Cascaded Compositional Residual Learning for Complex Interactive Behaviors

K. Niranjan Kumar, Irfan Essa and Sehoon Ha

***Abstract*— Real-world autonomous missions often require rich interaction with nearby objects, such as doors or switches, along with effective navigation. However, such complex behaviors are difficult to learn because they involve both high-level planning and low-level motor control. We present a novel framework, Cascaded Compositional Residual Learning (CCRL), which learns composite skills by recursively leveraging a library of previously learned control policies. Our framework combines multiple levels of pre-learned skills by using multiplicative skill composition and residual action learning. We also introduce a goal synthesis network and an observation selector to support combination of heterogeneous skills, each with its unique goals and observation spaces. Finally, we develop *residual regularization* for learning policies that solve a new task, while preserving the style of the motion enforced by the skill library. We show that our framework learns joint-level control policies for a diverse set of motor skills ranging from basic locomotion to complex interactive navigation, including navigating around obstacles, pushing objects, crawling under a table, pushing a door open with its leg, and holding it open while walking through it. The proposed CCRL framework leads to policies with consistent styles and lower joint torques, which we successfully transfer to a real Unitree A1 robot without any additional fine-tuning.**

I. INTRODUCTION

Real-world autonomous missions often involve various levels of motor skills ranging from simple locomotion and manipulation to interactive navigation. For instance, a quadrupedal tasked to fetch an object may need to walk to a nearby door, open it with its end-effector, and navigate to the destination (see Fig. 1). Traditionally, many researchers [1], [2], [3] have approached modeling such high-level behaviors by manually decomposing them into several low-level motor skills, such as locomotion, navigation, manipulation, and a high-level decision layer to modulate these skills. While effective, this model-based approach requires researchers to derive the explicit model of environmental interactions that are often complicated and cumbersome in such scenarios.

Learning-based approaches, such as deep reinforcement learning (deep RL) [4], [5], [6], hold promise for automatically generating effective motor policies from simple descriptions. However, learning complex behaviors is not straightforward due to many theoretical and practical challenges. For instance, motor creatures with many degrees of freedom (DoF), such as quadrupedal or bipedal robots, require a massive amount of simulation samples even for the simplest motor tasks due to their high-dimensional state and action spaces [7], [8], [9]. In addition, learning a high-level skill requires careful reward engineering, which is extremely time-consuming to tune. Researchers have investigated an explicit two-level hierarchical architecture to alleviate this problem [10], [11], but it only arranges a sequence from existing behaviors. For instance, it cannot combine locomotion and target-reaching into a door-opening skill.

Fig. 1: Our framework allows robots to learn complex environment interactions with a recursive hierarchy. The image shows a motion frame of the robot crawling under a table, reaching the door, pushing it open, and walking through it to reach a target location.

We present Cascaded Compositional Residual Learning (CCRL), a novel framework to learn a family of motor skills on a 12 DoF quadruped robot and interactively navigate around an indoor scene. Our architecture builds new skills by recursively leveraging a pre-learned library consisting of primitive (e.g., walking straight) or other high-level motor skills (e.g., door opening). We accomplish this nested skill composition by learning *multiple* levels of weighting networks (via multiplicative composition [12]) and residual policy networks [13], [14]. Additionally, we introduce *residual regularization* to strike a balance between maximizing a given reward function and adhering to a combination of foundational skills to ensure feasible motion on a real robot.

The main contributions of our work are as follows:

1) A cascaded compositional residual learning (CCRL) framework to recursively train skills with increasing complexity by reusing skills learned in previous steps.
2) A novel combination of multiplicative skill composition and residual learning which serves as the fundamental unit of our multi-level cascaded policy architecture.
3) Introduction of a goal synthesis network and an observation selector which allow seamless combinations of heterogeneous skills.
4) A constrained residual learning objective that anchors

This work was partly supported by the Cisco Research Corporation

K. Niranjan Kumar is with the Georgia Institute of Technology, Atlanta, GA 30332

Sehoon Ha is with the Georgia Institute of Technology, Atlanta, GA 30332 USA, and with Robotics, Google, Mountain View, CA 94043 USA

Irfan Essa is with the Georgia Institute of Technology, Atlanta, GA 30332 USA, and with Google, Mountain View, CA 94043 USA

newly acquired skills to achieve realistic and achievable motions.

5) Demonstration of interaction skills learned in simulation and successful transfer to a real Unitree A1 robot.

II. RELATED WORK

A. Robotic Locomotion

Controlling legged robots has been a long-standing topic of research in the robotics community. Traditionally, roboticists [15], [16], [17], [18], [19], [20] have approached this problem with a combination of different frameworks and algorithms like trajectory optimization, model-predictive and whole-body control to demonstrate robust and agile locomotion. However, such manual controller design techniques often require in-depth prior knowledge about the robot dynamics, which restricts its applicability to new problems. On the other hand, reinforcement learning [5] offers an automated controller design process by optimizing a policy for a reward function that measures performance on a given task. A body of work [21], [9], [8], [7], [22], [23], [24] in RL has demonstrated effective learning of locomotion policies. However, these learned policies often show degraded performance on hardware due to the difference between the simulation and the real world, referred to as the *sim-to-real* gap. Several techniques have been proposed to bridge the sim-to-real gap, such as domain randomization [7], [8], learning actuator dynamics [23], online adaptation [25], [9], real-world learning [26], [27]. Our work also leverages existing domain randomization techniques to deploy learned interactive behaviors to the real world.

B. Interactive Navigation

Navigation is a fundamental skill for autonomous robot missions. Several navigation problems have been proposed over the years [28], [11], such as: PointNav - navigating to a point in a map; ObjectNav - navigating to a selected object category in the scene; InteractiveNav - Navigating to a point in a scene that requires interaction with objects and furniture in the scene. While researchers have made significant progress in PointNav [29] and ObjectNav [30], [31], [32], InteractiveNav [11], [33], [34], [35] is still a challenging problem due to the difficulty in learning interaction dynamics between a robot and its environment. Li [11] learned a navigation policy on a mobile robot that can open a door to reach its target location. Konidaris [35] took an alternate approach where the robot was given access to a set of hard-coded controllers and had to learn when to use each skill. However, these approaches focus on just wheeled mobile robots, abstracted to be a simple cylinder with an attached manipulator, significantly limiting the space of possible interactions. Sunwoo [6] demonstrated interaction skills on a quadrupedal robot via manual motion-based control. In contrast, our work focuses on learning autonomous policies to directly control joint motors of a 12-DOF quadruped robot, to perform complex dynamic interactions such as door opening, object pushing and crawling under a table. To the best of our knowledge, our work is the first of it's kind, demonstrating end-to-end neural network policies, that can solve InteractiveNav on a high DOF legged robot, with a high success rate.

Another related problem is interactive search, where a robot has to *search* for an object in an environment. Most of the work in this area [36], [37], [38] focuses on searching for an object in a cluttered shelf or table. In a more generalized version of interactive search, a robot moves around a cluttered indoor scene searching for an object. While we do not tackle interactive search, our method complements existing techniques by enabling a robot *interactively* navigate to the object, once its location has been estimated.

C. Hierarchical Reinforcement Learning

Learning RL policies for complex tasks requires extensive reward engineering and hyperparameter tuning. A common technique to deal with such complexity is to decompose the policy into multiple sub-tasks. In hierarchical RL (HRL) [39], [40], higher level policies control and instruct low level policies like they were primitive actions. While an in-depth discussion about hierarchical RL is outside the scope of this paper, we point interested readers to refer [41], and restrict our discussion to related work in locomotion and navigation. A common approach to navigation is to use low-level locomotion primitives in combination with a high-level primitive selection or goal proposal network to perform navigation [42], [43], [44], [45]. Alternatively, Yang [10] proposed a gating network that combines the parameters of a collection of neural network primitives, to produce composite navigation policy. Similar to these works, we are interested in designing a hierarchy to solve a complex task, but our low-level policies are themselves complex (like opening a door) and not easily breakable into simpler policies. Therefore, a common "flat" hierarchy will not be sufficient to learn more challenging motor skills. We propose a multi-level cascaded hierarchy to build progressively complex policies starting out from a small set of learned skills. We build this hierarchy on the fly, by learning residuals [13], [14] that can perturb the output of an existing library of policies, to generate novel behaviors.

III. CASCADED COMPOSITIONAL RESIDUAL LEARNING

This section describes our cascaded compositional residual learning (CCRL) framework that learns complex motor behaviors, by recursively obtaining control policies and reusing them to solve increasingly difficult tasks. We will begin the section by explaining relevant background, followed by problem formulation, skill reusing mechanisms and learning algorithms.

A. Background: Markov Decision Process

Robot learning can be modeled as a Markov Decision Process (MDP) defined by the 5-tuple $\mathcal{M} : \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, r, \gamma \rangle$, where $\mathcal{S}$, $\mathcal{A}$ and r are the state space, the action space, and the reward function, respectively. $\mathcal{T}$ is a transition function that determines the next state given the current state and action. $\gamma \in (0, 1)$ is the discount factor. Our goal then, is

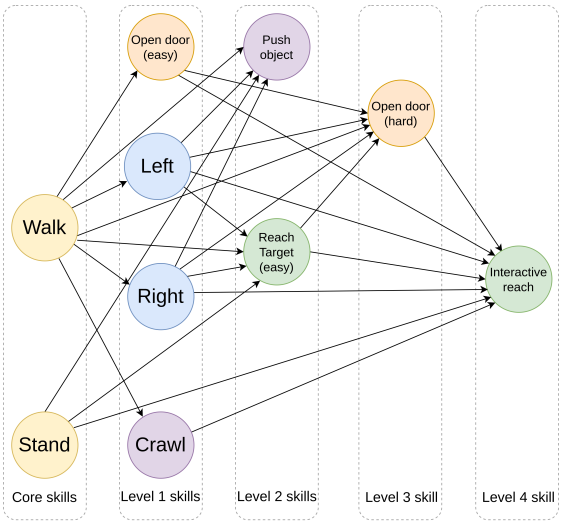


Fig. 2: Directed graph representing the relationship between skills. Each skill is built as a compositional policy over policies from earlier levels and a learned residual.

to find a policy $\pi : \mathcal{S} \rightarrow \mathcal{A}$ that maximizes a cumulative return $J(\pi) = \mathbb{E}_{\tau \sim \rho_\pi}\left[\sum_{t=0}^{T} r(s_t, a_t)\right]$, where ρ_π is the distribution over trajectories (τ) imposed by policy π and $\mathbb{E}$ is the expectation.

However, solving the given MDPs for high-level motor tasks, such as interactive navigation or full-body manipulation, is not straightforward. First, the given behavior involves multiple sub-tasks, such as walking or door opening, which require the manual design of very specific reward functions. In addition, the long-horizon nature of the problem makes it extremely sensitive to the choice of hyper-parameters. Further, we want to maintain smooth and stable motion styles to make them consistently feasible on actual hardware. Therefore, it is almost impossible to obtain such good motion controllers for these challenging motor tasks via simple reward engineering and hyperparameter tuning.

B. Background: Residual Learning

Residual learning [14], [13] offers a promising alternative to the traditional training-from-scratch RL framework by enabling the transfer of a skill learned on one task to another related but more difficult task. Consider a policy π_0 to be a solution for the base task that takes as input, state s and outputs action a. We can freeze the parameters of π_0 and learn a residual policy π_k to perturb the actions of π_0, to modify learned behaviors and accomplish a different goal. The final policy is obtained by adding the outputs:

$$\pi(s) = \pi_0(s) + \pi_k(s) \tag{1}$$

C. Problem Definition: Multi-skill Learning

In our formulation, we consider a set of n motor tasks described as MDPs $\Omega = \{\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$, where each $\mathcal{M}_k : \langle \mathcal{S}_k, \mathcal{A}, \mathcal{T}, r_k, \gamma \rangle \quad \forall \quad 0 < k < n$, models the MDP for a learnable skill. $\mathcal{S}_k, \mathcal{A}, \mathcal{T}, r_k$ and γ represent the state space, action space, transition function, reward function and discount factor respectively. Note that we allow unique state spaces for the problems, which indicates that some high-level behaviors require additional inputs for describing the task (e.g., the target location) or environmental states (e.g., the door hinge angle). On the other hand, we assume a single robot, which leads to unified actions and transition functions.

We then want to learn a collection of control policies $\mathcal{C} = \{\pi_1, \pi_2, \pi_3, \ldots, \pi_n\}$. A naïve approach is to independently solve each corresponding MDP. But in practice, it is often not feasible due to the challenges outlined in Sec. III-A. Our key insight is to "recursively" learn policies for challenging MDPs on top of the related "prerequisite" or "parent" skills instead of learning from scratch. To this end, we assume a Skill Decomposition Graph $\mathcal{G}$ by defining a set of prerequisite skills, where its vertices are the corresponding MDPs, and its edges represent the dependencies. For instance, Fig. 2, shows the dependencies of *Reach Target (Easy)* on *Walk*, *Stand*, *Turn Left* and *Turn Right*.

We assume the skill decomposition graph $\mathcal{G}$ to be intuitive to define (e.g., navigation requires straight walking and turning) but could also be automatically estimated by searching the space of all possible relations and retaining those that yield the best performance. Further, our framework is robust to the redundancy in prerequisite skills in the sense that it learns with subset of them or ignore irrelevant prerequisite skills (e.g., if the task does not require the robot to crawl, the crawling skill will be ignored). Please refer to Sec. IV-D.

D. Cascaded Compositional Residual Learning

A traditional flat architecture of HRL would train each of these policies from scratch, to solve the corresponding MDP. But due to challenges outlined in the previous section, it is not practically feasible. The key insight of our approach is that, instead of learning new skills from scratch, we build them on top of pre-learned skills, i.e. π_k could be a composite policy over the prerequisite policies $\mathcal{C}_k$, where $\mathcal{C}_k \subset \mathcal{C}$. Mathematically,

$$\pi_k = \mathcal{F}_k(\mathcal{C}_k \cup \{\pi_k^r\}) \tag{2}$$

where $\mathcal{F}_k$ is a merging function that combines the outputs of all $\pi_j \quad \forall \quad \pi_j \in \mathcal{C}_k$ and π_k^r is a residual neural network policy. To this end, our framework (Fig. 3) has three trainable networks, (1) a residual action network, (2) a weight network, and (3) a synthetic goal network, while freezing the prerequisite policies in $\mathcal{C}_k$.

Residual Network. The residual policy π_k^r generates a *residual action* that perturbs the actions generated by the learned policies to generate novel behavior that is absent in the set of pre-learned skills. We simply treat this residual network as an additional policy: the only difference being that the parameters of π_k^r are learnable during policy training.

Weight Network. While prior work in residual learning combines two policies by just adding them together, this approach has been observed to be sub-optimal by Peng et al. [12] when combining a large set of skills. They proposed Multiplicative Compositional Policy (MCP) that combines a

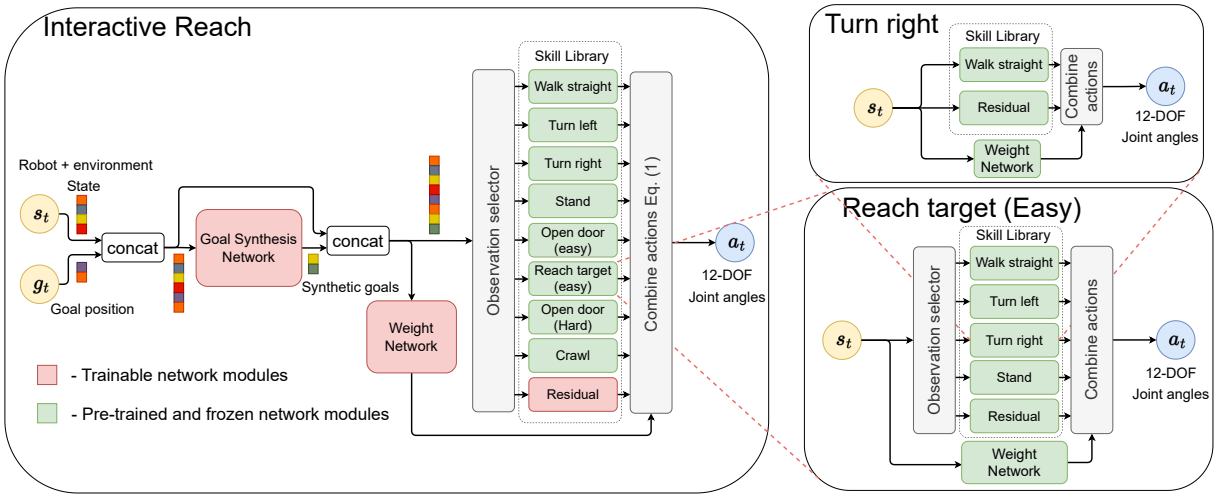


Fig. 3: Overview of our Cascaded Compositional Residual Learning, which consists of three learnable components: a synthetic goal network, a weight network, and a residual action network. The example illustrates the architecture for *Interactive Reach*, which uses *Reach target (Easy)* on top of other primitive skills.

set of stochastic policies while ensuring that multiple policies simultaneously work together and provide more flexibility in the final composite policy. MCP combines multiple skills by using the following equation:

$$\pi(a \mid s, g) = \frac{1}{Z(s,g)} \prod_{i=0}^{k} \pi_i(a \mid s, g)^{w_i(s,g)}, \tag{3}$$

where $w_i(s,g)$ are learned weighting functions and $Z(s,g)$ is a normalizing factor to ensure that the weights are between 0 and 1. We use MCP as the merging function $\mathcal{F}_k$.

Goal Synthesis Network. It is not intuitive to combine cascaded motor skills because some skills take heterogeneous goals as additional inputs. For example, the *Reach Target (Easy)* skill needs a goal location around the robot as input. But to use this skill as part of *Door open (Hard)*, we need to generate a collection of intermediate targets as input the policy. These targets are generated by the Synthetic Goals Network. We normalize the network outputs, scale them back to feasible goal ranges and concatenate them to the observations.

Observation Selector. Since each skill has its own observation space, we introduce an *Observation selector* which takes the input observations from the environment and selects the subset of observations relevant to each skill. The architecture is illustrated in Fig. 3. Our work provides a framework that builds novel complex skills, by leveraging a library of pre-learned composite skills, while ensuring control over the style of the composite. We use our framework to learn a wide range of interactive skills on a quadruped robot while implicitly guiding the *style* of the policy by controlling the weight given to the residual actions. With this framework we train a handful of basic policies from scratch, using RL, and use these policies to progressively build increasingly complex composite skills.

E. Auxiliary Loss for Constrained Residual Actions

In the previous section, we discussed learning novel skills by effectively leveraging a library of existing skills and a trainable residual. However, an unconstrained residual could dominate the output of the policy and give rise to unstable behaviors, which eventually lead to sub-optimal behaviors and unsuccessful sim-to-real transfer. Therefore, we introduce constraints into our policy optimization framework to control the extent to which residuals influence the policy. In our work, we use PPO (Proximal Policy Optimization) [46] as the RL algorithm of choice. Consider the standard PPO objective below:

$$L_t{}^{\text{PPO}}(\theta) = \hat{\mathbb{E}}_t \left[L_t{}^{\text{CLIP}}(\theta) - c_1 L_t{}^{\text{VF}}(\theta) + c_2 S\left[\pi_\theta\right](s_t) \right] \tag{4}$$

where $L_t{}^{\text{CLIP}}(\theta)$ is the clipped policy gradient objective, $L_t{}^{\text{VF}}(\theta)$ is the value function error and $S\left[\pi_\theta\right](s_t)$ is the entropy term. c_1, c_2 are weights to control the influence of each term over the total loss. We set coefficients c_1, c_2 to be 1.0 and 0.01 respectively. To constrain the effect of the residual, we introduce regularization terms to penalize the magnitude and weight given to the residual.

$$L_t{}^{\text{rm}} = \mid \mathbb{E}_t \left[\pi_{\text{res}}(a_t \mid s_t, g_t) \right] \mid_1 \tag{5}$$

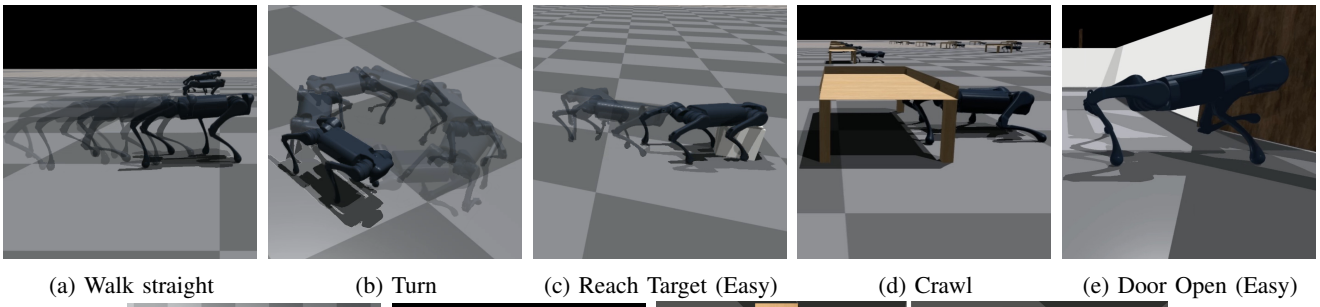
$$L_t{}^{\text{rw}} = \mathbb{E}_t \left[\mid w_{\text{res}}(s_t, g_t) \mid_1 \right] \tag{6}$$

L^{rm} and L^{rw} are penalties on the magnitude of the residual actions and the weights assigned to them respectively.

We modify PPO objective (eq. 4) to include regularization terms defined above:

$$L_t(\theta) = L_t{}^{\text{PPO}}(\theta) + c_3 L_t{}^{\text{rm}} + c_4 L_t{}^{\text{rw}}. \tag{7}$$

c_3, and c_4 are task dependent coefficients that are tuned to maximize the policy performance. Once we train a skill, we save the residual, weight, and goal synthesis networks and reuse them to train other skills. The critic is trained from scratch for every new skill.



(a) Walk straight (b) Turn (c) Reach Target (Easy) (d) Crawl (e) Door Open (Easy)

(f) Door Open (Hard) (g) Push Object (h) Interactive Reach

Fig. 4: Overview of the tasks we train our robot to accomplish.

IV. EXPERIMENTS

In this section, we design experiments to test the effectiveness of cascaded compositional residual learning (CCRL), by learning skills involving different levels of difficulty and comparing the performance with traditional RL baselines. Next, we present simulation analysis to investigate the sensitivity of the performance with respect to the skill choices and the importance of the auxiliary loss. Finally, we examine the learned skill policies on a real Unitree A1 robot and demonstrate robust sim-to-real transfer.

A. Problem Formulation

In our work, we train a collection of policies, that help an agent interactively navigate an indoor scene. Each skill has its own underlying MDP, but we provide a rough definition of the ingredients of the MDPs as follows.

States. A 60-dimensional state space consists of the base linear velocity (3), base angular velocity (3), base yaw angle (1), base position (2), gravity vector (3) projected on the base, joint angles (12), joint velocities (12), previous action (12), locations (x,y) of objects and the target relative to the robot (8), angle of the door joint(1), room index of the robot and target (2) and crawling height (1).

Actions. The action space is the desired target joint angles of the robot (12 for A1), which are fed into Proportional Derivative (PD) controllers.

Rewards. We use a collection of terms to account for different aspects of the robot motion. Let us denote the velocity as $\mathbf{v}$, the angular velocity as $\boldsymbol{\omega}$, the joint angles as $\mathbf{q}$, joint velocities as $\dot{\mathbf{q}}$, joint torques as $\boldsymbol{\tau}$, number of robot parts (excluding feet) in contact with the environment as $n_{contact}$, action taken at a given step as a_t, the angle made by the hinge of the door joint as q_{door}, the position of the target relative to the robot as $\mathbf{x}_{target}$, the position of the target relative to the object being pushed as $\mathbf{x}_{t2o}$ and Δt as the simulation time-step. The reward at time t is defined as the weighted sum of the following quantities:

(R1) Linear velocity tracking: $exp(-(\mathbf{v}_{\text{target}} - \mathbf{v})^2/\sigma_1)$
(R2) Angular velocity tracking: $exp(-(\omega_{z_{\text{target}}} - \omega_z)^2/\sigma_2)$
(R3) Pitch and roll penalty: $\omega_x^2 + \omega_y^2$
(R4) Joint acceleration penalty: $\sum(\frac{\dot{\mathbf{q}}_t - \dot{\mathbf{q}}_{t-1}}{\Delta t})^2$
(R5) Collision penalty: n_{contact}
(R6) Action change penalty : $\sum(\mathbf{a}_t - \mathbf{a}_{t-1})^2$
(R7) Torque penalty : $\sum \boldsymbol{\tau}^2$
(R8) Door angle: q_{door}
(R9) Distance to target: $exp(-\left\|\mathbf{x}_{\text{target}}\right\|/\sigma_3)$
(R10) Object-target distance: $exp(-\left\|\mathbf{x}_{t2o}\right\|/\sigma_3)$

σ_1, σ_2, σ_3 are scaling factors that we set to 0.25, 0.25 and 2.0 respectively.

Tasks. We define the following tasks, each aimed at developing a skill that is useful while navigating or exploring an indoor cluttered scene (Fig. 4). Their relationship is defined by the skill graph $\mathcal{G}$ (Fig. 2).

1) **Walk straight**: Walk forward in a straight line.
2) **Stand**: Stand in place when subjected to external disturbances.
3) **Turn left**: Turn to the left. The episode is successful if the robot circle backs to its initial yaw.
4) **Turn right**: Turn to the right. The episode is successful if the robot circle backs to its initial yaw.
5) **Reach target (Easy)**: Reach a target placed anywhere within a circle of radius $3m$ from the center of the robot. The episode is successful if the robot's center is within $0.5m$ from the goal (R9).
6) **Open door (Easy)**: Push door open with foot and hold it open while walking through it. The episode is successful if the robot crosses the door (R8).
7) **Open door (Hard)**: Reach the door from an arbitrary starting point in a room, push it open with a foot and hold it open while walking through it. The episode is

successful if the robot crosses the door (R8).

8) **Push object**: Push a cylindrical puck to a target location. The episode is successful if the puck is within $0.1m$ from the desired target location (R10).
9) **Crawl**: Crawl under a slab that gradually decreases in height, while avoiding collisions. The episode is successful if the robot crawls out from under the slab.
10) **Interactive Reach**: Reach a target object placed anywhere in a two-room house with the fixed layout of furniture and a door, while avoiding collisions. Robot is successful if its body center within $0.5m$ from the center of the target (R9).

All the tasks are trained with rewards R1-R7 in addition to the ones mentioned above.

B. Simulation Setup

We use Isaac Gym [47] to simulate our interactive environment and train our policies. We run 4096 environments in parallel, on a single NVIDIA Titan X GPU. During training, we randomize surface friction between the robot and the ground by randomly sampling from the range-$[0.5, 1.25]$. We implement all our policies as fully connected neural network layers with exponential linear units (*Elu*). See Table I for details. We train policies with our updated PPO objective (Equation 7).

Skill	Residual Architecture	Weight Network Architecture
Walk straight	[512, 256, 128]	-
Stand	[256, 128]	-
Turn left	[256, 128]	[128]
Turn right	[256, 128]	[128]
Reach target (Easy)	[256, 128]	[512, 256, 128]
Open door (Easy)	[512, 256]	[512, 256, 128]
Open door (Hard)	[512, 256]	[256, 128]
Push object	[256, 128]	[512, 256, 128]
Crawl	[256, 128]	[512, 256, 2]
Interactive Reach	[256, 128]	[512, 256, 2]

TABLE I: Network Architectures

C. Simulation Results

Our framework enables us to train complex long horizon interactive behaviors grounded in previously learned parent skills. Our policy was able to learn all the tasks described in Sec. IV-A and Fig. 4. The success rates (over 100 trials) of the learned policies are near 90%, except for the hardest task of *Interactive Reach* that shows a 72% success rate.

Performance Comparison. To highlight the importance of our cascaded residual framework, we compare our method against four baselines:

1) **Vanilla:** The policy is trained from scratch to solve the given problem, using the same architecture as the *Walk straight* skill.
2) **Curriculum:** A single policy is trained using a manually curriculum that gradually increases the complexity of the environment, using the same architecture as the *Walk straight* skill.
3) **Big Policy:** To show that our improved performance is not a byproduct of architecture size, we train a policy that uses the same neural network architecture as our Interactive Reach policy including sub-networks, but is trained from scratch.
4) **MCP+Residual:** To emphasize the importance of residual penalties, we compare against a naive combination of MCP and Residual learning by removing auxiliary regularization losses.

	Success Rate			
	Reach Target (Easy)	Open Door (Hard)	Push Object	Interactive Reach
Vanilla	0.96	0	0.04	0.46
Curriculum	0.92	**0.98**	0.52	0.48
Big Policy	0.92	0	0.01	0.22
MCP+Residual	**0.98**	0.12	0	0.48
CCRL	**0.98**	**0.98**	**0.89**	**0.72**

TABLE II: Comparison of Success Rates in Simulation

	Symmetry Index (%)	Average Torque ($N \cdot m$)
Vanilla	0.30	6.72
Curriculum	0.26	7.21
Big Policy	0.29	3.33
MCP+Residual	0.28	6.05
CCRL	**0.18**	**2.84**

TABLE III: Comparison of Trajectory Quality.

We train all policies until convergence and employ early stopping when required. To train all the skills in our framework we used a total of 3 billion environment steps. For most of the baselines, we stopped seeing any improvement in the reward or success rate beyond 0.5 billion environment steps for tasks in levels 1-3. For our Interactive reach task, we trained all our policies for 1.5 billion steps after which there is no noticeable improvement in performance. We compare the success rates on multiple tasks in Table II. Our approach, CCRL, typically outperforms all the other three baselines, Vanilla, Curriculum, and Big Policy, on challenging tasks. For the *Door Open (Hard)*, only CCRL and Curriculum achieve good success rates (98%) while Vanilla and Big policy fail. In our experience, it is almost impossible to learn an effective door-opening skill without a proper curriculum, such as learning to walk and open a door. However, for the most complicated task of Interactive Reach, CCRL shows the best success rate of 72% while all three baselines only show less than 48% success rates. This is because Interactive Reach requires careful coordination of all multiple different skills, navigation, crawling, obstacle avoidance, and door manipulation, which shows the importance of cascaded skill learning. For rest of the tasks our method reaches a success rate of 1.0 achieving the goal in every episode.

Control over Quality. The cascaded architecture also provides a natural way for controlling the style of the motion. For instance, in the case of *Reach Target (easy)*, the robot might jump toward the target or wiggle its legs to move gradually toward the target. While these behaviors might be successful in simulation, they do not transfer well to the real world as they are over-fitted to the simulation dynamics and parameters. To evaluate the quality of trajectories generated by the policy, we use two metrics: Similarity Index (SI)[48]

	Turn	Crawl	Reach Target	Door Open	Push Object
Curriculum	0.2	0	0	0	0
CCRL	**1**	**1**	**0.9**	**0.9**	**0.8**

TABLE IV: Success Rates on Hardware Experiments.

and average torque. The SI is defined as $SI\left(X_L, X_R\right) = 2\left|X_L - X_R\right| / (X_L + X_R)$, where X_L and X_R are the average torques used by the left and right motors.

We evaluate these metrics on *Reach Target (Easy)* in Table III for a fixed target location. CCRL uses a more symmetric gait due to its grounding on the core skills through the residual penalties and the lowest average torque when compared to baselines.

D. Simulation Analysis

This section intends to analyze the sensitivity of our learning framework, CCRL, by comparing the performance changes against the original formulation.

Sensitivity to bad parent skills. We first investigate the robustness of the proposed CCRL to check if it can learn to ignore a bad parent policy with low rewards. To this end, we learn a new policy on the *Reach Target (Easy)* task, but add one more expert policy that is randomly initialized and untrained. We notice that this policy performs similarly to our best policy (CCRL) in Table II, reaching a success rate of 98%. This showcases the robustness of CCRL, in ignoring bad parent policies.

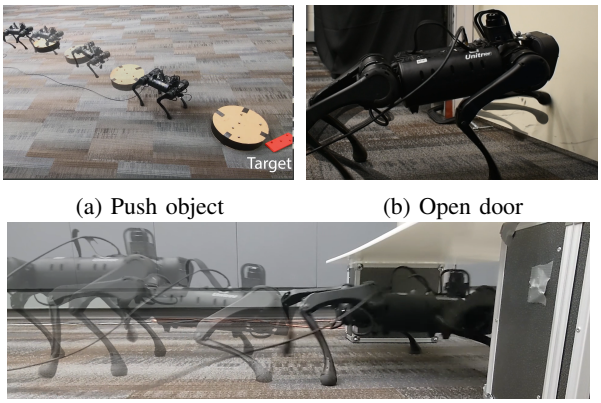
Sensitivity to irrelevant parent skills. We also check whether CCRL is sensitive to the design of the skill graph. Particularly, we investigate if CCRL can be robust to additional unnecessary skill dependencies, i.e., additional edges. We retrain a new policy to walk in a straight line with a redundant set of skills: {*Walk straight*, *Turn left*, *Turn right*}. Our framework ignores the redundancy and walks straight achieving an average $velocity$ tracking error of about 16%, same as the parent walking skill.

Importance of auxiliary loss. We further examine the importance of residual penalties ($L_t{}^{\text{rm}}$,$L_t{}^{\text{rw}}$) by retraining the *Reach Target (Easy)* policy without residual penalties ($c_3, c_4 = 0$). With this setup, the policy ends up overusing residual actions to learn behaviors that diverge too much from the base walking skill. While it still reaches the target in many cases, the quality of trajectories is poor and unsuitable sim-to-real transfer. Please refer to the supplemental video for comparison.

Importance of residual actions. The importance of residual actions varies per task. For instance, *Push Object* or *Open Door*, show a zero percent success rate if we ablate the residual actions. On the other hand, *Reach Target* does not rely on residual actions by achieving 96% success that is only 2% lower than the original.

E. Hardware Experiments

We transfer our policies trained in simulation to a real Unitree A1 quadruped robot. We use motion capture to identify and track the position of the robot and objects in the real world. Our policies do not require any additional finetuning on a real robot, demonstrating the robustness of the skills generated using CCRL.



(a) Push object (b) Open door

(c) Crawling

Fig. 5: We transfer the skills, (a) *Push object*, (b) *Open door* and (c) *Crawling* to the real world without any finetuning.

In Table IV, we show the success rate of our method compared to our strongest baseline, Curriculum learning. The other baselines were not successful on hardware. For each skill, we ran every policy 10 times on the real robot. Our method demonstrates robust and reliable sim2real transfer, achieving a high success rate (0.8) on even challenging tasks like *Push object*. Curriculum learning is unable to transfer successfully to even the simplest of task like *Turn*, when being subject to the same level of domain randomization.

In Fig. 1 we show an interactive navigation task, where the robot crawls under a desk, reaches a door, pushes it open with its leg, and walks through it to reach a target. We show example motions of additional skills such as *Push Object*, *Open Door (Easy)* and *Crawling* in Fig. 5.

V. CONCLUSION

In this paper, we presented a novel Cascaded Compositional Residual Learning (CCRL) framework that recursively learns policies by grounding them to a set of prerequisite skills learned in previous iterations. Using CCRL, we built interactive motor controllers on a high-dimensional quadrupedal robot while ensuring that the learned policies follow a style grounded in the parent skill library. We compared the policies learned to multiple baselines and showed the effectiveness of the proposed framework. We also demonstrated sim-to-real transfer of the learned motor skills.

For future work, we aim to automatically discover skill relationships instead of manually designing the skill graph. Additionally, our current policies adapt the behaviors based on the motion capture and memorization over the fixed layout. We hope to extend our approach to vision-based obstacle avoidance tasks by learning image-conditioned residual perturbations to pre-trained navigation skills. Additionally, we want to explore interaction policies that learn about objects [49] and adapt to the variability seen in the world.

REFERENCES

[1] L. Sentis and O. Khatib, "A whole-body control framework for humanoids operating in human environments," in *ICRA 2006.* IEEE, 2006, pp. 2641–2648.

[2] H. Ferrolho, V. Ivan, W. Merkt, I. Havoutis, and S. Vijayakumar, "Roloma: Robust loco-manipulation for quadruped robots with arms," *arXiv preprint arXiv:2203.01446*, 2022.

[3] S. Zimmermann, R. Poranne, and S. Coros, "Go fetch!-dynamic grasps using boston dynamics spot with external robotic arm," in *2021 IEEE ICRA*. IEEE, 2021, pp. 4488–4494.

[4] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "Deep reinforcement learning: A brief survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] S. Kim, M. Sorokin, J. Lee, and S. Ha, "Human motion control of quadrupedal robots using deep reinforcement learning," *RSS*, 2022.

[7] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," in *RSS*, Pittsburgh, Pennsylvania, June 2018.

[8] 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, p. eabk2822, 2022.

[9] A. Kumar, Z. Fu, D. Pathak, and J. Malik, "Rma: Rapid motor adaptation for legged robots," in *Robotics: Science and Systems*, 2021.

[10] C. Yang, K. Yuan, Q. Zhu, W. Yu, and Z. Li, "Multi-expert learning of adaptive legged locomotion," *Science Robotics*, vol. 5, no. 49, 2020.

[11] C. Li, F. Xia, R. Martin-Martin, and S. Savarese, "Hrl4in: Hierarchical reinforcement learning for interactive navigation with mobile manipulators," in *CoRL*. PMLR, 2020, pp. 603–616.

[12] X. B. Peng, M. Chang, G. Zhang, P. Abbeel, and S. Levine, "Mcp: Learning composable hierarchical control with multiplicative compositional policies," *NeurIPS*, vol. 32, 2019.

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

[14] T. Johannink, S. Bahl, A. Nair, J. Luo, A. Kumar, M. Loskyll, J. A. Ojea, E. Solowjow, and S. Levine, "Residual reinforcement learning for robot control," in *2019 ICRA*. IEEE, 2019, pp. 6023–6029.

[15] T. Apgar, P. Clary, K. Green, A. Fern, and J. W. Hurst, "Fast online trajectory optimization for the bipedal robot cassie." in *Robotics: Science and Systems*, vol. 101, 2018, p. 14.

[16] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *IROS 2016*.

[17] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing, and S. Kim, "Mit cheetah 3: Design and control of a robust, dynamic quadruped robot," in *IROS*. IEEE, 2018, pp. 2245–2252.

[18] 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 *IROS*. IEEE, 2018, pp. 1–9.

[19] M. H. Raibert, "Trotting, pacing and bounding by a quadruped robot," *Journal of biomechanics*, vol. 23, pp. 79–98, 1990.

[20] D. Kim, J. Di Carlo, B. Katz, G. Bledt, and S. Kim, "Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control," *arXiv preprint arXiv:1909.06586*, 2019.

[21] Z. Fu, A. Kumar, J. Malik, and D. Pathak, "Minimizing energy consumption leads to the emergence of gaits in legged robots," in *CoRL*, 2021.

[22] Z. Fu, A. Kumar, A. Agarwal, H. Qi, J. Malik, and D. Pathak, "Coupling vision and proprioception for navigation of legged robots," in *Proceedings of the IEEE/CVF CVPR*, 2022, pp. 17 273–17 283.

[23] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019.

[24] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *CoRL*. PMLR, 2022, pp. 91–100.

[25] W. Yu, J. Tan, Y. Bai, E. Coumans, and S. Ha, "Learning fast adaptation with meta strategy optimization," *IEEE RAL*, 2020.

[26] S. Ha, P. Xu, Z. Tan, S. Levine, and J. Tan, "Learning to walk in the real world with minimal human effort," in *CoRL*, 2020.

[27] L. Smith, J. C. Kew, X. B. Peng, S. Ha, J. Tan, and S. Levine, "Legged robots that keep on learning: Fine-tuning locomotion policies in the real world," in *2022 ICRA*. IEEE, 2022, pp. 1593–1599.

[28] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva, *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[29] E. Wijmans, A. Kadian, A. Morcos, S. Lee, I. Essa, D. Parikh, M. Savva, and D. Batra, "Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames," in *ICLR*, 2020.

[30] M. Sorokin, W. Yu, S. Ha, and C. K. Liu, "Learning human search behavior from egocentric visual inputs," in *Computer Graphics Forum*, vol. 40, no. 2. Wiley Online Library, 2021, pp. 389–398.

[31] D. S. Chaplot, D. P. Gandhi, A. Gupta, and R. R. Salakhutdinov, "Object goal navigation using goal-oriented semantic exploration," *NeurIPS*, vol. 33, pp. 4247–4258, 2020.

[32] D. Batra, A. Gokaslan, A. Kembhavi, O. Maksymets, R. Mottaghi, M. Savva, A. Toshev, and E. Wijmans, "Objectnav revisited: On evaluation of embodied agents navigating to objects," *arXiv preprint arXiv:2006.13171*, 2020.

[33] F. Xia, W. B. Shen, C. Li, P. Kasimbeg, M. E. Tchapmi, A. Toshev, R. Martín-Martín, and S. Savarese, "Interactive Gibson Benchmark: A Benchmark for Interactive Navigation in Cluttered Environments," *IEEE RAL*, vol. 5, no. 2, pp. 713–720, 2020.

[34] K.-H. Zeng, L. Weihs, A. Farhadi, and R. Mottaghi, "Pushing it out of the way: Interactive visual navigation," in *CVPR*, 2021, p. 9868.

[35] G. Konidaris, S. Kuindersma, R. Grupen, and A. Barto, "Autonomous skill acquisition on a mobile manipulator," in *AAAI*, 2011.

[36] K. N. Kumar, I. Essa, and S. Ha, "Graph-based cluttered scene generation and interactive exploration using deep reinforcement learning," in *2022 ICRA*. IEEE, 2022, pp. 7521–7527.

[37] M. Danielczuk, A. Angelova, V. Vanhoucke, and K. Goldberg, "X-ray: Mechanical search for an occluded object by minimizing support of learned occupancy distributions," *IROS*, pp. 9577–9584, 2020.

[38] H. Huang, M. Dominguez-Kuhne, V. Satish, M. Danielczuk, K. Sanders, J. Ichnowski, A. Lee, A. Angelova, V. Vanhoucke, and K. Goldberg, "Mechanical search on shelves using lateral access x-ray," in *2021 IROS*, 2021, pp. 2045–2052.

[39] P.-L. Bacon, J. Harb, and D. Precup, "The option-critic architecture," in *AAAI*, vol. 31, no. 1, 2017.

[40] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[41] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–35, 2021.

[42] T. Li, N. Lambert, R. Calandra, F. Meier, and A. Rai, "Learning generalizable locomotion skills with hierarchical reinforcement learning," in *2020 IEEE ICRA*. IEEE, 2020, pp. 413–419.

[43] D. Jain, A. Iscen, and K. Caluwaerts, "Hierarchical reinforcement learning for quadruped locomotion," in *IROS 2019*, pp. 7551–7557.

[44] N. Heess, G. Wayne, Y. Tassa, T. Lillicrap, M. Riedmiller, and D. Silver, "Learning and transfer of modulated locomotor controllers," *arXiv preprint arXiv:1610.05182*, 2016.

[45] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *TOG*, vol. 36, no. 4, pp. 1–13, 2017.

[46] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[47] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, and G. State, "Isaac gym: High performance GPU based physics simulation for robot learning," in *NeurIPS Datasets and Benchmarks Track*, 2021.

[48] W. Yu, G. Turk, and C. K. Liu, "Learning symmetric and low-energy locomotion," *ACM Transactions on Graphics (TOG)*, 2018.

[49] K. N. Kumar, I. Essa, S. Ha, and C. K. Liu, "Estimating mass distribution of articulated objects using non-prehensile manipulation," *arXiv preprint arXiv:1907.03964*, 2019.