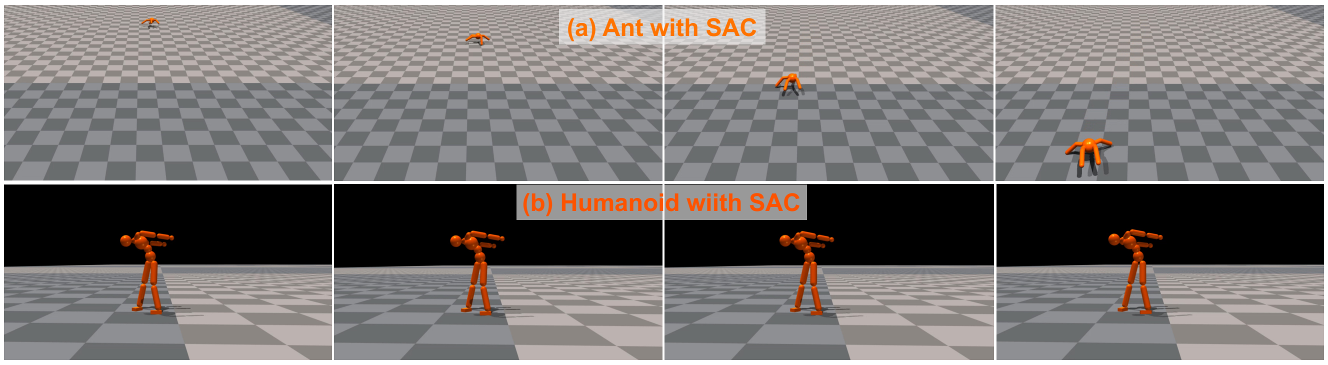


Figure A1. **Training standard RL environments with GROVE using SAC.** (a) The Ant agent executing "running forward" behavior—note the consistent forward progress across frames. (b) The Humanoid agent performing a "bow at a right angle"—observe how the agent maintains balance while achieving the specified angle. These results demonstrate that GROVE effectively guides agent behavior regardless of whether PPO or SAC is used as the underlying RL algorithm.

A. Additional Experimental Results

A.1. Open-Vocabulary Humanoid Skill Acquisition

While our main paper presents compelling quantitative evidence of GROVE's effectiveness, the qualitative differences become even more striking when visualized. In Figs. A3 and A4, we showcase these differences across nine distinct approaches: (i) MoMask [9], (ii) MotionGPT [13], (iii) TMR [34], (iv) AvatarCLIP [11], (v) AnySkill [8], (vi) VLM+LLM, (vii) LLM only, (viii) Pose2CLIP only, and (ix) our full GROVE framework.

To thoroughly evaluate generalization capabilities, we challenged each method with diverse open-vocabulary instructions: (i) "playing the suona", (ii) "running while jumping hurdle", (iii) "conduct the orchestra", (iv) "walking like a model", and (v) "position body in a shape of 'C'". The results reveal patterns across model categories. TMR, despite its sophistication, consistently fails to generate appropriate responses—a clear indication that these open-vocabulary prompts lie beyond the distribution of current text-motion datasets. MoMask demonstrates competence with familiar actions like "jumping hurdles" but falters when confronted with more novel or culturally specific instructions such as "playing the suona." Similarly, MotionGPT tends to produce conservative motions that only partially capture the intended behaviors, often defaulting to upright, small-amplitude movements that lack expressivity. AvatarCLIP likewise demonstrates limitations in translating textual instructions into fluid, meaningful motions.

Pushing the boundaries further, we designed an additional challenge set of instructions carefully crafted to satisfy two critical criteria:

- They emerge from an LLM's understanding of natural human movement patterns.
- They verifiably exist outside the domain of any open-source human text-motion dataset.

This design principle ensures that success cannot be attributed to mere retrieval of existing motion data—a true test of generative understanding. The challenge set includes diverse instructions such as "jump rope," "walking while sipping water," "swim with two arms," "throw a ball, one hand scratch forward," "hurrah with two arms," and "jump in place." These prompts require ian ntegrated understanding of physics, biomechanics, and semantic intent.

To capture the multidimensional nature of motion quality, our human evaluation framework employs three complementary metrics, each rated on a 0–10 scale:

- **Task completion:** Measures semantic fidelity—how faithfully the motion embodies the instruction's intent. Higher scores reflect more accurate realization of the specified action.
- **Motion naturalness:** Evaluates kinematic plausibility—the smoothness and continuity of movement patterns. This metric penalizes jarring transitions, unnatural accelerations, or anatomically implausible configurations.
- **Physics:** Assesses physical realism—how well the motion respects fundamental physical constraints. Higher scores indicate fewer artifacts like floating, ground penetration, or impossible joint relationships.

Together, these metrics provide a holistic assessment framework that aligns with human perception of motion quality. The framework reveals GROVE's ability to generate motions that not only accomplish the specified task but do so with natural, physically plausible movements—a significant advancement over existing approaches. For a more visceral understanding of these qualitative differences, we encourage readers to explore the additional videos and interactive visualizations available on our project website.

A.2. Standard RL Benchmarks

Beyond the standard RL benchmark results presented in our main paper, we conducted additional experiments to evaluate `GROVE`'s compatibility with alternative RL algorithms. While our primary results utilize Proximal Policy Optimization (PPO), we also tested our approach with Soft Actor-Critic (SAC), a different RL algorithm with distinct optimization characteristics.

For this comparison, we focused on two challenging environments from our benchmark suite: Humanoid and Ant. These environments feature complex dynamics and high-dimensional action spaces that provide a rigorous test of our framework's capabilities. To ensure direct comparability, we maintained the same text instructions used in our PPO experiments: "Humanoid bows at a right angle" for the Humanoid environment and "Ant is running forward" for the Ant environment.

The results, visualized in Fig. A1, demonstrate that `GROVE` successfully enables agents to perform the actions specified by the text instructions across both RL algorithms. This consistency across different optimization methods suggests that our multimodal reward framework provides effective guidance regardless of the underlying algorithm choice.

These findings complement our main results by showing that `GROVE`'s approach to open-vocabulary skill acquisition generalizes beyond a single RL algorithm, further supporting its utility as a flexible framework for teaching diverse behaviors to simulated agents.

B. LLM-based Reward

This section provides a comprehensive overview of our approach to designing and implementing LLM-based reward functions. We describe both the carefully engineered prompts that elicit effective reward functions and the resulting outputs from various LLM models. Our prompt engineering process addresses several critical requirements for embodied reinforcement learning with natural language guidance.

B.1. Prompt Input

Prompt for pre-trained controllers

You are a reward engineer trying to write reward functions to solve RL tasks as effectively as possible.
Your goal is to write a reward function for the environment that will help a humanoid character learn the task described in the text. The humanoid character is a physics-based character with 15 joints. There are other components in the model that keep the humanoid upright and moving like a human being, so your job is only to write a reward function that captures the essence of the described task.
The joint order for the humanoid is as follows:

```
SMPL_BONE_ORDER_NAMES = [
    "pelvis",
    "torso",
    "head",
    "right_upper_arm",
    "right_lower_arm",
    "right_hand",
    "left_upper_arm",
    "left_lower_arm",
    "left_hand",
    "right_thigh",
    "right_shin",
    "right_foot",
    "left_thigh",
    "left_shin",
    "left_foot"
]
```

In the simulator, we define the z as the up-axis.
Your reward function should use useful variables from the environment as inputs. As an example, the reward function signature can be:

```
@torch.jit.script
def compute_llm_reward():
    body_pos = infos["state_embeds"][:, :21, :3]  # [num, 21, 3]
```

```python
    body_rot = infos["state_embeds"][:, :21, 3:7]  # [num, 21, 4]
    body_vel = infos["state_embeds"][:, :21, 7:10]  # [num, 21, 3]
    body_ang_vel = infos["state_embeds"][:, :21, 10:13]  # [num, 21, 3]
    ...
    return reward, {}
```

You can parse each joint's position, rotation, velocity, and angular velocity from the tensors as follows:

```python
pelvis_pos, pelvis_rot, pelvis_vel, pelvis_ang_vel = body_pos[:, 0, :], body_rot[:, 0, :],
    body_vel[:, 0, :], body_ang_vel[:, 0, :]
torso_pos, torso_rot, torso_vel, torso_ang_vel = body_pos[:, 1, :], body_rot[:, 1, :],
    body_vel[:, 1, :], body_ang_vel[:, 1, :]
head_pos, head_rot, head_vel, head_ang_vel = body_pos[:, 2, :], body_rot[:, 2, :], body_vel
    [:, 2, :], body_ang_vel[:, 2, :]
...
```

the order is the same as the `SMPL_BONE_ORDER_NAMES`.
Since the reward function will be decorated with @torch.jit.script, please make sure that the code is compatible with TorchScript (e.g., use torch tensor instead of numpy array).
Make sure any new tensor or variable you introduce is on the same device as the input tensors. The output of the reward function should consist of two items:

- the total reward,
- a dictionary of each individual reward component.

The code output should be formatted as a python code string: ` ``` python ... ``` `.
Some helpful tips for writing the reward function code:

1. You may find it helpful to normalize the reward to a fixed range by applying transformations like torch.exp to the overall reward or its components
2. If you choose to transform a reward component, then you must also introduce a temperature parameter inside the transformation function; this parameter must be a named variable in the reward function and it must not be an input variable. Each transformed reward component should have its own temperature variable
3. Make sure the type of each input variable is correctly specified; a float input variable should not be specified as torch.Tensor
4. Most importantly, the reward codeś input variables must contain only attributes of the provided environment class definition (namely, variables that have prefix self.). Under no circumstance can you introduce new input variables.

Write a reward function for the following task: **open-vocabulary instruction**

B.2. LLM Reward Generated by Different Baselines

To determine the most effective LLM for reward generation, we conducted a systematic comparison using multiple language instructions. Here, we examine the outputs for a representative example: "boxing with two arms." We evaluated three prominent LLM models by analyzing their generated reward functions for correctness, physical plausibility, and alignment with the intended behavior. The evaluation extended beyond this single example to include more than five diverse text prompts that tested different aspects of motion understanding.

Our analysis revealed that while all tested models produced semantically reasonable reward structures, they differed significantly in their handling of physical constraints and motion dynamics. Specifically, *GPT-4o* demonstrated superior performance by consistently avoiding physically impossible reward terms or biomechanically unrealistic motion specifications.

Based on these comparative results, we selected *GPT-o1-preview* as our primary LLM-based reward generator for all experiments reported in this paper. This choice ensures that our baseline comparisons reflect the strongest possible implementation of pure LLM-based reward approaches.

Reward generated by GPT-o1-preview

```python
@torch.jit.script
def compute_llm_reward(self):
    body_pos = self.infos["state_embeds"][:, :15, :3]  # [num, 15, 3]
    body_rot = self.infos["state_embeds"][:, :15, 3:7]  # [num, 15, 4]
```

```
    body_vel = self.infos["state_embeds"][:, :15, 7:10]  # [num, 15, 3]
    body_ang_vel = self.infos["state_embeds"][:, :15, 10:13]  # [num, 15, 3]

    # Extract positions
    torso_pos = body_pos[:, 1, :]          # "torso" is at index 1
    left_hand_pos = body_pos[:, 8, :]      # "left_hand" is at index 8
    right_hand_pos = body_pos[:, 5, :]     # "right_hand" is at index 5

    # Compute distances to torso
    left_hand_to_torso_dist = torch.norm(left_hand_pos - torso_pos, dim=1)
    right_hand_to_torso_dist = torch.norm(right_hand_pos - torso_pos, dim=1)

    # Compute distance between hands
    hands_distance = torch.norm(left_hand_pos - right_hand_pos, dim=1)

    # Temperature parameters for transformations
    temp_hand_torso = 0.2
    temp_hands = 0.2

    # Compute reward components using exponential decay based on distances
    left_hand_reward = torch.exp(-left_hand_to_torso_dist / temp_hand_torso)
    right_hand_reward = torch.exp(-right_hand_to_torso_dist / temp_hand_torso)
    hands_proximity_reward = torch.exp(-hands_distance / temp_hands)

    # Total reward is the average of the three components
    llm_rewards = (left_hand_reward + right_hand_reward + hands_proximity_reward) / 3.0

    # Dictionary of individual reward components
    reward_components = {
        "left_hand_reward": left_hand_reward,
        "right_hand_reward": right_hand_reward,
        "hands_proximity_reward": hands_proximity_reward
    }

    return llm_rewards
```

Reward generated by GPT-4o

To create a reward function for the humanoid to perform boxing, we need to focus on rewarding actions that resemble boxing movements. This involves encouraging the humanoid to use its arms (specifically the shoulders, elbows, and wrists) actively, maintain balance, and possibly target specific points in space to simulate punches.
Here's an example reward function:

```
@torch.jit.script
def compute_boxing_reward(
    self
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    body_pos = self._rigid_body_pos  # [num, 15, 3]
    body_rot = self._rigid_body_rot  # [num, 15, 4]
    body_vel = self._rigid_body_vel  # [num, 15, 3]
    body_ang_vel = self._rigid_body_ang_vel  # [num, 15, 3]

    left_shoulder_vel = body_vel[:, 16, :]
    right_shoulder_vel = body_vel[:, 17, :]
    left_elbow_vel = body_vel[:, 18, :]
    right_elbow_vel = body_vel[:, 19, :]
    left_wrist_vel = body_vel[:, 20, :]
    right_wrist_vel = body_vel[:, 21, :]

    # Reward for arm movement (boxing punches)
    arm_movement_reward = (
        torch.norm(left_shoulder_vel, dim=1)
        + torch.norm(right_shoulder_vel, dim=1)
        + torch.norm(left_elbow_vel, dim=1)
        + torch.norm(right_elbow_vel, dim=1)
        + torch.norm(left_wrist_vel, dim=1)
        + torch.norm(right_wrist_vel, dim=1)
    )

    # Maintain balance by keeping pelvis stable
    pelvis_pos = body_pos[:, 0, :]
    pelvis_stability_reward = -torch.abs(pelvis_pos[:, 2] - 1.0)  # Assuming 1.0 is the
        desired pelvis height

    # Encourage facing forward by aligning the pelvis rotation with the forward direction
    forward_direction = torch.tensor([0.0, 0.0, 1.0], device=body_rot.device)
    pelvis_rot = body_rot[:, 0, :]
    facing_reward = torch.sum(pelvis_rot[:, :3] * forward_direction, dim=1)

    # Transform rewards with temperature parameters
    arm_movement_temp = 0.1
    stability_temp = 0.1
    facing_temp = 0.1

    arm_movement_reward = torch.exp(arm_movement_reward * arm_movement_temp)
    pelvis_stability_reward = torch.exp(pelvis_stability_reward * stability_temp)
    facing_reward = torch.exp(facing_reward * facing_temp)

    # Total reward
    total_reward = arm_movement_reward + pelvis_stability_reward + facing_reward

    reward_components = {
        "arm_movement_reward": arm_movement_reward,
        "pelvis_stability_reward": pelvis_stability_reward,
        "facing_reward": facing_reward,
    }

    return total_reward, reward_components
```

Reward generated by Gemini-1.5-Pro

```
@torch.jit.script
def compute_llm_reward(
    body_pos: torch.Tensor,
    body_rot: torch.Tensor,
    body_vel: torch.Tensor,
    body_ang_vel: torch.Tensor,
) -> Tuple[torch.Tensor, Dict[str, torch.Tensor]]:
    """
    Reward function for making the humanoid box.

    Args:
        body_pos: Position of each body part.
        body_rot: Rotation of each body part.
        body_vel: Velocity of each body part.
        body_ang_vel: Angular velocity of each body part.

    Returns:
        A tuple containing the total reward and a dictionary of reward components.
    """
    device = body_pos.device

    # Extract left and right hand positions and velocities.
    left_hand_pos = body_pos[:, SMPL_BONE_ORDER_NAMES.index("left_hand"), :]
    right_hand_pos = body_pos[:, SMPL_BONE_ORDER_NAMES.index("right_hand"), :]
    left_hand_vel = body_vel[:, SMPL_BONE_ORDER_NAMES.index("left_hand"), :]
    right_hand_vel = body_vel[:, SMPL_BONE_ORDER_NAMES.index("right_hand"), :]

    # Reward for hands close together (forming a boxing stance).
    hand_dist = torch.norm(left_hand_pos - right_hand_pos, dim=-1)
    hands_together_reward = torch.exp(-hand_dist * 2.0)

    # Reward for hands at head level.
    head_pos = body_pos[:, SMPL_BONE_ORDER_NAMES.index("head"), :]
    left_hand_head_diff = torch.abs(left_hand_pos[:, 1] - head_pos[:, 1]) # y-axis difference
    right_hand_head_diff = torch.abs(right_hand_pos[:, 1] - head_pos[:, 1]) # y-axis
        difference
    hands_at_head_level_reward = torch.exp(-(left_hand_head_diff + right_hand_head_diff) *
        2.0)


    # Reward for minimizing hand velocity (stable boxing stance).
    left_hand_vel_magnitude = torch.norm(left_hand_vel, dim=-1)
    right_hand_vel_magnitude = torch.norm(right_hand_vel, dim=-1)
    hands_velocity_reward = torch.exp(-(left_hand_vel_magnitude + right_hand_vel_magnitude) *
         0.5)

    # Combine reward components.
    reward = hands_together_reward + hands_at_head_level_reward + hands_velocity_reward

    # Reward dictionary.
    reward_dict = {
        "hands_together_reward": hands_together_reward,
        "hands_at_head_level_reward": hands_at_head_level_reward,
        "hands_velocity_reward": hands_velocity_reward,
    }

    return reward, reward_dict
```

C. Additional Details of Experiments

C.1. Ablative Analysis of RDP

Drawing inspiration from Eureka [26], we formulate GROVE as a Reward Design Problem (RDP) [39] where a VLM provides feedback on the quality of language-generated rewards. In our framework, a VLM-based reward R_V acts as an evaluator for the LLM-based reward R_L, establishing a mutually reinforcing relationship between the two reward components.

To implement this interaction effectively, we develop a dynamic reward regeneration mechanism. Specifically, we trigger the regeneration of the LLM-based reward R_L when we detect a sustained decline in performance—defined as eight consecutive steps with decreasing average R_V across all parallel environments, with the latest average falling below the threshold of 0.1. This rejection-based sampling approach serves as a critical quality control mechanism, preventing the optimization of R_L from diverging toward behaviors that may be mathematically optimal but visually inconsistent with the specified instruction.

To evaluate the importance of this RDP mechanism, Fig. A2 presents systematic qualitative comparisons across five diverse text commands: (i) "running while jumping hurdle", (ii) "playing the suona," (iii) "walking like a model," (iv) "position body in a shape of 'C'," and (v) "conduct the orchestra." Our project website provides additional examples for a comprehensive review.

The ablation results reveal two critical insights about the RDP mechanism. First, eliminating the rewriting component significantly reduces the effectiveness of constraints for certain instructions, causing generated behaviors to deviate from the specified text requirements. Second, actions generated without RDP frequently resemble those produced by the Pose2CLIP component in isolation, indicating insufficient integration of the LLM's semantic understanding. This lack of synergistic improvement—where the combined reward fails to outperform its individual components—underscores the importance of the RDP mechanism in GROVE's design for achieving robust instruction-following behavior.

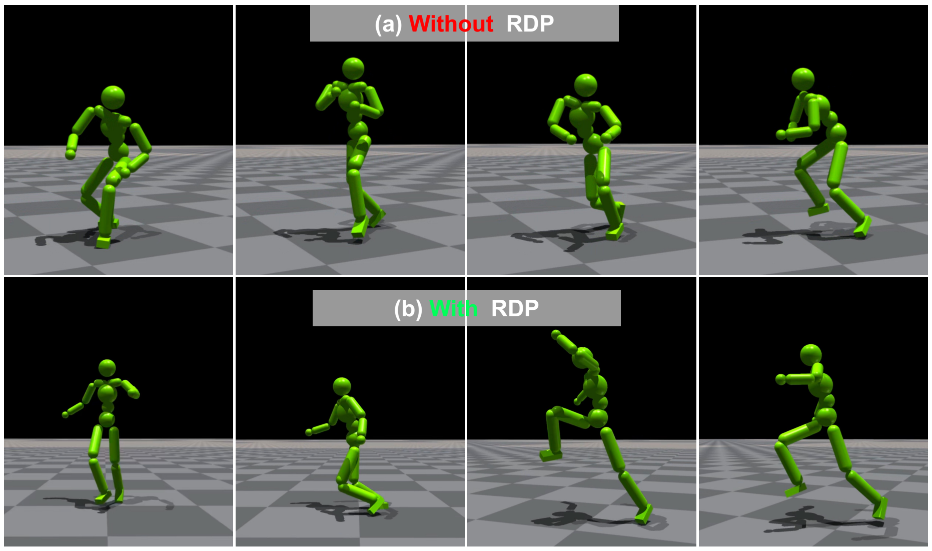


Figure A2. **Comparative performance with and without RDP.** We illustrate the qualitative difference in motion execution for the instruction "running while jumping hurdle." **(a) Without RDP:** The agent exhibits limited vertical displacement and fails to perform a proper hurdle-clearing motion, instead executing a minimal hop that would be insufficient to clear an obstacle. Note the relatively flat trajectory and diminished preparation phase. **(b) With RDP:** The agent demonstrates significantly improved biomechanics with proper jump preparation, extension during flight, and recovery phases. Observe the pronounced knee lift, extended flight phase, and appropriate body orientation—all critical elements of successful hurdle clearance. This comparison highlights how RDP substantively improves motion quality by dynamically refining LLM-based rewards through VLM feedback, resulting in semantically accurate and physically plausible behaviors.

C.2. Quantifying Policy Convergence with Expert Reward Distance

To quantitatively evaluate policy learning efficiency across different reward formulations, we developed a standardized metric called reward distance, reported in Tab. 2. This metric measures how quickly a policy converges to optimal behavior according to task-specific expert criteria, regardless of the actual reward function used during training.

For each RL benchmark, we first define an expert reward function that captures the fundamental objective of the task. For instance, in the *Ant Running Forward* task, the expert reward is expressed as $r_{\text{expert}} = v_y$, where v_y represents the agent's forward velocity—a direct measure of how well the agent accomplishes the primary goal of forward movement.

Importantly, we track these expert reward values independently from whatever reward function is actually used during policy training. This allows us to fairly compare different reward formulations based on their effectiveness at accomplishing the core task objective. The reward distance is then computed as the area above the expert reward curve throughout training. Lower values indicate faster convergence to optimal behavior, providing a quantitative measure of training efficiency that enables direct comparison across different reward approaches.

C.3. Human Evaluation of Motion Quality and Task Alignment

To rigorously evaluate the perceptual quality of generated motions beyond computational metrics, we conducted a comprehensive user study involving 30 participants with diverse backgrounds and varying levels of familiarity with computer animation and motion synthesis.

We employed a within-subjects experimental design, where each participant evaluated multiple motion sequences generated by different methods. To ensure methodological rigor, we incorporated several controls: (i) repeated items to assess participant consistency, (ii) randomized presentation order to mitigate sequence effects, and (iii) balanced exposure to different motion types across participants to control for potential biases.

The study was implemented through a custom web-based platform that presented motion sequences with standardized viewing angles and playback speeds. To minimize evaluation fatigue and maintain data quality, the system assigned each participant a unique task sequence with optimized pacing and appropriate rest intervals. Participants rated motions on multiple dimensions including semantic accuracy, physical plausibility, and overall naturalness, providing a multifaceted assessment of motion quality that complements our quantitative metrics.

Table A1. **Dataset Composition for `Pose2CLIP` Training.** We detail the diverse sources and sampling strategies used to create `Pose2CLIP` training dataset. While AMASS and IDEA400 provide the foundation with high-quality motion capture data (together contributing over 36 hours), we incorporated additional Motion-X subsets at full resolution to capture specialized movement patterns. To enhance robustness to simulation-specific poses and reduce distributional shift during deployment, we augmented the dataset with frames sampled directly from our reinforcement learning training episodes. The varying sampling rates (1/5 for most sources) were implemented to balance computational efficiency with representation diversity while preventing overrepresentation of repetitive motions.

Dataset	Sample Rate	Frames	Total Hours
AMASS	1/5	357,728	16.56
IDEA400	1/5	424,144	19.64
Other Subsets	1	885,757	8.20
Training Cases	1/5	36,272	1.68

D. `Pose2CLIP`: Architecture and Implementation

D.1. Dataset Construction and Processing

The effectiveness of our `Pose2CLIP` model relies on its exposure to diverse and representative pose data. To achieve this, we constructed a comprehensive training dataset integrating multiple high-quality motion sources. Our primary data foundations include the SMPL model poses from the AMASS training split and the extensive Motion-X dataset, specifically incorporating IDEA400 and four additional specialized subsets (Animation, HuMMan, Kungfu, and Perform) that capture a wide range of human movements.

To enhance robustness to physical simulator artifacts, we implemented an iterative data enrichment strategy. Pose data generated during the reinforcement learning training process is automatically captured, re-oriented to canonical coordinates, and incorporated back into the model's training dataset. This approach creates a diverse learning environment that includes both naturalistic human movements from motion capture and poses that may deviate from typical human motion patterns, particularly those generated during early exploration phases of the reinforcement learning process.

To maintain computational efficiency and prevent overfitting to redundant examples, we applied a systematic downsampling procedure for deduplication, particularly for highly similar poses occurring in repetitive motions. The precise distribution of data sources and their relative contributions to our final training dataset are detailed in Tab. A1.

Our `Pose2CLIP` model is engineered specifically for the 15-joint kinematic skeleton used in our animated character, accepting pose inputs represented as $\theta \in \mathbb{R}^{15\times 3}$ (Euler angles for each joint). To extend the applicability of our approach to broader research applications, we additionally trained and open-sourced a variant called `Pose2CLIP`-M, specifically designed for the standard SMPL skeleton with 24 joints. This model accepts inputs in the form $p = (\theta^{SMPL}, h_{root})$, where $\theta^{SMPL} \in \mathbb{R}^{24\times 3}$ encodes the Euler angles of all 24 joints and h_{root} represents the height of the root joint relative to the ground plane.

D.2. Quantitative Evaluation of `Pose2CLIP`

`Pose2CLIP` directly maps humanoid poses to CLIP features, a capability developed by training on over 1.7 million pose-image feature pairs, covering a wide range of actions (*e.g.*, walking, punching, jumping, reclining). To assess the accuracy and completeness of the features learned from this extensive dataset, we construct a similarity matrix M.

Each row i of the matrix represents a pose p_i, and each column j represents a text description t_j. The element $m_{ij} = \text{sim}(p_i, t_j)$ in the matrix denotes the similarity between p_i and t_j. We established the ground truth by recruiting 25 participants to rate the similarity of each pose–description pair on a 0–1 scale. To evaluate whether the method can distinguish different actions under the same text description, we apply a linear transformation to each column so that its minimum and maximum values become 0 and 1, respectively. This normalization process yields the ground truth similarity matrix M_{GT}.

We evaluate six approaches using this framework: (i) IsaacGym's native rendering with CLIP, (ii) image prompt with CLIP (which enhances rendered images with a red circle to direct attention), (iii) text prompt with CLIP (which alters subject types in text descriptions), (iv) VLM-RM [37] (which modify CLIP text features to remove agent-specific details), (v) Blender Render with CLIP (utilizing high-fidelity, human-like rendering), and (vi) our proposed model. After undergoing the same normalization process as the ground truth, these methods yield their respective similarity matrices $M_k (k = 1, 2, ..., 6)$. We employ matrix similarity $\text{sim}(M_k, M_{GT}) = 1 - \dfrac{\sqrt{\sum (M_k - M_{GT})^2}}{\sqrt{\sum (M_k^2 + M_{GT}^2)}}$ to score the methods.

The comparison results in Tab. A2 reveal distinct performance tiers among the evaluated approaches. The four baseline methods—IsaacGym + CLIP [8], Image Prompt, Text Prompt, and VLM-RM [37]—show limited effectiveness with simi-

Table A2. **Comparative Performance Analysis of Pose-to-CLIP Feature Mapping Approaches.** We show matrix similarity scores between human-annotated ground truth and six distinct methods for deriving semantic embeddings from poses. Results demonstrate that while conventional rendering modifications (Image/Text Prompting, VLM-RM) yield minimal improvements over the baseline (IsaacGym + CLIP), photorealistic rendering (Blender + CLIP) achieves substantially higher semantic alignment. Notably, our `Pose2CLIP` attains equivalent performance (0.48 vs. 0.49) without requiring any rendering pipeline, validating our direct pose-to-embedding approach as both efficient and semantically accurate. These findings confirm that our model successfully distills the essential pose semantics comparable to high-fidelity visual representation.

Approach	IsaacGym + CLIP	Image Prompt	Text Prompt	VLM-RM	Blender + CLIP	Pose2CLIP
Matrix Score	0.37	0.38	0.38	0.36	**0.49**	**0.48**

larity scores tightly grouped between 0.36 and 0.38. This clustering suggests that simple adjustments to rendering or text prompting provide minimal benefit for semantic alignment.

Blender Render with CLIP achieves a score of 0.49 and demonstrates that high-quality visual representation substantially improves pose-text association accuracy. Remarkably, our `Pose2CLIP` achieves a nearly identical score of 0.48 while eliminating the rendering pipeline entirely. This performance parity validates our direct mapping approach and confirms that `Pose2CLIP` successfully extracts the essential semantic information from physical poses without computationally expensive rendering.

Our final `Pose2CLIP` implementation is highly efficient, with just 2.9 million parameters, and achieves an average cosine similarity of 0.86 between predicted and ground truth features on our validation set. This combination of accuracy and computational efficiency makes `Pose2CLIP` particularly suitable for real-time applications where rendering would introduce unacceptable latency.

E. Implementation Details of GROVE

To support reproducibility, we provide comprehensive implementation details for the key components of our framework.

E.1. LLM Prompt

Our systematically engineered LLM prompt incorporates seven critical components specifically designed to elicit high-quality reward functions:

1. **Role specification:** We position the LLM as an expert reward engineer specializing in robot learning and motion synthesis, establishing an appropriate technical context for generation.
2. **Goal definition:** We explicitly define the objective as designing a reward function for RL of a specified task, emphasizing the importance of minimal yet complete formulations.
3. **Environment description:** We provide detailed information about the simulation environment, including coordinate system conventions (e.g., up-axis direction), available physical quantities, and relevant state variables accessible to the reward function.
4. **Agent specification:** We outline the agent's morphology comprehensively, listing each joint by name and index to enable precise targeting of individual body parts within the reward function.
5. **Example template:** We supply a code skeleton demonstrating proper function signature, expected inputs/outputs, and basic structure, serving as a reference pattern for the generated function.
6. **Design principles:** We highlight core guidelines for effective reward design, stressing the importance of sparse, well-shaped rewards and noting that auxiliary components already manage stability and locomotion.
7. **Task instruction:** Finally, we include the verbatim natural language instruction I, which serves as the primary objective to be optimized by the reward function.

This structured approach significantly improves both the quality and consistency of generated reward functions compared to more generic prompting methods.

E.2. `Pose2CLIP`

Our `Pose2CLIP` implementation combines high-quality visual rendering with an efficient neural architecture design:

1. **Rendering pipeline:** We utilize Blender's EEVEE rendering engine, a physically-based real-time renderer, configured to produce 224×224 images that precisely match CLIP's input requirements. The setup incorporates realistic three-point lighting and employs physically accurate materials derived from the SMPL-X model.
2. **Neural architecture:** The `Pose2CLIP` model features a two-layer MLP with hidden layer dimensions [256, 1024], followed by a direct linear projection to the CLIP feature dimension. Between layers, we incorporate GELU activations

to enhance both training stability and generalization capability.

3. **Training configuration:** We train the model using Adam optimizer with a learning rate of 10^{-4} and a batch size of 512. The training process implements a cosine learning rate schedule with warm-up during the first 10% of training steps. All experiments run on a single NVIDIA 4090 GPU over eight hours, requiring approximately 18GB of GPU memory.

Despite its compact design of only 2.9 million parameters, the resulting model achieves remarkable fidelity in mapping between pose space and CLIP feature space, demonstrating an average cosine similarity of 0.86 between predicted and ground truth features on our validation set.

E.3. Low-level Controller

The hierarchical control system maintains consistent architectural patterns across its key components. The encoder, low-level control policy, and discriminator each employ MLP architectures with identical hidden layer dimensions [1024, 1024, 512]. For efficient representation, we utilize a 64-dimensional latent space $\mathcal{Z}$.

Critical hyperparameters detailed in Tab. A3 include an alignment loss weight of 0.1, uniformity loss weight of 0.05, and a gradient penalty coefficient of 5. We train the low-level controller using PPO [38] within the IsaacGym physics simulation environment. The training process runs on a single NVIDIA A100 GPU at a 120 Hz simulation rate over a four-day period, encompassing a diverse dataset of 93 distinct motion patterns to ensure coverage of the humanoid action space.

Table A3. **Hyperparameters for the training and operation of our hierarchical control system components.**

Low-level Controller configuration detailing neural architecture parameters, loss weightings, and PPO training settings.

Parameter	Value
dim(Z) Latent Space	64
Align Loss Weight	0.1
Uniform Loss Weight	0.05
w_{gp} Gradient Penalty	5
Encoder Regularization	0.1
Samples Per Update	131072
Policy/Value Minibatch	16384
Discriminator Minibatch	4096
γ Discount	0.99
Learning Rate	2×10^{-5}
GAE(λ)	0.95
TD(λ)	0.95
PPO Clip Threshold	0.2
T Episode Length	300

Standard RL benchmarks. Benchmark configuration with task-specific architecture dimensions and optimization parameters.

Parameter	Value
ANYmal & Ant MLP	[256, 128, 64]
Humanoid MLP	[400, 200, 100]
ANYmal & Ant LR	3×10^{-4}
Humanoid LR	5×10^{-4}
Activation	elu
γ Discount	0.99
KL_threshold	0.008
TD(λ)	0.95
PPO Clip Threshold	0.2
T Episode Length	300

E.4. Standard RL Benchmarks

For our standard RL benchmark evaluations, we implement control policies using MLPs with a consistent architecture of three hidden layers [400, 200, 100]. These policies are trained using the PPO algorithm [38] within the IsaacGym simulation environment. We maintain a learning rate of 5×10^{-4} throughout the training process and conduct all experiments on a single NVIDIA A100 GPU with a fixed simulation frequency of 60 Hz. Additional hyperparameters governing the training process are fully documented in Tab. A3 for comprehensive reproducibility.

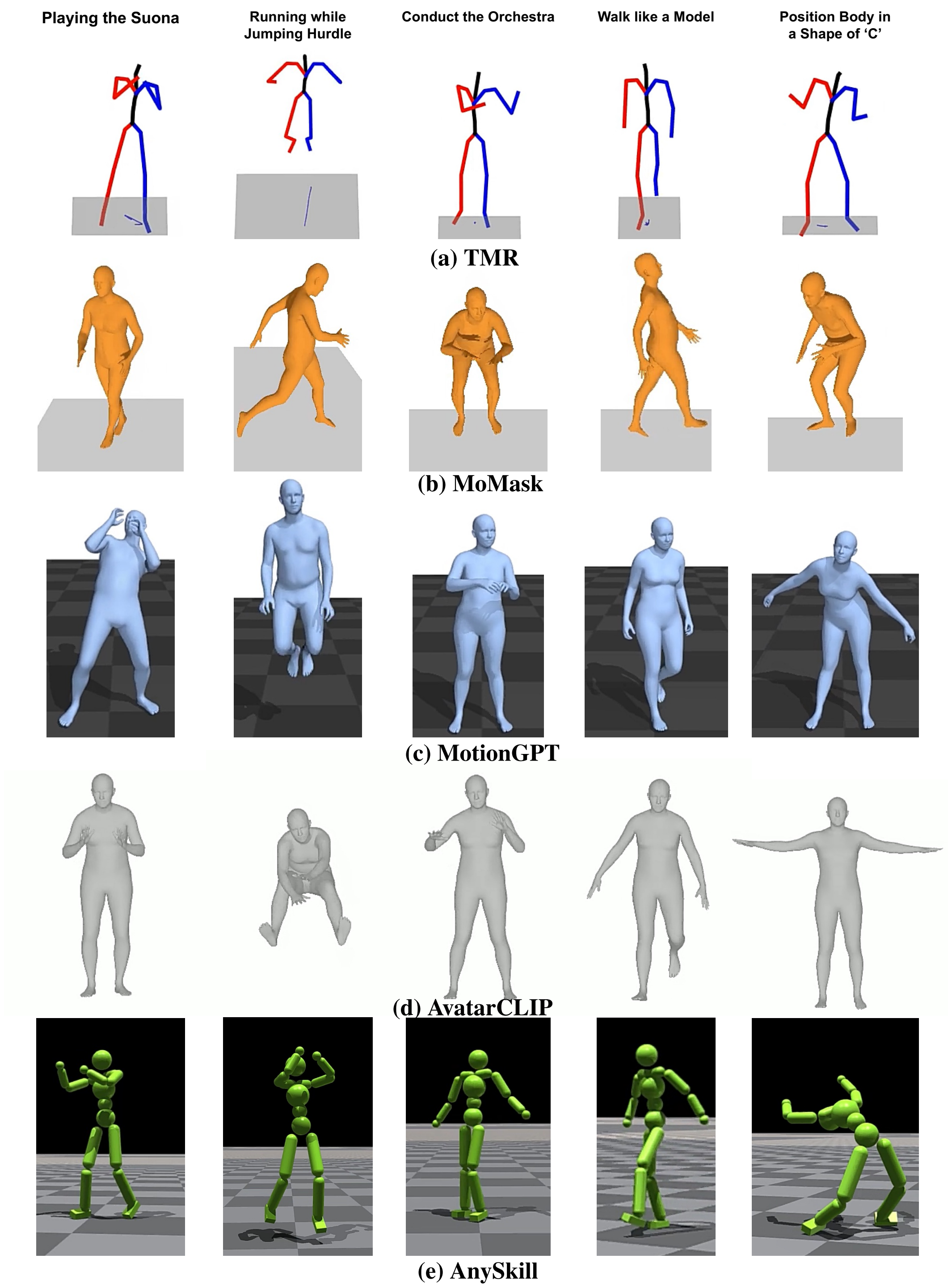
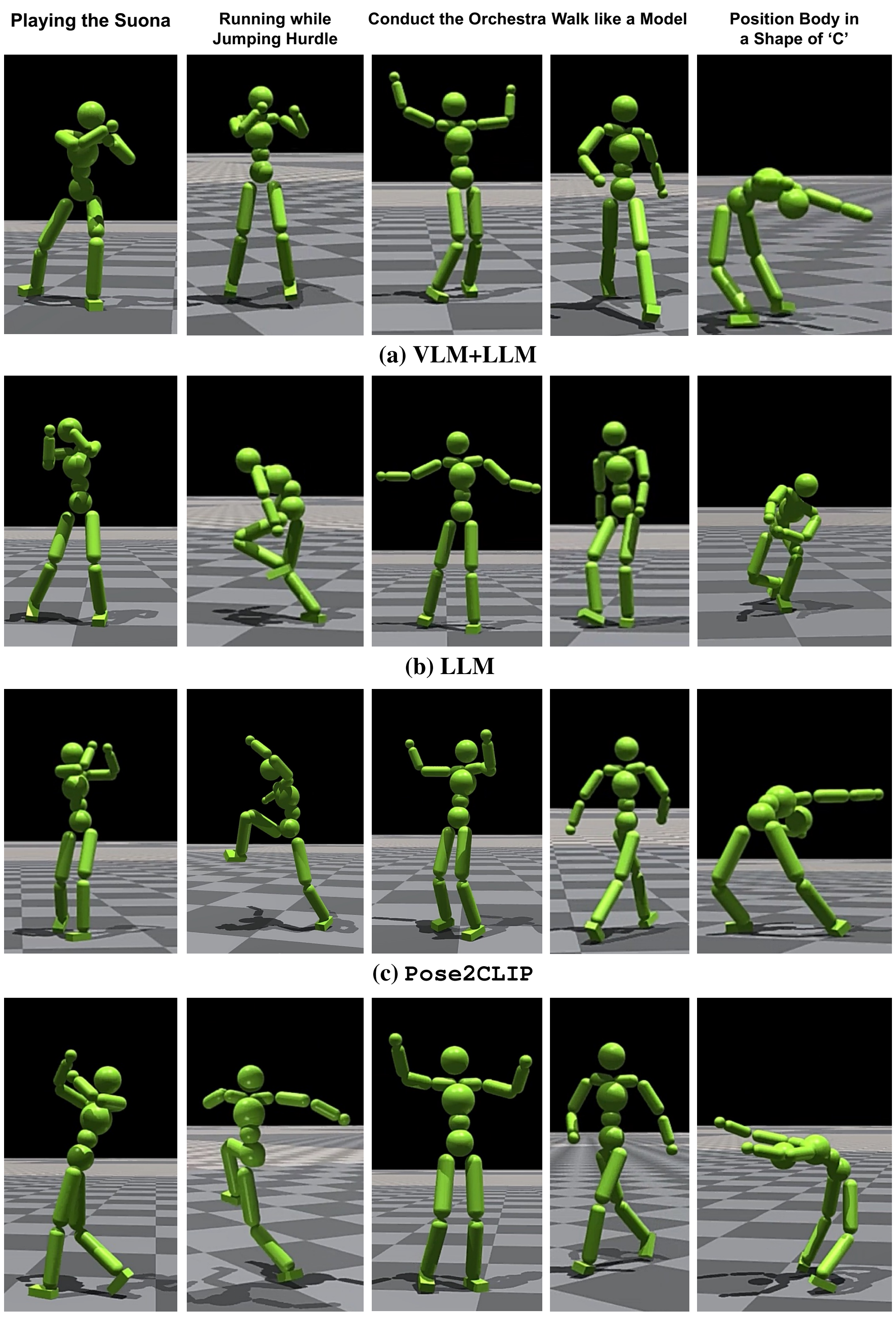


Figure A3. **Qualitative comparison of motion synthesis approaches.** We compare the performance of five baseline methods across five diverse text prompts. (a) Consistent with our quantitative findings in Tab. 1, TMR exhibits minimal capability to interpret the provided instructions. (b) While MoMask performs adequately on common actions like "jump," it struggles significantly with less familiar prompts. (c) MotionGPT generates motions that partially align with the instructions but predominantly defaults to upright postures with limited movement range, failing to capture the full expressiveness of the intended behaviors. (d) AvatarCLIP demonstrates potential but requires substantial improvement in motion quality. (e) AnySkill (VLM) consistently fails to execute the requested actions effectively. For comprehensive evaluation, we provide complete video demonstrations on our project website.



(d) `Pose2CLIP` + LLM

Figure A4. **Qualitative results of our motion synthesis approach.** We showcase diverse motion sequences generated by our `Pose2CLIP` + LLM model across five distinct text prompts, demonstrating the system's versatility and expressiveness. Each column showcases a different instruction: "Playing the Suona" (leftmost), where the agent adopts appropriate hand positions for the wind instrument with natural body movements; "Running while Jumping Hurdles," exhibiting biomechanically sound preparation, elevation, and landing phases; "Conduct the Orchestra," displaying expressive arm gestures with coordinated body positioning; "Walk like a Model," featuring the characteristic cross-stepping gait rather than simple forward motion; and "Position Body in a Shape of 'C"' (rightmost), showing precise body configuration control. Each row represents a key frame from the continuous motion sequence, highlighting our model's ability to generate temporally coherent, contextually appropriate movements that capture both the semantic meaning and physical nuances of the requested actions. These qualitative results align with our quantitative findings in Tab. 4, demonstrating superior motion range, coherence, and task-specific adaptations compared to alternative approaches. Complete motion sequences are available as videos on our project website.