding learnt dynamics or constraints into an MPC formulation (e.g., [21]). A thorough survey of synthesizing MPC and RL is presented in [22].

Another promising direction is to adopt a parallel or *residual* policy architecture, where the outputs of a fixed control prior are modified with corrective actions from a trained

[1]Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge MA, USA.

[1]Department of Mechanical Engineering, School of Smart Mobility, Korea University, Seoul, South Korea.

Video: https://www.youtube.com/watch?v=L2NrPD4yiMs

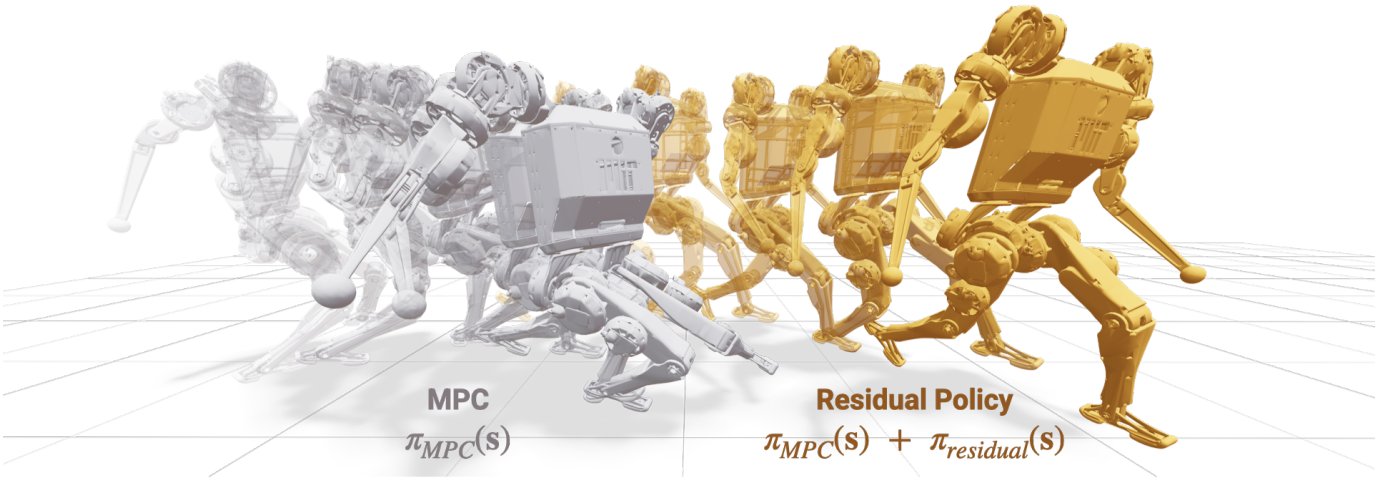

Fig. 1. Simulated locomotion for MPC (left, grey) and the residual policy (right, orange) at velocity commands of 2.25 m/s and -1.5 rad/s. The MPC controller fails and terminates from self-collision, while the residual policy is able to stabilize the robot. We use Viser for 3D visualization in this work [1].

policy [23–25]. Whereas a hierarchical architecture depends on reasonable outputs from the higher-level controller, a parallel architecture can have its components interact with each other adaptively. From the perspective of model-based control, the residual network would serve as a corrective layer for model-based outputs - given model simplifications or uncertainties in assumed parameters such as the ground height, environment friction, or inertial parameters, a learned layer could accordingly adjust the model-based torques. MPC formulations are typically sensitive to parameter inputs such as noise or contact timing, and mismatches can quickly cause solutions to diverge. RL policies, on the other hand, can be trained to be robust to these uncertainties. A learned, adaptive layer could retain the interpretability and tunability of the core model-based controller, while adapting to erroneous model assumptions or changing environmental conditions.

From the perspective of reinforcement learning, the fixed control prior in the residual architecture can be considered as a "warm start" for the policy. In the RL setting, we can freely design any number of sparse, nondifferentiable, and/or complex rewards as desired, and with the residual architecture, initialize the search for a satisfactory optimal policy with a model-based controller. By starting "closer" to an optimal policy, training is more sample efficient as the control prior guides the agents towards desirable regions of state space [23].

However, embedding MPC directly into a learning pipeline introduces a severe computational bottleneck. Training residual policy would require solving high-dimensional optimizations for each of the thousands of simulated agents necessary for RL. While prior work has explored training residual policies with other underlying controllers, including linear-quadratic regulators (LQRs), pretrained policies, and instantaneous whole-body controllers [23–28], these are much smaller problems, operating only on a single timestep. An MPC formulation for the same system optimizes over the entire prediction horizon while enforcing dynamics constraints, dramatically increasing the problem dimensionality and complexity. Due to the prohibitive computation required, relatively few works have explored training a policy alongside an MPC controller.

Jenelten et al. [20] accomplished this by leveraging large scale CPU clusters to evaluate many instances of the MPC in parallel, but this was only possible by solving the MPC at 2-3 Hz during training. With the additional computational overhead of transferring data between the CPU and GPU, training required weeks for full convergence. The proposed controller was also hierarchical rather than residual, where the policy was trained to track the predicted MPC states and foot commands rather than modify them in real-time. In contrast, our work investigates the effect of including concurrent, real-time MPC within an RL training loop at the torque-level.

Instead of performing MPC solves and RL iterations on separate devices, recent parallelization tools such as `CusADi` and `cuDSS` can be leveraged to efficiently solve many MPC instances concurrently on the GPU so that all memory is shared, eliminating any overhead from data transfer [29, 30]. This enables high-frequency MPC solutions to be integrated into the RL training loop without the prohibitive latency and memory transfer costs observed in prior work.

For our work, we study the effects of pairing a residual policy architecture with a whole-body MPC controller, where the outputs of a trained network and predictive controller are directly blended at the torque-control level, as shown in Fig. 2. We introduce a kinodynamic whole-body MPC formulation that is parallelized for high-frequency GPU evaluation and is solved concurrently with the residual network at 100 Hz in IsaacGym [31]. To the authors' knowledge, this represents the first work for legged systems embedding high-frequency MPC at the torque level within an RL training loop. With this setup, the residual policy outperforms both standalone MPC and baseline end-to-end RL controllers.

This residual training exhibits substantially higher sample efficiency and converges to a greater asymptotic reward than end-to-end RL. By providing a strong MPC control prior, the residual architecture biases the policy towards realistic and desirable locomotion behaviors, whereas end-to-end RL tends to exploit simulation idiosyncrasies and produce aphysical

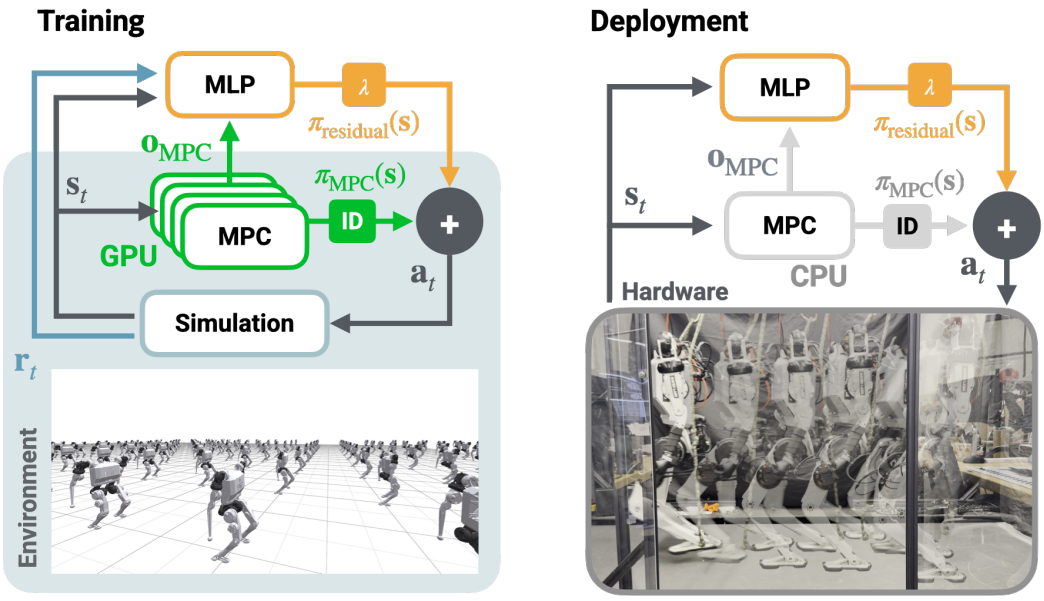

Fig. 2. Diagram of the proposed residual policy architecture during training and deployment. Both the multilayer perceptron (MLP) policy and MPC are evaluated at the control frequency of 100 Hz, and are blended downstream with a scaling factor of λ on the policy output. The MPC solution is converted into desired torques, joint positions, and joint velocities with the inverse dynamics (ID) of the system. We efficiently evaluate the MPC controller across thousands of instances on the GPU for concurrent training with the residual network.

motions. This substantially reduces the need for the arduous reward engineering and tuning that typically burden standard RL training.

From a performance standpoint, the residual policy markedly extends the range of trackable velocity commands beyond the capabilities of the baseline MPC, as shown in Fig. 1. It also learns to modify the MPC outputs to satisfy complex constraints such as self-collision, which are challenging to encode in model-based formulations without system-specific heuristics. For instance, prior work has relied on constraining distances between geometric primitives throughout the MPC horizon, or using lower-level whole-body optimizations to prevent collision [5, 32]. With RL, it is much easier to define sparse, non-differentiable rewards, and learn to avoid self-collision through data-driven experience. Additionally, we find that the residual policy exhibits zero-shot adaptability to unseen changes in the requested gait and uneven terrain. While MPC fails under contact plans with double stance or flight phases, the residual policy is able to track these trajectories reliably. Similarly, over uneven terrain, the MPC baseline quickly fails due to incorrect assumptions about contact timings, whereas the residual policy successfully navigates the variations of the ground.

Finally, we examine how the learned corrections from the residual network interact with the MPC prior, and interpret the residual actions with respect to the concurrent MPC torques. Interestingly, we find that the residual torques are generally antagonistic to the MPC torques, particularly shortly before a planned touchdown contact. Examining these corrective torques suggests that the residual policy heavily engages the ankle near touchdown and takeoff. We hypothesize that this behavior may explain the robustness present in learned locomotion policies with respect to making and breaking contact, and could serve as a heuristic for designing controllers in the future.

We summarize our contributions as follows:

- Embedding a GPU-parallelized, high-frequency MPC controller into RL environments for in-the-loop training.
- Analyzing how the MPC control prior guides and interacts with the residual network during locomotion.
- Demonstrating zero-shot adaptation of the residual policy to changes in the MPC parameters.
- Validating the residual policy on hardware for the MIT Humanoid [33].

II. Background

A. Model Predictive Control

MPC is a model-based control technique that optimizes a cost function over a finite horizon, subject to system dynamics and constraints. By repeatedly solving and updating this optimization problem at high frequency, the controller can adapt its actions to changing conditions and disturbances.

We denote the discrete, finite-time optimal control problem generally as

$$\begin{aligned} \min_{\mathbf{x}[\cdot],\mathbf{u}[\cdot]} \quad & l_T(\mathbf{x}_T,\mathbf{u}_T) + \sum_{i=0}^{T-1} l_i(\mathbf{x}_i,\mathbf{u}_i) \\ \text{s.t.} \quad & \mathbf{x}_0 = \bar{\mathbf{x}}_0 \\ & \mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i,\mathbf{u}_i), \quad i = 0,\dots,T-1 \\ & \mathbf{g}_i(\mathbf{x}_i,\mathbf{u}_i) \le 0, \quad i = 0,\dots,T \end{aligned} \tag{1}$$

where $\mathbf{x}[\cdot] \in \mathbb{R}^{n_x \times T}$ and $\mathbf{u}[\cdot] \in \mathbb{R}^{n_u \times T}$ are the state and control trajectories with initial condition $\bar{\mathbf{x}}_0$ and $\mathbf{f} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_x}$ is the nonlinear dynamics of the system. The system is constrained by $\mathbf{g} : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{m}$, and optimized

for the costs $l : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ summed across the timesteps i.

This nonlinear program (NLP) can be solved effectively with sequential quadratic programming (SQP) approaches [34]. By iteratively solving subproblems of the original NLP with a quadratic model of the cost and subject to linearized constraints, initial guesses to (1) can quickly converge to an optimal solution.

Under computational limits, the so-called *real-time iteration* scheme is an effective technique to quickly approximate an optimal solution [35]. The scheme involves strictly limiting the number of subproblems solved, directly trading optimality for responsiveness. For real-time systems with fast dynamics, solving even a single subproblem can be sufficient for stable closed-loop performance while greatly reducing computational effort [29, 35].

B. Residual Policy Learning

Standard deep reinforcement learning methods are initialized with arbitrary policies π that are trained to maximize the expected return of a task, given as

$$J(\pi) = \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\right], \tag{2}$$

where $\tau = (s_0, a_0, s_1, a_1, \ldots)$ is a state-action trajectory sampled from π, γ is the discount factor, and s_t, a_t, and $r(s_t, a_t)$ are the state, action, and reward received at time t respectively.

A residual network (ResNet) is a neural network with a "shortcut" or "skip" connection, that allows the input to be added directly to the output of a series of layers [36]. When the output of a desired function is close to the input, a residual network structure can significantly improve the convergence speed and accuracy of training in a supervised context. As one example, discrete dynamical systems are well-suited for a residual structure, as the evolution of the state is typically a small perturbation from the previous state, $x_{k+1} = x_k + f(x_k)$. This has been explored deeply in the context of *physics-informed neural networks* (PINNs), where the residual architecture can be used to learn complex dynamics such as fluid flow or chemical reactions [37, 38].

This idea has been extended to learning residuals of *policies*, rather than individual inputs - given a nominal controller π_0, a residual policy π_{residual} can be trained to maximize (2), where trajectories are drawn from samples of $\pi = \pi_0 + \pi_{\text{residual}}$. This approach has been shown to substantially improve sample efficiency, convergence speed, and asymptotic performance of policies in a variety of tasks [23, 25, 26]. Here, we focus on adapting and improving locomotion with this architecture, aiming to augment the performance of MPC with a learned residual layer, as shown in Fig. 2.

III. MPC Formulation and Parallelization

For designing MPC controllers in general, care needs to be taken to balance model fidelity with computational constraints. A formulation equipped with accurate dynamics and detailed costs/constraints may be severly limited by the time required for a solution, limiting its replanning rate and overall responsiveness. Conversely, a simplified model may be solved quickly, but could make unrealistic predictions that lead to poor or limited closed-loop performance. This consideration is especially important for GPU parallelization, where a learning iteration requires solving many MPC problems for each trajectory rollout in a batch - complex formulations can greatly increase training times, requiring days or even weeks for policies to converge [19, 20].

Fig. 3. Visualized predictions for the MPC formulation (distances are adjusted for visual clarity). A Bezier curve is defined by the peak swing height, takeoff velocity, and touchdown velocity. This curve is parametrized by the contact phase to output desired foot heights.

The equations of motion for a floating-base robot with generalized coordinates $\mathbf{q} \in \mathbb{R}^n$ and generalized velocities $\dot{\mathbf{q}} \in \mathbb{R}^n$ can be written as

$$\mathbf{M}(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}(\mathbf{q})^{\intercal}\,\mathbf{F} + \mathbf{S}^{\intercal}\boldsymbol{\tau}, \tag{3}$$

where $\mathbf{M}(\mathbf{q}) \in \mathbb{R}^{n\times n}$ is the mass matrix, $\mathbf{h}(\mathbf{q}, \dot{\mathbf{q}}) \in \mathbb{R}^n$ is the vector of Coriolis, centrifugal, and gravitational forces, $\mathbf{J}(\mathbf{q}) \in \mathbb{R}^{3n_c\times n}$ is the contact Jacobian for n_c contact points, $\mathbf{F} \in \mathbb{R}^{3n_c}$ is the vector of contact forces, $\mathbf{S} \in \mathbb{R}^{n_j\times n}$ is a selection matrix mapping actuated joint torques to generalized forces, and $\boldsymbol{\tau} \in \mathbb{R}^{n_j}$ is the vector of actuated joint torques. These system dynamics are the primary bottleneck for MPC problems, and many formulations focus on reducing its complexity to be amenable for real-time control [2–5, 7]

Our system is the MIT Humanoid [33], weighing roughly 25 kg in total with ten actuated joints for the legs and eight for the arms. For this work, we only use proprioceptive measurements from the internal IMU and joint encoders, and do not rely on contact sensors or cameras for locomotion. We define $n_c = 4$ contact points for our system, corresponding to the right toe, right heel, left toe, and left heel for the contact Jacobians.

A. Formulation

For our work, we formulate a "floating-base kinodynamic" optimization as a balance between overly simplified single rigid-body models and a computationally prohibitive whole-body MPC formulation. Only the dynamics of the floating base are considered by multiplying the equations of motion with a selection matrix $\mathbf{S}_{base} = [\mathbf{I}_{n_b\times n_b}\ \mathbf{0}_{n_b\times n_j}]$, where n_b

is the number of base coordinates and n_j is the number of joints. This reduces the equations of motion to

$$\mathbf{M}_b(\mathbf{q})\,\ddot{\mathbf{q}} + \mathbf{h}_b(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{J}_b(\mathbf{q})^\intercal\,\mathbf{F}, \tag{4}$$

which serves as a path constraint of dimension n_b on $\mathbf{q}$, $\dot{\mathbf{q}}$, $\ddot{\mathbf{q}}$,and $\mathbf{F}$[1].

With this dynamics constraint, we choose to optimize over the generalized coordinates $\mathbf{q}$, velocities $\dot{\mathbf{q}}$, and ground reaction forces $\mathbf{F}$, over a horizon length of $T = 12$. Writing these decision variables as $\mathbf{z} := [\mathbf{q}^\intercal\ \dot{\mathbf{q}}^\intercal\ \mathbf{F}^\intercal]^\intercal$, our MPC formulation is

$$\begin{aligned}
\min_{\mathbf{q}[\cdot],\dot{\mathbf{q}}[\cdot],\mathbf{F}[\cdot]} \quad & \sum_{i=0}^{T} (\mathbf{z}_i - \mathbf{z}_{i,des})^\intercal Q_{\mathbf{z}} (\mathbf{z}_i - \mathbf{z}_{i,des}) \Delta t_i \qquad (5)\\
\text{s.t.} \quad & \forall\ i = 0, \ldots, T-1\\
\text{(Initial state)} \quad & \mathbf{q}_0 = \bar{\mathbf{q}}_0,\\
& \dot{\mathbf{q}}_0 = \bar{\dot{\mathbf{q}}}_0,\\
\text{(Integration)} \quad & \dot{\mathbf{q}}_{i+1} = \dot{\mathbf{q}}_i + \Delta t_i\,\ddot{\mathbf{q}}_i,\\
& \mathbf{q}_{i+1} = \mathbf{q}_i + \Delta t_i\,\dot{\mathbf{q}}_{i+1},\\
\text{(Dynamics)} \quad & \mathbf{M}_b(\mathbf{q}_i)\,\ddot{\mathbf{q}}_i + \mathbf{h}_b(\mathbf{q}_i, \dot{\mathbf{q}}_i) = \mathbf{J}_b(\mathbf{q}_i)^\intercal\,\mathbf{F}_i,\\
& \ddot{\mathbf{q}}_i = (\dot{\mathbf{q}}_{i+1} - \dot{\mathbf{q}}_i)/\Delta t_i,\\
\text{(Contact)} \quad & \|\mathbf{F}_{x,y}\| \le \mu\,\mathbf{F}_z \cdot \boldsymbol{\phi}_i,\\
& \mathbf{J}(\mathbf{q}_i)\,\dot{\mathbf{q}}_i \cdot \boldsymbol{\phi}_i = \mathbf{0},\\
\text{(Swing)} \quad & \mathbf{r}(\mathbf{q}_i)_z = h_{i,des},\\
\text{(Joint limits)} \quad & \mathbf{q}_{i,joint} \in \mathcal{Q},\\
& \dot{\mathbf{q}}_{i,joint} \in \dot{\mathcal{Q}},
\end{aligned}$$

where $Q_{\mathbf{z}} \succeq 0$ is a diagonal weight matrix, $\mathbf{z}_{i,des}$ is a desired state, Δt_i is the timestep duration, $\bar{\mathbf{q}}_0$ and $\bar{\dot{\mathbf{q}}}_0$ are the current state of the robot, μ is the static coefficient of friction, $\boldsymbol{\phi}_i$ is the contact phase, $\mathbf{r}(\mathbf{q}_i)_z$ is the foot height, $h_{i,des}$ is the desired foot height, and $\mathcal{Q}$ and $\dot{\mathcal{Q}}$ are the feasible set of joint position and velocity limits.

For the desired state trajectory $\mathbf{z}_{des}[\cdot]$, we set the desired vertical ground reaction forces equal to the system's weight divided by the number of contacts. For $\mathbf{q}_{i,des}$ and $\dot{\mathbf{q}}_{i,des}$, we use a simple extrapolation heuristic based on the MPC command $\mathbf{c} \in \mathbb{R}^4$. The command consists of the desired height c_h, linear velocities c_{v_x} and c_{v_y}, and yaw angular velocity c_{ω_z}, and we compute the desired trajectories as follows:

$$q^{h}_{i,des} = c_h, \tag{6}$$
$$q^{\theta_z}_{i,des} = q^{\theta_z}_{i-1,des} + \Delta t_i c_{\omega_z}, \tag{7}$$
$$\dot{q}^{v_x}_{i,des} = c_{v_x}, \tag{8}$$
$$\dot{q}^{v_y}_{i,des} = c_{v_y}, \tag{9}$$
$$\dot{q}^{\omega_z}_{i,des} = c_{\omega_z}, \tag{10}$$

for $i = 0, \ldots, T-1$. The desired joint positions are set to a nominal position for each timestep, and all other desired quantities are set to zero. To avoid sensitivity to absolute position drift, the weights on the x and y states are set to zero as well.

We design a fixed contact schedule with phase variable $\boldsymbol{\phi} \in \mathbb{R}^{n_c}$ for each contact point. For our work, we set the period to be 0.8s and enforce contact switches from stance to swing when $\boldsymbol{\phi} = 0.5$, but these can easily be adjusted to command double stance or flight phases, as described in [41]. This phase variable is integrated through the predicted horizon of the MPC, and each node is assigned a corresponding contact state based on whether the phase is larger or smaller than the switching value. Our controller fixes a phase offset of 0.5 between the left and right contact points, commanding the humanoid to step flat footed without heel-toe transitions.

The foot height is constrained to a curve $h_{i,des}$ based on the duration of swing, as shown in Fig. 3. This curve is formulated as a fifth-order Bezier curve with the following conditions:

$$B(t_{sw} = 0.0) = 0, \tag{11}$$
$$B(t_{sw} = 1.0) = 0, \tag{12}$$
$$B(t_{sw} = 0.5) = z_{\text{swing}}, \tag{13}$$
$$\dot{B}(t_{sw} = 0.0) = v_{\text{to}} \tag{14}$$
$$\dot{B}(t_{sw} = 1.0) = v_{\text{td}} \tag{15}$$

where t_s is the time during swing, scaled to be between 0 and 1, and v_{to}, v_{td}, and z_{swing} are the desired takeoff velocity, touchdown velocity, and swing height respectively.

The formulation in (5) sacrifices the ability to reason about individual joint dynamics and torques, limiting the controller's predictions for dynamic motions close to the actuation limits of the robot. Additionally, we do not have any bounds on the floating base states $\mathbf{q}_{i,base}$ and $\dot{\mathbf{q}}_{i,base}$, or self-collision constraints as in [5], as these can further increase computational complexity. With the residual policy architecture, we show that these limitations can be overcome by the learned policy, and the controller combined is capable of avoiding self-collisions and termination.

B. Solver Strategy

As discussed in subsection II-A, we use the *real-time iteration* scheme to solve the nonlinear optimization with a single SQP iteration per MPC solve. This reduces (5) to a single quadratic program (QP) to solve, which can be approached with various active-set, interior-point, or alternating direction implementations [42–45].

We adopt the techniques in the Operator-Splitting Quadratic Program (OSQP) library [44] to solve (5). Compared to other QP solution methods, the alternating direction method of multipliers (ADMM) used in OSQP requires the fewest symbolic computations per iteration. Interior point and active-set methods modify the KKT matrix at each iteration (from updating constraint Jacobians and/or the barrier parameter), requiring a full refactorization of the system at each step. The KKT matrix is seldom modified during solves with ADMM, and subsequent iterations consist only of simple projection and matrix-vector operations.

[1]Projecting (4) to the center of mass frame is exactly the *centroidal dynamics* formulation used in prior work [6, 7, 39, 40]. We found that there was little difference in closed-loop performance between these formulations, while the base dynamics formulation was slightly faster to compute.

The most expensive step is factorizing and solving the KKT system of the QP subproblem, given as

$$\underbrace{\begin{bmatrix}\mathbf{P}+\sigma\mathbf{I} & \mathbf{A}^\intercal \\ \mathbf{A} & -\rho^{-1}\mathbf{I}\end{bmatrix}}_{\mathbf{A}_K}\mathbf{v}_K = \underbrace{\begin{bmatrix}\sigma\mathbf{x}_{\mathrm{QP}}-\mathbf{q} \\ \mathbf{z}_{\mathrm{QP}}-\rho^{-1}\mathbf{y}_{\mathrm{QP}}\end{bmatrix}}_{\mathbf{b}_K}, \tag{16}$$

where $\mathbf{P}$, $\mathbf{q}$, and $\mathbf{A}$ are the cost Hessian[2], the cost Jacobian, and the constraint Jacobian of the MPC formulation (5) respectively, and $\mathbf{I}$ is the identity matrix. x_{QP} and y_{QP} are the current primal and dual variables of the QP subproblem, σ is the ADMM penalty parameter, and ρ is the dual variable scaling factor.

To solve the system, the first step involves factorizing the KKT matrix $\mathbf{A}_K$

$$\mathbf{L}_K\mathbf{D}_K\mathbf{L}_K^\intercal = \mathbf{A}_K, \tag{17}$$

into a lower-triangular matrix $\mathbf{L}_K$ and a diagonal matrix $\mathbf{D}_K$. For a single QP solve, this LDL factorization only needs to be performed once. The subsequent iterations of the QP are performed by updating $\mathbf{b}_K$, and performing back-substitution to solve for $\mathbf{v}_K$:

$$\mathbf{L}_K\mathbf{D}_K\mathbf{L}_K^\intercal\,\mathbf{v}_K = \mathbf{b}_K, \tag{18}$$

$$\mathbf{v}_K = \mathbf{L}_K^{-\intercal}\mathbf{D}_K^{-1}\mathbf{L}_K^{-1}\mathbf{b}_K. \tag{19}$$

Evaluating (19) is computationally inexpensive, requiring only diagonal scaling and forward/backward substitution.

After the system is solved for $\mathbf{v}_K$, $\mathbf{b}_K$ and the primal/dual variables are updated with simple projection and linear algebra operations. These operations and the linear solve (19) are then repeated for a fixed number of QP iterations N_{QP}, or until convergence. For details on these computations, readers are referred to Section 3.2 in [44].

C. MPC Computation

Overall, the MPC controller is evaluated as follows. First, an initial guess $\mathbf{z}[\cdot]$ is computed based on the current state of the robot and the desired command with f_{init}. This consists of some nominal pose repeated for every timestep of the trajectory, with all other quantities set to zero. The parameters of the MPC problem $\boldsymbol{\theta}_{\mathrm{MPC}}$ such as the desired states, contact schedule, and commands are then updated with f_{param}, consisting of (6)-(10) and (11)-(15).

Next, the KKT system is constructed with f_{KKT}, which computes the necessary Hessians and Jacobians of (5) and assembles them into the form shown in (16). A scaling step is performed with the Ruiz equilibration method f_{Ruiz}, which improves the conditioning of the KKT matrix $\mathbf{A}_K$ [46]. The KKT matrix is then factorized ((17)) with the `cuDSS` library, which will be discussed further in subsection III-D [30]. The factorized components of the KKT matrix are used to compute the QP iterations with repeated linear solves and updates with f_{ADMM} ((18)-(19)).

After N_{QP} iterations, the solution $\Delta\mathbf{z}_{N_{\mathrm{QP}}}$ to the QP is added to the initial guess $\mathbf{z}_{\mathrm{init}}[\cdot]$ with an optional line search (we simply keep $\alpha = 1$). Finally, the spatial recursive Newton-Euler algorithm (RNEA) [47] is used to compute feedforward torques from the first timestep of the MPC solution, $\mathbf{q}_{\mathrm{MPC}} = \mathbf{q}^*[0]$, $\dot{\mathbf{q}}_{\mathrm{MPC}} = \dot{\mathbf{q}}^*[0]$, and $\mathbf{F}_{\mathrm{MPC}} = \mathbf{F}^*[0]$. The inverse dynamics f_{RNEA} are computed as

$$\begin{aligned}\boldsymbol{\tau}_{\mathrm{RNEA}} = \mathbf{M}(\mathbf{q}_{\mathrm{MPC}})\,\ddot{\mathbf{q}}_{\mathrm{MPC}} + \mathbf{h}(\mathbf{q}_{\mathrm{MPC}}, \dot{\mathbf{q}}_{\mathrm{MPC}}) \\ - \mathbf{J}(\mathbf{q}_{\mathrm{MPC}})^\intercal\,\mathbf{F}_{\mathrm{MPC}}\,.\end{aligned} \tag{20}$$

The commanded joint torques for the MPC is then sent to the platform as

$$\boldsymbol{\tau}_{\mathrm{MPC}} = \mathbf{K}_p(\mathbf{q}_{\mathrm{MPC}} - \mathbf{q}) + \mathbf{K}_d(\dot{\mathbf{q}}_{\mathrm{MPC}} - \dot{\mathbf{q}}) + \boldsymbol{\tau}_{\mathrm{RNEA}}. \tag{21}$$

A summary of the necessary computations is shown in Alg. 1.

Algorithm 1 GPU-Parallelized MPC via ADMM

in parallel:
$\mathbf{z}_{\mathrm{init}}[\cdot] \leftarrow f_{\mathrm{init}}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\phi})$ ▷ Initial guess
$\boldsymbol{\theta}_{\mathrm{MPC}} \leftarrow f_{\mathrm{param}}(\mathbf{q}, \dot{\mathbf{q}}, \mathbf{c}, \boldsymbol{\phi})$ ▷ MPC parameter update
$\mathbf{A}_K, \mathbf{b}_K \leftarrow f_{\mathrm{KKT}}(\mathbf{z}_t[\cdot], \boldsymbol{\theta}_t)$ ▷ Form KKT system
$\hat{\mathbf{A}}_K, \hat{\mathbf{b}}_K \leftarrow f_{\mathrm{Ruiz}}(\mathbf{A}_K, \mathbf{b}_K)$ ▷ Ruiz equilibration
$\mathbf{L}_K, \mathbf{D}_K \leftarrow \mathtt{cuDSS}_{\mathrm{fac}}(\hat{\mathbf{A}}_K)$ ▷ Eq. (17)
for $i \leftarrow 0$ to N_{QP} **do** ▷ QP iterations
  $\mathbf{v}_K \leftarrow \mathtt{cuDSS}_{\mathrm{solve}}(\hat{\mathbf{b}}_K, \mathbf{L}_K, \mathbf{D}_K)$ ▷ Eq. (19)
  $\Delta\mathbf{z}_i, \hat{\mathbf{b}}_K \leftarrow f_{\mathrm{ADMM}}(\mathbf{v}_K)$
end for
$\mathbf{z}^*[\cdot] \leftarrow \mathbf{z}[\cdot] + \alpha\Delta\mathbf{z}_{N_{QP}}$ ▷ Line search
$\boldsymbol{\tau}_{RNEA}, \mathbf{q}_{MPC}, \dot{\mathbf{q}}_{MPC} \leftarrow f_{\mathrm{RNEA}}(\mathbf{z}^*[\cdot])$
return $\boldsymbol{\tau}_{MPC}, \mathbf{q}_{MPC}, \dot{\mathbf{q}}_{MPC}, \mathbf{z}^*[\cdot]$

D. Solver Parallelization

To compute the MPC solution in parallel, we use the `CusADi` library [29], which is a code-generation framework for generating parallelized GPU kernels.

As discussed in [29], `CasADi` includes native symbolic LDL factorization and solve routines that could be used to address (17) and (19) [48]. However, the code-generated kernel for factorizing the KKT matrices requires millions of instructions, nearing CPU memory limits for compilation. Instead, we use NVIDIA's **D**irect **S**parse **S**olve (`cuDSS`) library to address the factorization and linear solve steps, which is optimized for parallel GPU computation. The library contains routines for sparse LDL factorization of indefinite matrices and linear solves, suited for the KKT system in (16) [30]. The expensive factorization step only needs to be computed once per MPC solve, while the inexpensive linear solves can reuse the factorized matrices to solve for the primal and dual variables in parallel.

The remaining auxiliary functions, highlighted in green in Alg. 1, are generated for the GPU with `CusADi` and are executed in parallel for each MPC solve. The number of instructions range between 10^3 to 10^6 for these functions, and are straightforward to generate and compile.

IV. Residual Architecture

We investigate the use of residual policy learning to improve and adjust the outputs of an MPC locomotion controller for

[2]Note that the cost of (5) is in linear least-squares form, and the Gauss-Newton Hessian approximation is equivalent to the full Hessian.

the MIT Humanoid [33]. As shown in Fig. 2, we evaluate the MPC and residual policy at the same control frequency of 100 Hz for both training and deployment. The MPC is parallelized on the GPU for training, and deployed on the CPU with OSQP for online evaluation. The residual network is a simple multilayer perceptron (MLP) with three layers, each with 256 exponential linear unit (ELU) nodes. The policy observes the full state of the robot, contact phase variables, and the value from the MPC, denoted as $\mathbf{o} = [\mathbf{p}^\intercal \ \boldsymbol{\theta}^\intercal \ \mathbf{q}_j^\intercal \ \boldsymbol{\omega}^\intercal \ \mathbf{v}^\intercal \ \dot{\mathbf{q}}_j^\intercal \ \boldsymbol{\phi}^\intercal \ V_{\text{MPC}}]^\intercal \in \mathbb{R}^{54}$, and outputs actions for the legs $\mathbf{a} \in \mathbb{R}^{10}$. To reduce reward shaping complexity and tuning, we chose to limit our scope to adapting the leg torques only for locomotion. Although the residual networks were tested with arm actions as well, we observed only minimal deviations from the MPC controller's predictions for the upper body. Additionally, simple box constraints on the arm joints were sufficient to prevent self-collision from the arms through the MPC horizon. A more thorough treatment on learning adaptive arm motion is given in [49].

Surprisingly, the residual policy is still able to learn and reliably adjust to the MPC controller despite being given very limited information about its outputs. While we originally allowed the network to observe the output torques and predicted states, we found that this made sim-to-sim transfer and sim-to-real transfer less reliable, as the distributions of these quantities were sensitive to the IsaacGym simulator.

A. Reward Design

For this work, we study the role of the baseline MPC controller as a *prior* for guiding the policy towards favorable states, rather than as an expert actor to be imitated. Consequently, we limit the scope of our experiments to "standard" RL rewards given directly from environment transitions to clearly isolate the effects of the residual architecture[3]. We wish to demonstrate that the residual policy allows the humanoid to learn behaviors beyond the capabilities of the designed MPC controller without overriding its outputs entirely. To this end, we design a set of reward functions that are generally aligned with the MPC objectives, but include nondifferentiable and sparse terms that would be difficult to directly optimize for in an online setting, such as self-collision avoidance or termination conditions. We give self-collision penalties when contact forces are detected between the humanoid's links, and termination conditions when the floating base has large deviations in velocities, orientation, or height, as in [52]. A full list of the rewards used is detailed in Table I.

The set of rewards given to the system is intentionally minimal. We avoid giving overly specific rewards such as foot guidance, air time, or contact-scheduling terms to study how the MPC can guide policy training towards favorable states with minimal tuning, as detailed in subsection V-B. More importantly, we seek to move away from fitting complex rewards to desirable locomotion behavior in favor of using physically-grounded models that can plan and adapt behavior online.

[3]As opposed to adversarial or distribution-imitating frameworks [50, 51]. Future work could investigate the effect of other reward paradigms attached to a residual architecture.

TABLE I
REWARD FUNCTIONS AND THEIR WEIGHTS.

Reward Name	Weight	Function
Lin. vel. tracking	10.0	$\exp\left(-\left\|\frac{\mathbf{c}_{x,y}-\mathbf{v}_{x,y}}{1+\lvert\mathbf{c}_{x,y}\rvert}\right\|_2^2/\sigma\right)$
Ang. vel. tracking	5.0	$\exp\left(-\lVert\mathbf{c}_\omega-\boldsymbol{\omega}_z\rVert_2^2/\sigma\right)$
1st order action rate	-1e-3	$\lVert(\mathbf{a}_t-\mathbf{a}_{t-1})/\Delta t\rVert_2^2$
2nd order action rate	-1e-4	$\lVert(\mathbf{a}_t-2\mathbf{a}_{t-1}+\mathbf{a}_{t-2})/\Delta t\rVert_2^2$
Torques	-1e-4	$\lVert\boldsymbol{\tau}\rVert_2^2$
Orientation	1.0	$\exp\left(-\lVert\mathbf{g}_{x,y}\rVert_2^2/\sigma\right)$
Height	1.0	$\exp\left(-\lVert\mathbf{c}_z-\mathbf{p}_z\rVert_2^2/\sigma\right)$
Joint regularization	1.0	$\frac{1}{n_j}\sum_{j=1}^{n_j}\lVert\mathbf{q}_j-\hat{\mathbf{q}}_j\rVert_2^2$
Self-collision	-1.0	$\mathbb{1}_{\text{collision}}$
Termination	-100	$\mathbb{1}_{\text{terminate}}$

B. Blending Strategy

It is important that the residual policy has little effect on the baseline controller when initialized [23–25]. This is to ensure that the residual controller is aligned with the control prior at the start of training, instead of destabilizing it with random outputs. If the action space of the policy output is the same as that of the nominal policy, this can easily be achieved by initializing the policy network to have weights and biases of zero (e.g., a position controlled manipulator, with position-space actions for the residual policy). However in our case, our baseline MPC outputs torques, while policies are typically trained with joint setpoint actions. To study the effect of the action space on the residual architecture, we consider three different strategies for blending these outputs:

1) **Joint action, joint blending:**

$$\boldsymbol{\tau} = \mathbf{K}_p(\mathbf{q}_{\text{cmd}}+\lambda\mathbf{a}-\mathbf{q})+\mathbf{K}_d(\dot{\mathbf{q}}_{\text{cmd}}-\dot{\mathbf{q}})+\boldsymbol{\tau}_{\text{cmd}}, \quad (22)$$

2) **Joint action, torque blending:**

$$\boldsymbol{\tau}_{\text{residual}} = \mathbf{K}_p(\mathbf{a}+\hat{\mathbf{q}}-\mathbf{q})-\mathbf{K}_d\dot{\mathbf{q}} \quad (23)$$

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{\text{MPC}}+\lambda\boldsymbol{\tau}_{\text{residual}}, \quad (24)$$

3) **Torque action, torque blending:**

$$\boldsymbol{\tau} = \boldsymbol{\tau}_{\text{MPC}}+\lambda\mathbf{a}, \quad (25)$$

where $\mathbf{a}$ is the action output from the residual policy, $\hat{\mathbf{q}}$ are default joint positions, and $\boldsymbol{\lambda}$ is a scalar blending factor.

In the first case, the policy modifies the joint setpoint command from the MPC torques computed in (21). Here, a zero-initialized policy will not affect the MPC outputs, but the policy is relative to $\mathbf{q}_{\text{cmd}}$ which is non-stationary. In the second case, the policy outputs joint setpoints relative to default joint positions, and the final torques from the MPC and residual policy are blended together. However, the residual policy will have non-zero torques even if the network is zero-initialized. Consequently, we introduce a weighting parameter to control the initial influence of the policy. In the third case, the policy directly outputs torques, which are then combined with the MPC torques. We analyze how the choice of action space affects the residual policy's ability to learn from the MPC prior in subsection V-B

C. Training Procedure

We use the IsaacGym simulation environment to gather simulation rollouts of our residual controller [31]. Due to GPU VRAM limitations from the parallelized MPC, we limit the number of parallel environments to 2048. To train the policy, we use Proximal Policy Optimization (PPO) [53] with a clipped objective and generalized advantage estimation (GAE) [54]. The same PPO hyperparameters from [52] are used for all experiments, and a summary of the training procedure is shown in Alg. 2. Note that any RL algorithm could be used to optimize the policy, but we choose PPO for its simplicity and effectiveness in continuous control tasks.

Algorithm 2 Residual Policy Training

Require: Initial policy π_θ, π_{MPC}

for $n = 0, \dots, N-1$ episodes **do**
  Initialize exporation noise $\mathcal{N}$
  Sample initial state $\mathbf{s}_0 \sim \mathcal{D}$
  for $t = 0, \dots, H-1$ timesteps **do**
    Compute MPC: $\mathbf{z}_t^*[\cdot], \boldsymbol{\tau}_{\text{MPC},t} \leftarrow$ Alg. 1
    Sample policy: $\mathbf{a}_t \sim \pi_\theta(\mathbf{s}_t, \mathbf{z}_t^*[\cdot]) + \mathcal{N}_t$
    Blend outputs: $\boldsymbol{\tau}_t \leftarrow f_{\text{blend}}(\boldsymbol{\tau}_{\text{MPC},t}, \mathbf{a}_t)$ ▷ IV-B
    Simulate envs.: $\mathbf{s}_{t+1} \sim \mathcal{T}(\mathbf{s}_t, \boldsymbol{\tau}_t)$
    Store transitions: $\mathcal{R} \leftarrow (\mathbf{s}_t, \boldsymbol{\tau}_t, \mathbf{s}_{t+1})$
  end for
  Sample transitions: $(\mathbf{s}, \boldsymbol{\tau}, \mathbf{s}) \sim \mathcal{R}$.
  Optimize policy with RL: $\pi_{\theta_{t+1}}$. ▷ PPO
end for

V. Results

We study the performance of the residual architecture for learning locomotion tasks on the MIT Humanoid [33]. All reported results are collected on a desktop computer equipped with an NVIDIA RTX 3090 GPU and a 10th Gen. Intel Core i9-10900K CPU. Ablation studies and various sweeps are evaluated on an NVIDIA A100 GPU.

We consider three controllers for comparison.

- **MPC**: The fixed MPC controller described in subsection III-A.
- **Residual**: The proposed residual architecture.
- **End-to-End**: A controller learned with conventional RL end-to-end.

Unless specified otherwise, each of these controllers is given the same rewards, range of commands, initial configurations, disturbances, domain randomization, etc. during training to ensure a fair comparison.

To clearly analyze the effect of incorporating model-based priors into RL, we limit our scope to these three controllers. While many state of the art controllers exist for humanoid platforms (both learned and model-based), we wish to isolate the relative benefits over strictly model-based and strictly learning-based benchmarks, which could transfer to more complex training/controller architectures. Therefore, we limit our scope to studying the advantages of the residual architecture for combining MPC and RL specifically, rather than benchmarking the absolute performance of the residual policy against all other possible humanoid controllers.

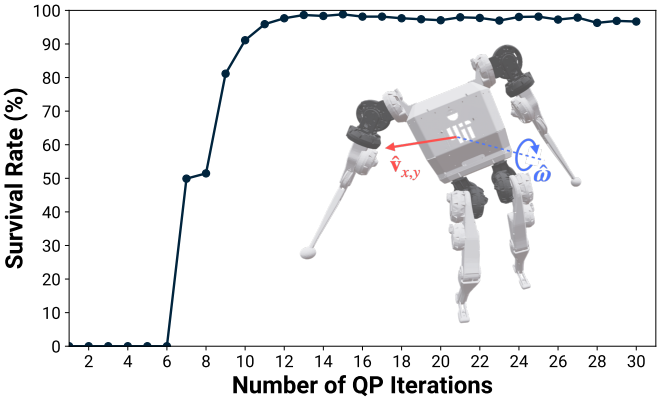

Fig. 4. Plot of survival rate vs. number of QP iterations for MPC environments subject to randomized initial velocities in the range $\|\hat{\mathbf{v}}_{x,y}\| \leq 0.5$ m/s and $\|\hat{\boldsymbol{\omega}}\| \leq 0.5$ rad/s. Beyond a certain number of iterations, the marginal benefit is negligble while increasing computation time.

A. GPU-Parallelization of MPC

We begin by characterizing the computations necessary for the MPC controller outlined in subsection III-A. From Alg. 1, the only way to reduce the total number of computations is by reducing N_{QP} - all other calculations are strictly necessary for evaluation. Consequently, we sweep the controller across a range of initial linear and angular base velocities while stepping with zero velocity command, and record the percentage of robots that survive termination (defined with simple bounds on height and self-collision within five seconds of the initial disturbance). As shown in Fig. 4, we find that the marginal benefit of additional QP iterations decreases significantly once the solution is "good enough" under these simple conditions. Conservatively, we set $N_{\text{QP}} = 25$ for the controller for the remainder of the work. During hardware deployment on the CPU, this can be evaluated between 200-300 Hz single-threaded.

To generate the parallelized controller, the `CasADi` objects for each of the functions in Alg. 1 must be unrolled and compiled as a `CUDA` kernel. This can be a significant bottleneck, as `CasADi` functions can easily contain millions of instructions, equating to a kernel with millions of lines of code to compile. By making use of `cuDSS`, we avoid compiling kernels for the KKT factorization and backsolve, the most computationally expensive steps. Table II presents details about the functions that were parallelized with `CusADi`. The most expensive functions to generate and evaluate are those that form the KKT matrices - namely, this consists of the cost Hessians and constraint Jacobians of the MPC formulation.

Next, we study the computational load incurred from the GPU-parallelized MPC. In Fig. 5, the speedup relative to CPU evaluation is shown. For the CPU evaluation time, we optimistically estimate perfect parallelization between the ten cores of the i9-10900K Intel CPU, and do not include the additional time necessary to transfer the data from the CPU to the GPU for training, which is the primary bottleneck [29]. Despite this, GPU evaluation is still roughly 2.5x faster than CPU evaluation for parallelization across 1,000 environments.

TABLE II
FUNCTIONS FOR MPC PARALLELIZATION ($N_{\text{QP}} = 25$).

Function	Num. instr.	Compile time (s)	Eval. time (ms)
f_{init}	9.64e2	36.6	2.35 ± 0.1
f_{param}	9.19e2	38.5	8.03 ± 0.4
f_{KKT}	1.05e6	1.91e3	27.46 ± 0.9
f_{Ruiz}	3.72e5	1.53e3	23.65 ± 0.8
f_{ADMM}	2.45e4	133.1	3.11 ± 0.2
f_{RNEA}	7.34e3	39.2	0.27 ± 0.1
$\texttt{cuDSS}_{\text{fac}}$	-	-	81.75 ± 1.7
$\texttt{cuDSS}_{\text{solve}}$	-	-	12.72 ± 0.4
Total	-	-	492.82 ± 2.8

For each simulation step, the majority of the evaluation time is spent performing ADMM iterations (70.7%). The total evaluation time could be further reduced by adjusting the N_{QP} parameter to be less conservative.

While the computations are efficiently parallelized with `CusADi`, solving thousands of optimization problems at each timestep of the simulation still incurs significant overhead compared to standard RL training. While the authors of [20] evaluate their MPC formulation at 2-3 Hz, we evaluate MPC at the same frequency as the policy (100 Hz) which greatly increases total training time. For 1,000 environments, a single simulation step takes roughly 0.5 seconds, as shown in Fig. 5. This scales linearly with the number of simulation steps performed per PPO iteration. For our work, we use 24 steps, corresponding to roughly 12 seconds per gradient update. This is approximately 8x slower than end-to-end RL on the same computer. However, we show in subsection V-B that the parallelized MPC prior significantly shapes the convergence of the policy, steering exploration towards more desirable behavior. Although training requires more time, the final policy can be more reliably guided than by painstakingly tweaking reward weights and environment parameters.

B. Training

With the fixed set of rewards from Table I, we first consider the effect of the proposed blending strategies. Specifically, we vary the action space of the residual policy (joint setpoint/torque) and the manner of blending (joint-space/torque-space). We train each of these residual policies with the same set of rewards and a zero-initialized policy network, as described in [23], and set the blending factor $\lambda = 0.1$.

As shown in Fig. 6, we find that the residual policy learns more effectively from a joint-space representation of its actions. Although the *joint-torque* training produces nonzero torque at the onset of training as in Equation 23, the difference to the *joint-joint* training is negligible. The torque space representation controls the joints at the acceleration level, making it more difficult for the policy to smoothly explore actions with high advantage. Additionally, the scale of torque-space actions are significantly larger than joint-space actions. With the same action and action rate regularization weights across the experiments, this could have affected the convergence of the *torque-torque* training.

Ultimately, we use the *joint-torque* strategy for blending the MPC and residual policy outputs. While both the *joint-joint* and *joint-torque* experiments demonstrated similar performance, in the event of a diverging MPC solution, the *joint-joint* strategy would output actions relative to potentially infeasible $\mathbf{q}_{\text{MPC}}$ setpoints. To avoid this potential failure mode, we use the *joint-torque* strategy to ensure reasonable policy outputs regardless of the MPC solution.

With the blending strategy established, we compare the reward curves for the residual policy against the MPC-only and RL-only baselines, shown in Fig. 7. While both the end-to-end and residual policies show significant improvements over

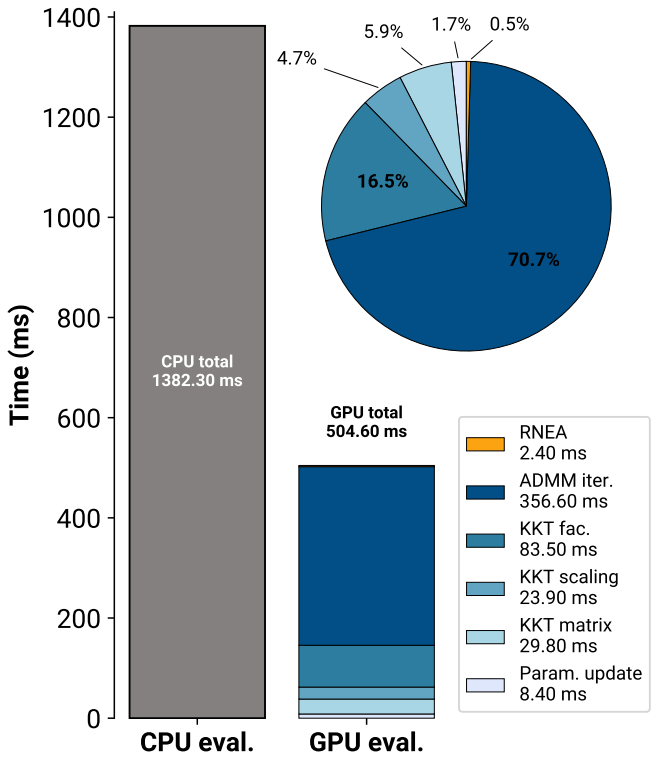

Fig. 5. Computation time required for evaluating the full MPC controller on the CPU and GPU across 1000 environments in IsaacGym. We optimistically estimate the required time for the CPU assuming 10 perfectly parallel cores, and do not include the time necessary for CPU-GPU data transfer. More detailed benchmarking results for `CusADi` on the CPU and GPU are shown in [29].

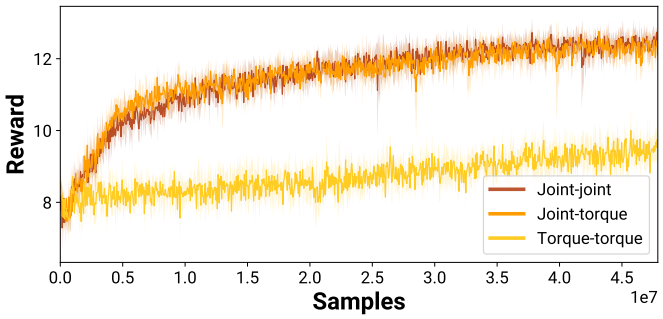

Fig. 6. Reward comparison of three different blending strategies between MPC and the residual policy. While there is little difference between joint-space action representations for the residual policy, a torque-space action representation performs significantly worse.

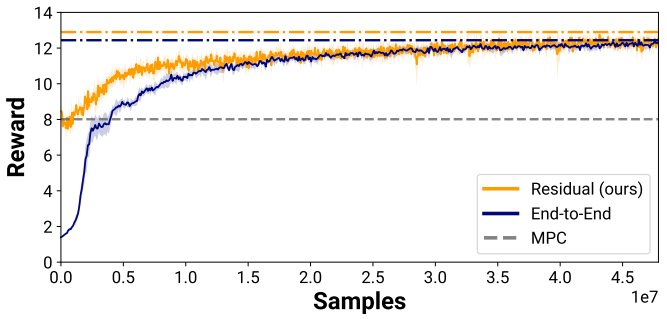

Fig. 7. The reward incurred during training between the residual and end-to-end controllers. The fixed MPC prior maintains a constant value as it contains no network to be optimized from PPO iterations. The residual architectures shows improvements in sample efficiency and asymptotic performance compared to the end-to-end training.

the reward incurred by the MPC-only controller, the residual training converges faster and to a higher asymptote than the end-to-end training. As discussed in [23, 25], the residual architecture allows the policy to leverage strong control priors by being initialized at the performance of the baseline MPC.

While the residual training exhibits improvements in sample efficiency compared to the end-to-end training, the wall clock time is significantly longer. For 1,000 iterations of PPO, end-to-end training only requires around 30 minutes, whereas the residual can range from three to four hours. Purely from the perspective of time to convergence, evaluating the parallelized MPC in the RL loop is far more expensive than standard end-to-end training.

However, the qualitative and quantitative differences in the converged policies are notable. As shown in Fig. 8, the end-to-end and residual controllers exhibit entirely different behaviors for locomotion. Despite being given the exact same rewards, environments, initialization conditions, and so on, the end-to-end policy learns to rapidly oscillate the ankle joint of the humanoid to "glide" across the floor, exploiting the physics of the environment. While this behavior may be feasible in the IsaacGym simulation, it is unlikely that this would transfer safely to hardware.

In contrast, the residual policy learns to stay close to the MPC solutions, exhibiting clear stepping behavior at the same frequency as the MPC. When we compare key characteristics between the two policies such as the joint velocities, torques, mechanical power, and vertical ground reaction force, we find that the residual policy has a smaller median and range overall. Despite being given the same rewards related to actuation (torque and action rate penalization), the residual policy demonstrates lower extremes in the experienced joint velocities, torques, mechanical power expended, and ground reaction forces. The inclusion of the MPC prior greatly affects policy training, such that even for the same set of rewards, the converged behavior ranges from physically unrealistic to viable for hardware deployment.

More broadly, RL can be sensitive to the reward signals, initialized states, curricula, and algorithm hyperparameters of the training setup. Algorithms like PPO theoretically can converge to global optima, but in practice, it is difficult to balance exploration and exploitation of the policy to achieve desirable behavior. The feedback loop of tuning a training procedure can also be time-consuming and unintuitive. As additional reward shaping terms are added to guide the policy, it becomes more difficult to parse the individual and possibly combinatoric effects from all the rewards.

Instead of engineering complex rewards and/or curricula to guide the policy towards desirable states, control priors can serve to initialize the policy "close to" a desired behavior. The residual architecture can be viewed as "warm start" in the search for an optimal policy, initialized as an MPC controller.

To demonstrate this, we investigate how the states explored by the residual architecture and the end-to-end policy differ through training. We compare the states experienced from noisy, exploratory actions at the initial stage of training with the final stage of training for the two policies, alongside the distribution from the (fixed) MPC controller alone. The states from 24 simulation steps are collected, the same amount of data seen from each iteration of PPO. To visualize the 48 dimensional state-space of the humanoid platform, we use the Uniform Manifold Approximation and Projection (UMAP) technique [55] to reduce the dimension of the state data to a 2D manifold, as shown in Fig. 9. For the UMAP metric, we use the L-1 norm because of the differences in scale between state variables (e.g., joint angles vs. base linear velocities). However, the shape of the mappings appear to be similar regardless of the metric used.

We find that despite being given the same reward signal, the convergence of the states is strongly affected by the bias from the MPC prior. At the start of training, with a zero-initialized policy network, the state distribution of the residual policy is predictably close to that of the MPC controller. This influence throughout training alters the evolution of the policy - whereas the end-to-end policy converges to an entirely different distribution of states, the residual remains close to the initial MPC distribution despite being given the same reward signal. Some aspects of the UMAP visualization are similar, most likely corresponding to base velocities or initialization conditions, but overall, the differences in state distribution reflect the qualitative differences in locomotion behavior shown in Fig. 8. The MPC prior biases the visited states during training, and consequently, the converged locomotion behavior of the residual policy. It acts as a "warm-start" for the RL training, initializing state-action pairs close to the MPC controller.

C. Performance

We demonstrate how the residual architecture outperforms the baseline MPC controller in several metrics. First, we evaluate the policies with uniformly randomized velocity commands, and plot the 95% kernel density estimate of the commands with less than 0.25 m/s tracking error, which we consider "achieved". The resulting boundaries are shown in Fig. 10.

The residual policy substantially improves the achievable velocities from the MPC baseline, by roughly 78% in v_x, 12% in v_y, and 9% in ω_z. While the performance between the

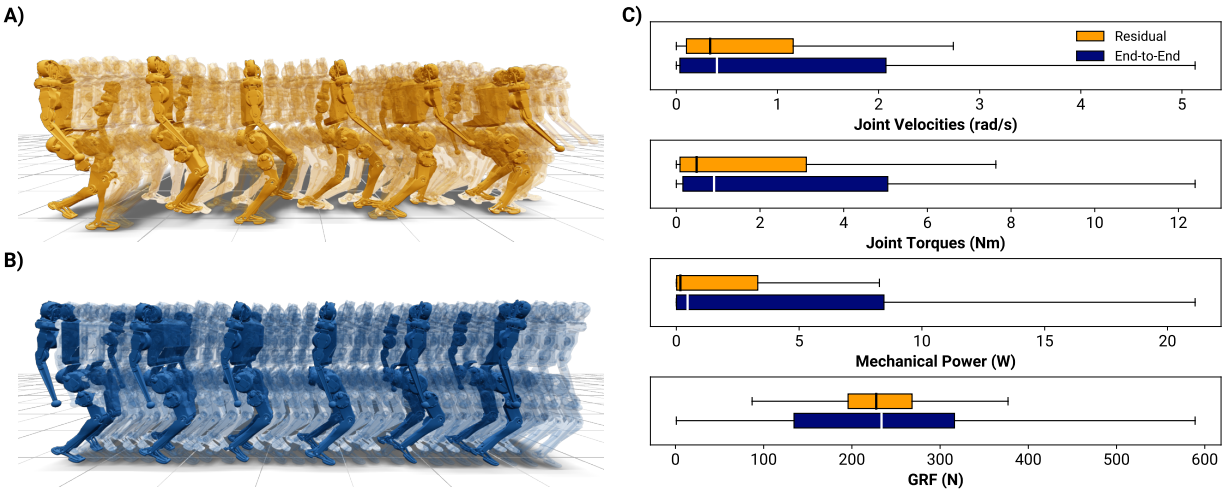

Fig. 8. **A)** Snapshots of the residual policy in motion walking forward at 1 m/s. **B)** Snapshots of the end-to-end policy trained with the same set of rewards. The policy exhibits unrealistic "gliding" motion, where the feet rapidly tap the ground to traverse the environment. **C)** Comparative distributions between the residual and end-to-end policies. Across all metrics, the residual policy exhibits lower medians and spreads compared to the end-to-end policy, suggesting more viable sim-to-real transfer.

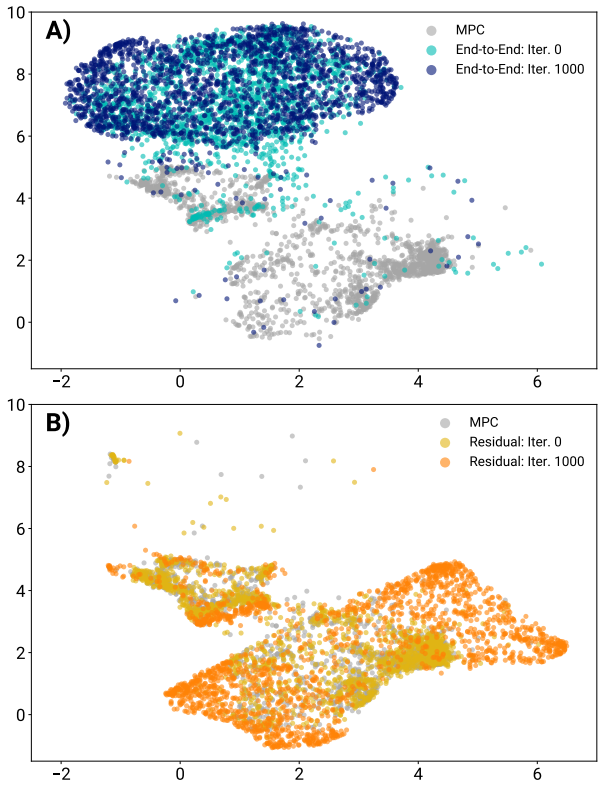

Fig. 9. Distribution of states experienced at the start of training and at training convergence, visualized with UMAP dimensionality reduction. The states incurred from the fixed MPC controller are shown in gray. **A)** State visualization for the end-to-end policy. **B)** State visualization for the residual policy.

residual and end-to-end policies are similar, as discussed in the previous section, it seems unlikely that the end-to-end policy would be viable for hardware deployment. The velocities achievable by the end-to-end policy are likely attained by exploiting numeric irregularities in the contact dynamics of the simulator.

Secondly, we explore how the residual architecture adapts the MPC controller for rewards omitted from the MPC cost. While most of the reward terms in Table I are included in the tracking cost of the MPC controller, a notably difficult constraint to enforce in a model-based setting is self-collision. Prior works have used simple collision primitives or geometric planning to avoid self-collision, but these methods tend to be heuristic or computationally expensive to solve online [5, 56]. Our MPC formulation contains no self-collision constraints, and we observed that from large commanded ω_z, the knees of the humanoid consistently collided while turning, as shown in Fig. 11. The plotted self-collision penalty shows that the MPC controller consistently experiences this self-collision while turning at a constant velocity.

With the residual architecture however, the MPC outputs are modified so that self-collision is avoided entirely. The residual policy learns to raise the base height of the humanoid torso and adjust its swing trajectory so that the knees never collide in the same manner. While this is a single illustrative example, the self-collision penalty from the MPC baseline steadily decreases throughout training, approaching zero at convergence. More broadly, the RL training paradigm is well-suited for costs that are not differentiable or easy to express, while MPC can capture a majority of desirable behavior from a principled, physics-based standpoint. The residual architecture allows the system to learn behaviors that are difficult to express as smooth constraints in an optimization, and combine the benefits of predictive, model-based control with experience-driven learning.

Next, we investigate the residual policy's adaptability to out-of-distribution scenarios. Specifically, we consider whether it is able to respond to changes in 1) the requested contact schedule from the MPC and 2) uneven terrain, neither of which was experienced during training.

During inference over flat terrain, we modified the phase parameters of the MPC controller so that a double stance and flight phase would be present in the contact schedule. Although the MPC prior alone was unable to walk stably with these

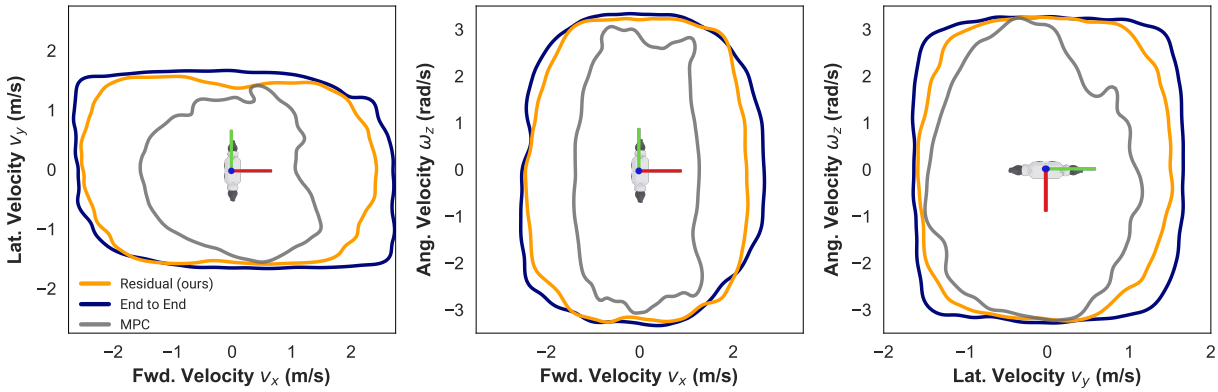

Fig. 10. 95% kernel density estimate boundaries of the trackable velocities for each policy. The residual policy network substantially improves the viable range of commands over the MPC baseline.

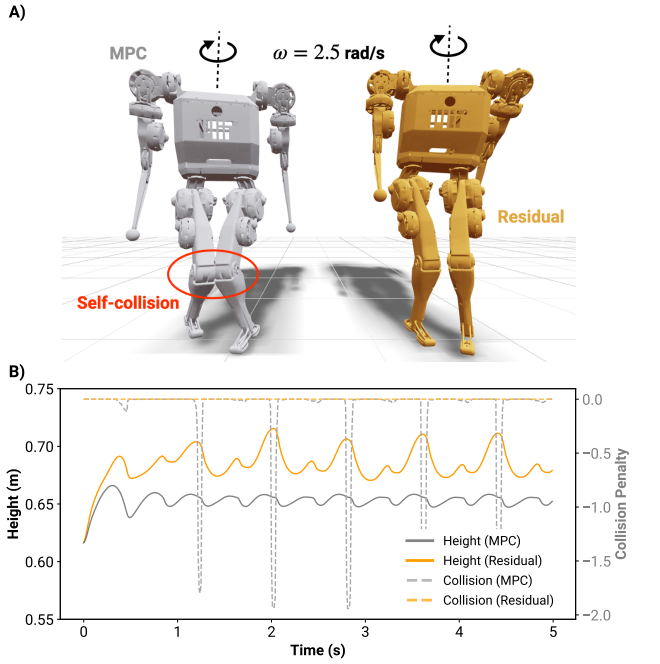

Fig. 11. **A)** The MPC policy collides at the knees when turning at 2.5 rad/s, while the residual does not. **B)** Plot of the controllers' height and self-collision penalty while turning.

changes, the residual policy was able to obey these unseen gait commands, as shown in Fig. 12 While prior work has relied on imitating distributions of reference data to adapt gaits, as in [51, 57], our residual architecture is able to immediately modify its contact pattern without being trained for it.

We also modify the terrain parameters in IsaacGym, and compare the performance of the baseline MPC policy against the residual architecture. As shown in Fig. 13, the residual policy is able to navigate unseen terrain with zero fine-tuning or domain randomization. Despite only being trained on flat, planar ground, the residual policy is capable of navigating uneven terrain with no adaptation, whereas the MPC controller alone immediately fails from incorrect assumptions about the ground height. For the discrete terrain with more pronounced steps, the feet of the system from the residual policy would occasionally be caught by the environment. To address this, we modify the swing height parameter in the MPC (see Fig. 3) to enforce higher stepping from 0.075 m to 0.15 m. Although this parameter was also not modified in training, the residual policy immediately adapts to the new swing behavior, and is able to successfully traverse terrain that requires higher clearance. To our surprise, the residual policy seems surprisingly robust to modifications in the parameters of the underlying MPC controller, allowing for tuning of behavior post-training.

Finally, we demonstrate deploying the proposed residual architecture on hardware on the MIT Humanoid [33], as shown in Fig. 14. The MPC is evaluated at 100 Hz onboard the robot with OSQP [44], and the residual network outputs are directly added to the MPC torques as described in subsection IV-B. Video results are available in the attached material. Out of concern for the integrity of the hardware, experiments were limited in scope to avoid catastrophic failures for the custom system (e.g., testing self-collisions at high angular velocities). Before deployment, we validated the residual policies in our custom simulator with more accurate contact dynamics. We observed significant discrepancies in the simulated rollouts, even for the MPC controller alone, likely explaining the sim-to-real gap for hardware deployment.

D. Residual Policy Analysis

In this section, we aim to analyze *when* and *how* the residual network is used during locomotion, especially in relation to the output of the MPC.

We begin by studying under what conditions the residual policy is more "active". To compare the residual and MPC torques, we use the ratio of their respective magnitudes

$$\frac{\|\boldsymbol{\tau}_{\text{residual}}\|}{\|\boldsymbol{\tau}_{\text{MPC}}\|}. \tag{26}$$

While we could also normalize the residual torque against the net joint torque, the MPC and residual torques can oppose each other, masking their true relative contribution.

From Fig. 15, we see that the contribution from the residual torques is asymmetric with respect to the commands. Interestingly, the residual torque ratio is positively correlated with the commanded forward velocity. We hypothesize that the MPC is actually more stable walking backwards than

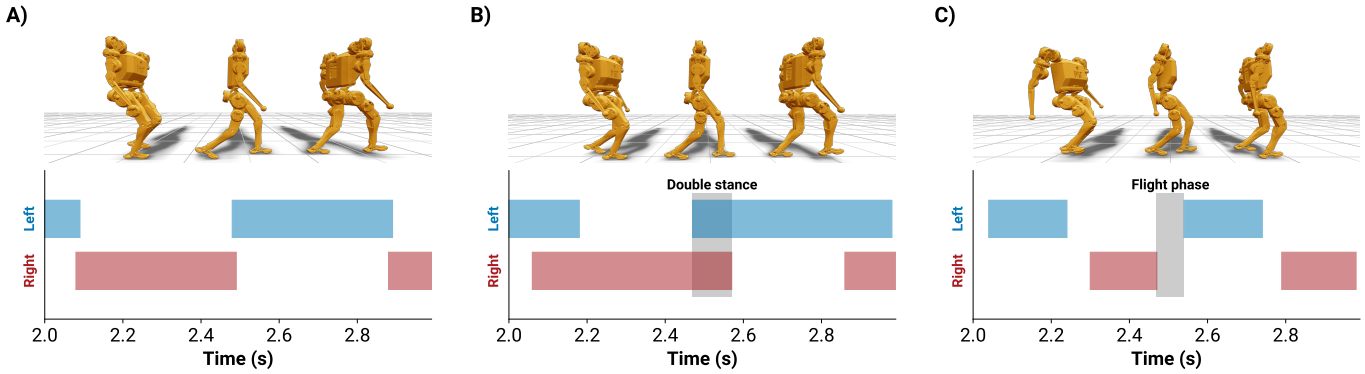

Fig. 12. The gait parameters of the MPC are varied after training the residual network to induce double stance and flight phase in the contact schedule. While the MPC is incapable of maintaining these gaits stably, the residual network can adapt to these out of distribution contact modes despite not experiencing them during training.

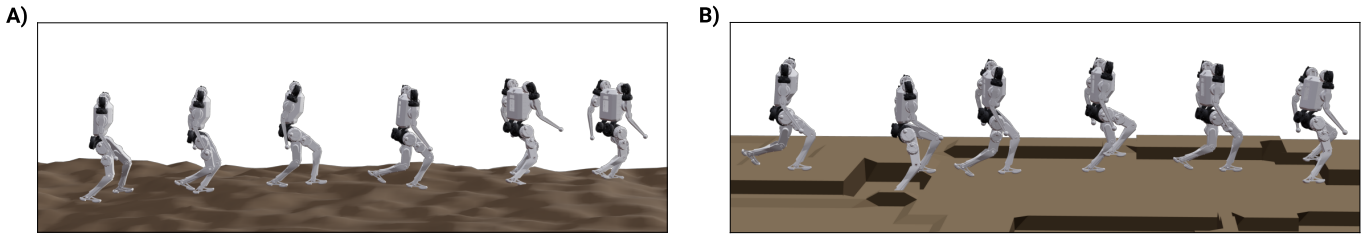

Fig. 13. The residual policy is capable of traversing terrain never seen during training. Despite only being trained over flat ground, the residual is capable of navigating various surfaces including **A)** uneven, rough terrain and **B)** discrete, sloped terrain.

Fig. 14. Hardware validation of the residual policy architecture on the MIT Humanoid [33], for **A)** forward walking, **B)** lateral walking, and **C)** turning.

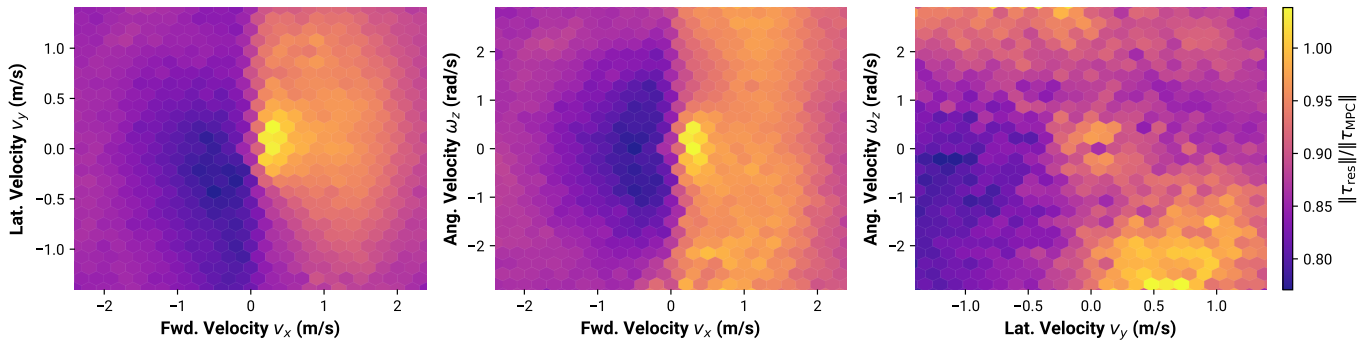

Fig. 15. The ratio of residual to MPC torque magnitudes over the commanded velocities. The residual policy is leveraged asymmetrically, especially for v_x commands.

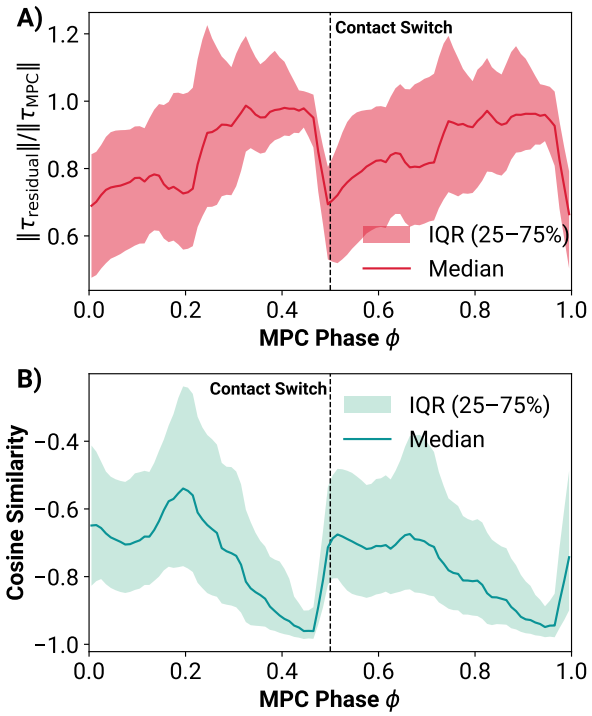

Fig. 16. **A)** The percentage of torque from the residual with respect to MPC as a function of phase. **B)** The cosine similarity between the residual and MPC torques.

forwards, as was the case with [5], and therefore relies less on the residual policy. When comparing the effect of the commanded lateral and angular velocity, the ratio is high in the second and fourth quadrants (negative v_y, positive ω_z and positive v_y, negative ω_z respectively), and low in the first and third quadrants (positive v_y, positive ω_z and negative v_y, negative ω_z respectively). We believe this is from the lack of self-collision constraints in the MPC formulation. With positive/negative lateral velocity and negative/positive angular velocity, the humanoid turns inwards as it moves laterally, risking collision with the forward-protruding knees of the body. The residual policy needs to actively raise the height of the body and modify the MPC swing trajectory to avoid this collision. With the other combinations, the humanoid is turning outwards, spreading the knees and stance legs apart, which has a much lower risk of self-collision.

Additionally, we plot the ratio of residual torques as a function of the contact phase in Fig. 16A. During training, we set the contact parameters such that at $\phi = 0.5$, the commanded contact switches from swing to stance for the right foot, and stance to swing for the left. Interestingly, we see an increase in the residual torque ratio when the phase is near the switch, right before the feet are expected to make contact with the ground. Given the robustness of the RL locomotion policies to uneven terrain and imprecise contact timing, we attempt to interpret and characterize the actions of the residual network more closely.

To do so, we compute the normalized dot product between the MPC torques and residual torques

$$\cos(\theta) = \frac{\boldsymbol{\tau}_{\text{residual}} \cdot \boldsymbol{\tau}_{\text{MPC}}}{\|\boldsymbol{\tau}_{\text{residual}}\| \, \|\boldsymbol{\tau}_{\text{MPC}}\|}, \tag{27}$$

across the phase, as shown in Fig. 16B. This cosine similarity characterizes the "antagonism" of the residual torque: a value of -1 implies that the residual network exactly opposes the MPC torques in direction, while a value of 1 implies that the residual network is purely additive, scaling the MPC torque by some factor. When we visualize this as a function of the phase, the residual torques almost *completely oppose* the MPC output near contact switches. Indeed, the residual torques seem to generally be misaligned with the MPC torques, implying the residual significantly modulates the controller throughout swing and stance.

Finally, we analyze the differences between the residual and baseline MPC policy in Fig. 17. Both controllers are initialized from the same conditions, and commanded to walk forward at 1.0 m/s across uneven terrain. As discussed previously, the residual policy is capable of successfully navigating uneven terrain, even when it was never encountered during training. For the MPC controller, however, the accumulated errors from uneven terrain destabilize its solution quality leading to failure shortly after initialization. For the respective policies, we plot the vertical tracking performance over this terrain, effective vertical ground reaction force (see Appendix A), sagittal joint torques, and the foot states. We notice that the residual policy exhibits significant heel-toe transitions during locomotion, despite not being given rewards to encourage this behavior. This is reflected in the increased ankle torques at the end of the stance phase and the early takeoff of the foot. While the vertical force profiles of the two controllers are similar, the residual policy commands a larger reaction force at touchdown, and generally uses more noticeable ankle strategies. The MPC controller is constrained to touchdown with both the heel and toe at the same time, but the residual policy learns to adapt this, evidenced by the early takeoff of the foot center and ankle torques near the end of stance. The touchdown velocity of the residual policy also returns to zero in two discernible stages, roughly when the phase is 0.48 and 0.52. This likely corresponds to the heel and toe touching down separately. In comparison, the average MPC touchdown velocity is a single sharp peak, and the adherence to a strict contact schedule makes it difficult to adapt to unexpected contacts.

VI. Conclusion

In this work, we present a unified control architecture that integrates model predictive control with reinforcement learning to achieve robust and adaptable behavior. We introduced an efficient GPU-parallelized MPC formulation that enables concurrent, in-the-loop policy training at high frequencies, making it feasible to train with optimal controllers in the RL loop on a single desktop computer. Our analysis shows how the MPC prior can shape and guide the learning process, and we closely study how the residual network can modify MPC outputs to improve robustness under environmental uncertainty. Finally, we validate the proposed locomotion controller on hardware for the MIT Humanoid [33], demonstrating that the combined approach translates from simulation to physical platforms.

While these results are promising, our investigation is only the first step into exploring how predictive controllers and

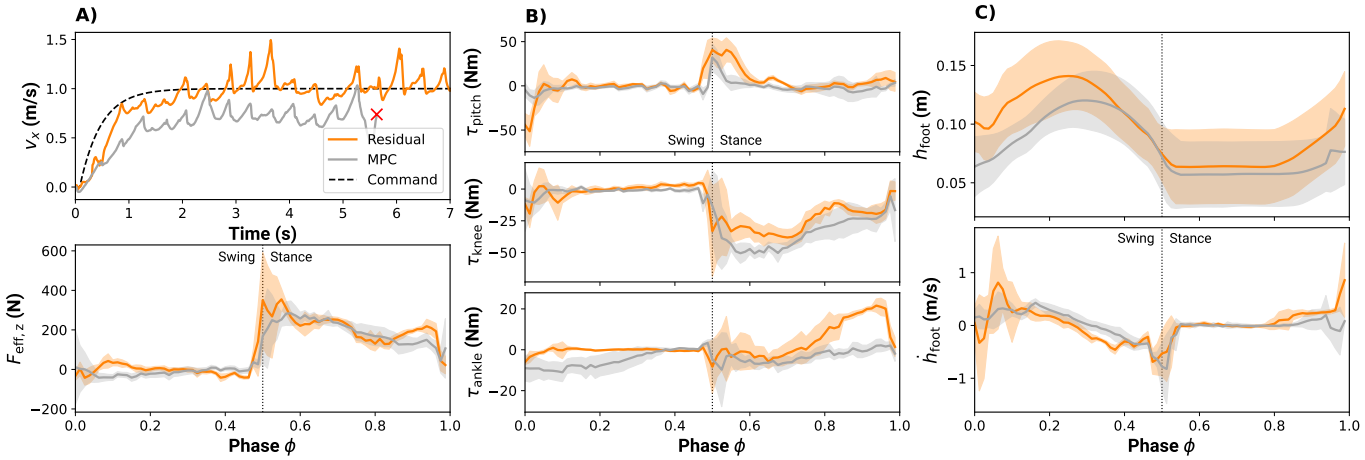

Fig. 17. Analysis of residual and MPC data from locomotion over uneven terrain at 1.0 m/s. **A)** Velocity tracking and effective vertical commanded force for the right foot. **B)** Sagittal joint torques through the phase. The residual policy exhibits emergent toe-off and heel touchdown behavior from training. **C)** Height and velocity profiles of the center of the foot link during swing and stance.

learned policies can be blended together within the RL training process. Our future work will build upon this control architecture in several ways. While we strictly limited our scope to only modify the outputs of the predictive controller, we plan to extend the network to also output MPC parameters, such as cost function weights or contact schedules directly. We also plan to study how the value returned by the MPC solution can be used to estimate uncertainty of the model-based controller, and appropriately balance between the network and MPC. Large perturbations or errors in model parameters cause spikes in the predicted value, which could serve as a salient signal for the condition of the MPC solution.

Another direction of interest is to design network architectures that mimic the operations required to solve the MPC problem - namely, a factorization step and iterations - to greatly reduce the computational time required. By carefully constructing a network with permutation invariance and recurrence akin to the ADMM iterations, we anticipate that we can approximate the MPC problem more effectively than with a standard MLP architecture. With the MPC formulation parallelized on the GPU, data for the individual linear solves, iterations, and full solution would be cheap to collect.

VII. Acknowledgments

The authors would like to thank David Nguyen, Kendrick Cancio, Annika Marschner, and AZ Krebs at the Biomimetic Robotics Lab for their insightful feedback on the saturation and value of Fig. 13.

References

[1] B. Yi et al., *Viser: Imperative, web-based 3d visualization in python*, 2025. arXiv: 2507.22885 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2507.22885

[2] S. Hong, J.-H. Kim, and H.-W. Park, "Real-time constrained nonlinear model predictive control on so (3) for dynamic legged locomotion," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020, pp. 3982–3989.

[3] Y. Ding, A. Pandala, C. Li, Y.-H. Shin, and H.-W. Park, "Representation-free model predictive control for dynamic motions in quadrupeds," *IEEE Transactions on Robotics*, vol. 37, no. 4, pp. 1154–1171, 2021.

[4] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt, and S. Kim, "Dynamic locomotion in the mit cheetah 3 through convex model-predictive control," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, IEEE, 2018, pp. 1–9.

[5] C. Khazoom, S. Hong, M. Chignoli, E. Stanger-Jones, and S. Kim, "Tailoring solution accuracy for fast whole-body model predictive control of legged robots," *IEEE Robotics and Automation Letters*, 2024.

[6] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *2014 IEEE-RAS International Conference on Humanoid Robots*, IEEE, 2014, pp. 295–302.

[7] G. García, R. Griffin, and J. Pratt, "Mpc-based locomotion control of bipedal robots with line-feet contact using centroidal dynamics," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, IEEE, 2021, pp. 276–282.

[8] J. Shen and D. Hong, "Convex model predictive control of single rigid body model on so (3) for versatile dynamic legged motions," in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 6586–6592.

[9] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, eabk2822, 2022. DOI: 10.1126/scirobotics.abk2822 eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abk2822. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abk2822

[10] X. Gu, Y.-J. Wang, and J. Chen, "Humanoid-gym: Reinforcement learning for humanoid robot with zero-shot

sim2real transfer," *arXiv preprint arXiv:2404.05695*, 2024.

[11] I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath, "Real-world humanoid locomotion with reinforcement learning," *Science Robotics*, vol. 9, no. 89, eadi9579, 2024.

[12] R. Batke et al., "Optimizing bipedal maneuvers of single rigid-body models for reinforcement learning," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, IEEE, 2022, pp. 714–721.

[13] Z. Xie, P. Clary, J. Dao, P. Morais, J. Hurst, and M. Panne, "Learning locomotion skills for cassie: Iterative design and sim-to-real," in *Conference on Robot Learning*, PMLR, 2020, pp. 317–329.

[14] Z. Xie, G. Berseth, P. Clary, J. Hurst, and M. Van de Panne, "Feedback control for cassie with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2018, pp. 1241–1246.

[15] J. Siekmann, K. Green, J. Warila, A. Fern, and J. Hurst, "Blind Bipedal Stair Traversal via Sim-to-Real Reinforcement Learning," in *Proceedings of Robotics: Science and Systems (RSS)*, 2021.

[16] J. Hwangbo et al., "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, eaau5872, 2019. DOI: 10.1126/scirobotics.aau5872 eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aau5872. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aau5872

[17] J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping.," in *ICML*, 1998.

[18] OpenAI, *Faulty reward functions in the wild*, https://openai.com/index/faulty-reward-functions/, Accessed: 2025-09-19.

[19] A. Romero, Y. Song, and D. Scaramuzza, "Actor-critic model predictive control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 14 777–14 784.

[20] F. Jenelten, J. He, F. Farshidian, and M. Hutter, "Dtc: Deep tracking control," *Science Robotics*, vol. 9, no. 86, eadh5401, 2024.

[21] M. Lutter et al., "Learning dynamics models for model predictive agents," *arXiv preprint arXiv:2109.14311*, 2021.

[22] R. Reiter et al., "Synthesis of model predictive control and reinforcement learning: Survey and classification," *arXiv preprint arXiv:2502.02133*, 2025.

[23] T. Silver, K. Allen, J. Tenenbaum, and L. Kaelbling, "Residual policy learning," *arXiv preprint arXiv:1812.06298*, 2018.

[24] J. Y. Luo, Y. Song, V. Klemm, F. Shi, D. Scaramuzza, and M. Hutter, "Residual policy learning for perceptive quadruped control using differentiable simulation," *arXiv preprint arXiv:2410.03076*, 2024.

[25] T. Johannink et al., "Residual reinforcement learning for robot control," in *2019 international conference on robotics and automation (ICRA)*, IEEE, 2019, pp. 6023–6029.

[26] E. Cramer, B. Frauenknecht, R. Sabirov, and S. Trimpe, "Contextualized hybrid ensemble q-learning: Learning fast with control priors," *arXiv preprint arXiv:2406.19768*, 2024.

[27] D. Youm, H. Jung, H. Kim, J. Hwangbo, H.-W. Park, and S. Ha, "Imitating and finetuning model predictive control for robust and symmetric quadrupedal locomotion," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7799–7806, 2023. DOI: 10.1109/LRA.2023.3320827

[28] J. Cheng, D. Kang, G. Fadini, G. Shi, and S. Coros, "Rambo: Rl-augmented model-based whole-body control for loco-manipulation," *IEEE Robotics and Automation Letters*, vol. 10, no. 9, pp. 9462–9469, 2025. DOI: 10.1109/LRA.2025.3594984

[29] S. H. Jeon, S. Hong, H. J. Lee, C. Khazoom, and S. Kim, "Cusadi: A gpu parallelization framework for symbolic expressions and optimal control," *IEEE Robotics and Automation Letters*, 2024.

[30] NVIDIA, *Nvidia cudss (preview): A high-performance cuda library for direct sparse solvers*, https://docs.nvidia.com/cuda/cudss/, 2023.

[31] V. Makoviychuk et al., "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[32] C. Khazoom, D. Gonzalez-Diaz, Y. Ding, and S. Kim, "Humanoid self-collision avoidance using whole-body control with control barrier functions," in *2022 IEEE-RAS 21st International Conference on Humanoid Robots (Humanoids)*, IEEE, 2022, pp. 558–565.

[33] A. SaLoutos, E. Stanger-Jones, Y. Ding, M. Chignoli, and S. Kim, "Design and development of the mit humanoid: A dynamic and robust research platform," in *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, IEEE, 2023, pp. 1–8.

[34] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[35] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[37] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational physics*, vol. 378, pp. 686–707, 2019.

[38] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis, "Deepxde: A deep learning library for solving differential equations," *SIAM review*, vol. 63, no. 1, pp. 208–228, 2021.

[39] D. E. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous robots*, vol. 35, pp. 161–176, 2013.

[40] P. M. Wensing and D. E. Orin, "Improved computation of the humanoid centroidal dynamics and application for whole-body control," *International Journal of Humanoid Robotics*, vol. 13, no. 01, p. 1 550 039, 2016.

[41] G. Bledt, P. M. Wensing, S. Ingersoll, and S. Kim, "Contact model fusion for event-based locomotion in unstructured terrains," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2018, pp. 4399–4406.

[42] A. G. Pandala, Y. Ding, and H.-W. Park, "Qpswift: A real-time sparse quadratic program solver for robotic applications," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3355–3362, 2019.

[43] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "Qpoases: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[44] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[45] R. Schwan, Y. Jiang, D. Kuhn, and C. N. Jones, "Piqp: A proximal interior-point quadratic programming solver," in *2023 62nd IEEE Conference on Decision and Control (CDC)*, IEEE, 2023, pp. 1088–1093.

[46] D. Ruiz, "A scaling algorithm to equilibrate both rows and columns norms in matrices," CM-P00040415, Tech. Rep., 2001.

[47] R. Featherstone and D. Orin, "Robot dynamics: Equations and algorithms," in *Proceedings 2000 ICRA. Millennium conference. IEEE international conference on robotics and automation. Symposia proceedings (Cat. No. 00CH37065)*, IEEE, vol. 1, 2000, pp. 826–834.

[48] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019. DOI: 10.1007/s12532-018-0139-4

[49] H. J. Lee, S. H. Jeon, and S. Kim, "Learning humanoid arm motion via centroidal momentum regularized multi-agent reinforcement learning," *IEEE Robotics and Automation Letters*, 2025.

[50] J. Ho and S. Ermon, "Generative adversarial imitation learning," *Advances in neural information processing systems*, vol. 29, 2016.

[51] A. Tang et al., "Humanmimic: Learning natural locomotion and transitions for humanoid robot via wasserstein adversarial imitation," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 13 107–13 114.

[52] S. H. Jeon, S. Heim, C. Khazoom, and S. Kim, "Benchmarking potential based rewards for learning humanoid locomotion," *arXiv preprint arXiv:2307.10142*, 2023.

[53] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[54] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[55] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018. DOI: 10 . 21105 / joss . 00861 [Online]. Available: https://doi.org/10.21105/joss.00861

[56] N. D. Ratliff, J. Issac, D. Kappler, S. Birchfield, and D. Fox, "Riemannian motion policies," *arXiv preprint arXiv:1801.02854*, 2018.

[57] A. Reske, J. Carius, Y. Ma, F. Farshidian, and M. Hutter, "Imitation learning from mpc for quadrupedal multi-gait control," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 5014–5020.

APPENDIX
JOINT-TO-CARTESIAN ACTUATION MAPPING

Having compared and analyzed both residual and MPC outputs in joint space in Section V-D, it is natural to examine how these actions contribute in a more intuitive 3D task space, namely the contact frame located at the foot. Therefore, to better understand how the residual and MPC outputs influence locomotion in task-space coordinates, we transform the output torques from joint space into task space. The details of this transformation are provided in this section.

To analyze the six-dimensional ground reaction wrench (GRW) exerted on each foot, we apply a quasi-static mapping that converts joint torques into task-space GRWs. Specifically, we use the quasi-static equations of motion (i.e., neglecting inertial and Coriolis forces), a common approach used to control task-space ground reaction force [2–4]. This method enables the computation of equivalent joint torques that produce the same effect as if external forces were applied at specified contact locations.

$$\boldsymbol{\tau} = (\mathbf{J})^{\intercal} \cdot \mathbf{W}, \tag{28}$$

where $\boldsymbol{\tau} \in \mathbb{R}^6$ denotes the joint torques for each leg, $\mathbf{J} \in \mathbb{R}^{6\times6}$ is the Jacobian matrix mapping joint-space velocities to task-space velocities, and $\mathbf{W} = [f_x, f_y, f_z, m_x, m_y, m_z] \in \mathbb{R}^6$ represents the task-space GRW acting at the foot. Given the $\boldsymbol{\tau}$, if the Jacobian matrix is full rank, it can be inverted to uniquely determine $\mathbf{F}$ from $\boldsymbol{\tau}$. For our system, the MIT Humanoid [33], each leg has five actuators without having an ankle-roll actuator (i.e., $\boldsymbol{\tau}_a = 0$), we constrain the direction in which GRWs can be applied:

$$0 = \boldsymbol{\tau}_a = \mathbf{a} \cdot \mathbf{W}, \tag{29}$$

where $\mathbf{a}$ is the corresponding row of $(\mathbf{J})^{\intercal}$. As a result, during locomotion the sole of the foot typically makes line contact with the ground. Therefore, we specify three translational ground reaction forces (f_x, f_y, f_z) and two ground reaction

moments (e.g., m_y, m_z) that are orthogonal to the contact line direction, and solve for the ground reaction moment m_x:

$$m_x = g(f_x, f_y, f_z, m_y, m_z) \tag{30}$$

Substituting Eq. 30 into Eq. 29, we obtain

$$\boldsymbol{\tau}' = (\mathbf{J}')^{\intercal} \cdot \mathbf{W}', \tag{31}$$

where $\boldsymbol{\tau}' \in \mathbb{R}^5$ denotes the joint torques excluding ankle-roll torque, $\mathbf{W}' = [f_x, f_y, f_z, m_y, m_z] \in \mathbb{R}^5$ represents the GRW excluding the line-contact direction, and $\mathbf{J}' =\in \mathbb{R}^{5\times5}$ is the corresponding Jacobian matrix. We now have a square, full-rank matrix $\mathbf{J}'$ that can be inverted to uniquely determine $\mathbf{W}'$ given the joint torques $\boldsymbol{\tau}'$. Substituting $\mathbf{W}'$ into Eq. 30, we can then compute the ankle-roll torque and recover the full six-dimensional wrench $\mathbf{W}$.