

# Elucidating the Design Space of Torque-aware Vision-Language-Action Models

## A Appendix

In this appendix, we provide a comprehensive elaboration of the technical, experimental, and implementation details of our study. Sec. A.1 presents additional qualitative visualizations to supplement the main text, highlighting the system’s performance in various scenarios. Sec. A.2 meticulously derives the wrench-to-torque mapping for a 7-DOF manipulator, including the full spatial Jacobian, joint-specific partitions, and a quasi-static simplification for efficient torque computation. Sec. A.3 and A.4 detail the experimental protocols for the torque-integration and torque-history encoding strategies, ensuring reproducibility. Sec. A.5 outlines the implementation specifics of the joint action-torque diffusion objective, clarifying how torque prediction is achieved alongside action generation. Sec. A.6 provides further insights into the experimental setup, quantitative results, and cross-model as well as cross-embodiment evaluations, ensuring a robust understanding of our results. Sec. A.7 describes the architectural specifications of the baseline VLA models ( $\pi_0$  and RDT). Finally, Sec. A.8 evaluates the system’s efficiency in terms of training and inference, demonstrating that our torque-aware enhancements do not significantly impact computational performance.

### A.1 Additional Visualizations

We provide additional visualizations of the model’s execution process for the tasks described in the main text in Figures 9 and 10. For the contact-rich tasks, the torque response of the Shoulder Pitch, Shoulder Roll, and Wrist Yaw joints during execution is plotted in the last column, similar to Figure 1. We have marked the torque changes corresponding to failed and successful attempts for each task. See our website for complete [videos](#).

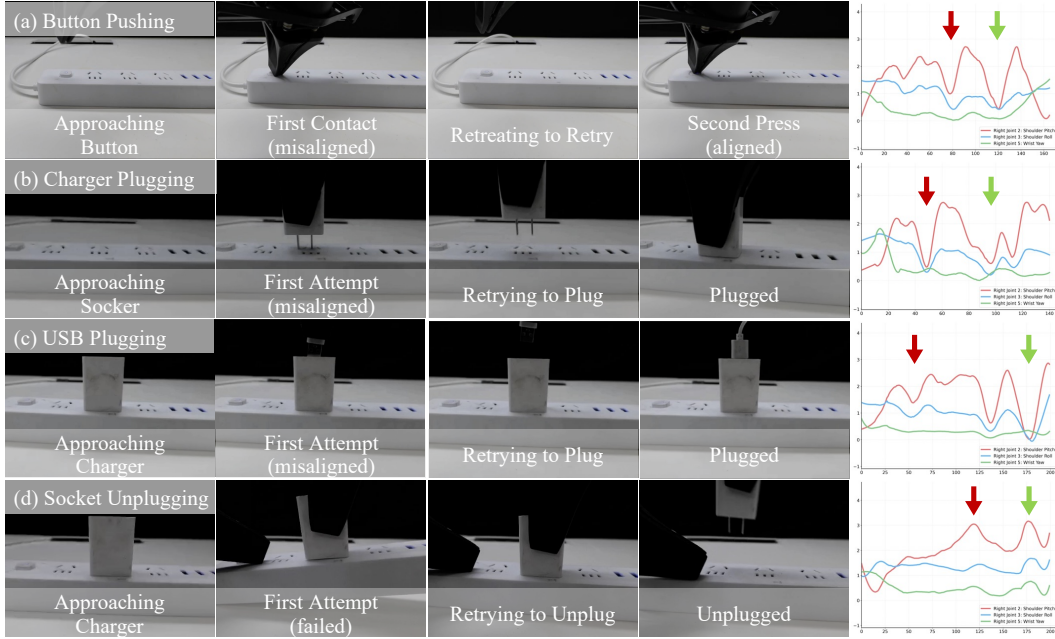


Figure 9: **Visualization of contact-rich tasks.** The last column visualizes the torque response of some joints during the task. For each task, the first failed attempt is marked with a red arrow, and the final successful attempt is marked with a green arrow.



Figure 10: **Visualization of general tasks.** (a) Bottle pick and place. (b) Liquid pouring. (c) Stacking cubes. (d) Push-to-position. (e) Opening a drawer.

## 446 A.2 Detailed Wrench-to-Torque Mapping for a 7-DOF Manipulator

447 Below we expand the derivation in Sec.3, make explicit the structure of the Jacobian, and specialize  
 448 it to a 7-DOF arm in which the first six joints are revolute actuators (shoulder→wrist) and the 7-th  
 449 joint is the gripper’s open/close degree of freedom.

### 450 A.2.1 Full Spatial Jacobian

451 For a serial arm with  $n$  joints, the geometric Jacobian  $J(\mathbf{q}) \in \mathbb{R}^{6 \times n}$  stacks the linear and angular  
 452 velocity components:

$$J(\mathbf{q}) = \begin{bmatrix} J_v(\mathbf{q}) \\ J_\omega(\mathbf{q}) \end{bmatrix}, \quad J_v, J_\omega \in \mathbb{R}^{3 \times n}. \quad (4)$$

453 For a revolute joint  $j$  with axis  $\hat{\mathbf{z}}_j$  and origin  $\mathbf{p}_j$  (in the base frame):

$$\begin{aligned} J_{v[:,j]} &= \hat{\mathbf{z}}_j \times (\mathbf{p}_e - \mathbf{p}_j), \\ J_{\omega[:,j]} &= \hat{\mathbf{z}}_j, \end{aligned} \quad (5)$$

454 where  $\mathbf{p}_e$  is the end-effector position.

### 455 A.2.2 Partition for the 7-DOF Arm

456 Let the joint order be:

$$[q_1, \dots, q_6, q_7] = \underbrace{[\text{Sh. Yaw, Sh. Pitch, Sh. Roll, Elb., Wr. Yaw, Wr. Pitch}]_{\text{arm (1-6)}}}_{\text{arm (1-6)}}, \underbrace{[\text{Gripper open/close}]_7}_{7}. \quad (6)$$

457 Then:

$$J(\mathbf{q}) = [J_{\text{arm}}(\mathbf{q}); J_{\text{grip}}(\mathbf{q})], \quad J_{\text{arm}} \in \mathbb{R}^{6 \times 6}, \quad J_{\text{grip}} \in \mathbb{R}^{6 \times 1}. \quad (7)$$

458 **Gripper column.** The gripper’s opening motion does not change the Cartesian pose of the  
 459 tool-centre-point (TCP), so:

$$J_{\text{grip}}(\mathbf{q}) = \mathbf{0}_{6 \times 1} \Rightarrow \tau_{\text{ext},7} = 0. \quad (8)$$

Consequently, forces at the TCP do not back-propagate torque to  $q_7$  :

$$\tau_{\text{ext},7} = J_{\text{grip}}^\top \mathbf{F}_{\text{ext}} = 0. \quad (9)$$

**Arm columns 1:6.** Each column is computed using the revolute formula above. For clarity we show the symbolic structure:

$$J_{\text{arm}} = \begin{bmatrix} \hat{z}_1 \times (\mathbf{p}_e - \mathbf{p}_1) & \dots & \hat{z}_6 \times (\mathbf{p}_e - \mathbf{p}_6) \\ \hat{z}_1 & \dots & \hat{z}_6 \end{bmatrix}. \quad (10)$$

### A.2.3 Wrench–Torque Projection

Given the external wrench  $\mathbf{F}_{\text{ext}} = [\mathbf{f}, \mathbf{m}]^\top \in \mathbb{R}^6$ :

$$\boldsymbol{\tau}_{\text{ext}} = J^\top(\mathbf{q}) \mathbf{F}_{\text{ext}} = \begin{bmatrix} J_{\text{arm}}^\top \mathbf{F}_{\text{ext}} \\ 0 \end{bmatrix} \in \mathbb{R}^7, \quad (11)$$

where  $\tau_{\text{ext},1:6} = J_{\text{arm}}^\top \mathbf{F}_{\text{ext}}$ , and  $\tau_{\text{ext},7} = 0$ .

### A.2.4 Quasi-static Simplification

Under low-velocity manipulation  $\dot{\mathbf{q}}, \ddot{\mathbf{q}} \approx 0$ , Eq. (1) simplifies to:

$$\boldsymbol{\tau}_{\text{measured}} \approx \mathbf{G}(\mathbf{q}) + \begin{bmatrix} J_{\text{arm}}^\top \mathbf{F}_{\text{ext}} \\ 0 \end{bmatrix}. \quad (12)$$

Subtracting  $\mathbf{G}(\mathbf{q})$  yields the external component  $\delta\boldsymbol{\tau}_{\text{ext}}$ , where only joints 1–6 are informative for contact detection. Large residuals in those joints directly indicate contact onset, direction, and magnitude, while small residuals in  $q_7$  confirm that gripper actuation alone does not contribute contact-induced torques.

### A.2.5 Practical Notes for Implementation

- **Gravity term  $\mathbf{G}$ :** Estimate via CAD model; recalibrate with payload when grasping heavy objects.
- **Torque from motor currents:** Use manufacturer-provided  $k_t$  with thermal compensation.
- **Contact detection:** Threshold  $\delta\tau$  values per joint to detect abnormal force events.

This detailed formulation clarifies the exact Jacobian structure, shows why the gripper joint is torque-insensitive to TCP forces, and provides concrete implementation guidance suitable for reproducibility in a top-tier robotics venue.

## A.3 Experimental Protocols for Sec. 4.1: Torque-Integration Architectures

**Experiment about Comparison between Architectures.** For the **Enc** and **DePost** architectures (Figure 2(a)(c)), we randomly initialize a MLP to project the effort token into the latent space. This MLP is structured with layers mapping from an input dimension of 14 (effort dim) to  $2 \times \text{width}$ , followed by a Swish activation, and then mapping from  $2 \times \text{width}$  to  $\text{width}$ . The  $\text{width}$  here corresponds to the model’s internal dimension: 2048 for the conditioning encoder in the **Enc** architecture and 1024 for the diffusion decoder in the **DePost** architecture. The state input of  $\pi_0$  is composed of the 14-dimensional joint positions, followed by 18 dimensions of zero-padding. For the **DePre** architecture (Figure 2(b)), we place the 14-dimensional joint efforts into the last 14 positions of this 32-dimensional state.

**Experiment about Better Input Alignment.** HSIC is a powerful nonparametric measure capable of detecting complex nonlinear relationships between variables without requiring assumptions about their distribution. Normalized HSIC provides a value between 0 and 1, where higher values indicate

stronger statistical dependence. To evaluate modality alignment, we analyze the  $\pi_0$  model trained using the **DePost** method on the Button Pushing task. We process input tokens obtained from frames randomly sampled from the training dataset of this task through the model’s transformer backbone (18 layers total). We then extract the intermediate representations, specifically the hidden states at the output of the 12th layer, corresponding to these input tokens. For the HSIC computation, the number of these extracted intermediate token representations used for each modality (action, angle, torque, image, and text instruction) is downsampled to 32. The normalized HSIC is then computed pairwise between each combination of modalities using RBF kernels, shown in Figure 3.

**Experiment about Sensitivity of Decoder.** To assess the sensitivity of the encoder and decoder to input variations, we add additive Gaussian noise with a standard deviation of 0.1 to the input tokens. For the encoder, noise is applied to all input tokens, whereas for the decoder, noise is applied specifically to the state token (the first token in the input sequence). The results are presented in Table 2.

#### A.4 Experimental Protocols for Sec. 4.2: Torque-History Encoding

**Experiment about Comparison between Architectures.** For historical effort input, we uniformly sampled 10 frames from the past 2 seconds, including the current frame. In the H Tokens configuration, each of these 14-dimensional effort tokens is processed independently using an MLP with the same architecture as described in Appendix A.3, which projects it into the latent space. Conversely, in the 1 Token configuration, the historical effort from all 10 frames is flattened and concatenated into a single 140-dimensional token before being fed into the MLP.

**Experiment about Better Input Pattern Completeness.** To investigate the completeness of the input pattern, we introduced an additional token into the input token sequence. This token was sampled from a standard normal distribution and had the same shape as other tokens in the sequence. We then modified the input and autoregressive masks accordingly, following the masking patterns applied to the effort tokens to integrate this new token into the sequence processing. Table 4 shows the results of the experiment.

#### A.5 Implementation of the Joint Action-Torque Diffusion Objective (Sec. 5)

To enable the model to simultaneously output future effort (torque) for supervision alongside action predictions, we expanded the dimension of the action input and output projection linear layer of the original model (Figure 5). When loading the pre-trained weights, we initialized the portion of the modified weight matrix corresponding to the original action output dimensions with the pre-trained values. The remaining weights, which correspond to the newly added dimensions for future effort prediction, were initialized using smaller values. This initialization strategy is designed to minimally affect the original pre-trained behavior initially, allowing the model to gradually learn the new prediction task during the finetuning process. The predicted future effort sequence has a length  $H = 50$  steps, matching the action chunk length. Figure 6 shows an example inference from the model trained on the Button Pushing task. The figure plots the model’s predicted future effort per frame against the corresponding ground truth for three selected axes, using an observation frame sampled from the task’s validation data as input.

#### A.6 Additional Details in Sec. 6

##### A.6.1 Details in Experimental Setup

In the  $\pi_0$  experiments, we followed the original setup using images from three viewpoints as input: top, left wrist, and right wrist. All  $\pi_0$  experiments were based on its publicly available pre-trained checkpoint and finetuned both encoder and decoder using LoRA. Training was performed for 30k gradient steps on  $4 \times$  NVIDIA L20 GPUs. The RDT experiments used the same GPU setup for full parameter training for 40k gradient steps. Inference was performed using an onboard RTX 4090 GPU. All variants of  $\pi_0$  used an inference action horizon of 50 steps, and RDT used 64 steps. Other

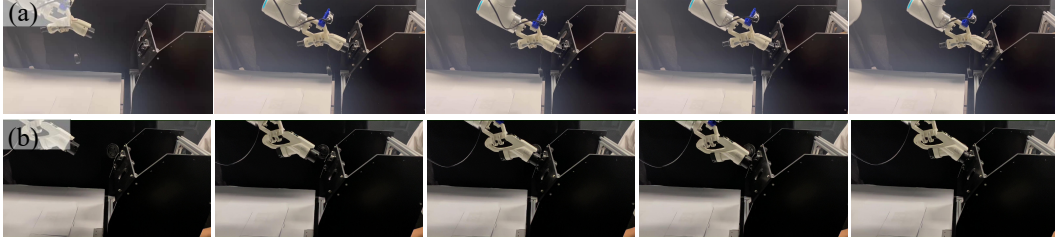


Figure 12: **Cross-Embodiment Task Execution: Charging Connector Insertion.** (a) The robotic manipulator successfully inserts a fast-charging connector into the charging port. (b) The robotic manipulator inserts a slow-charging connector into the charging port.

settings remained consistent with the original implementation. For all tasks, 400 demonstrations were collected using teleoperation.

#### A.6.2 Details about Quantitative Results

In the experiments (Table 5), for  $+obj$  settings, we set the value of  $\beta$  to 1. For  $+obs + obj$  settings, we set the value of  $\beta$  to 0.1. For the ACT and RDT models, their inference action horizons were 8 and 64 steps, respectively.

#### A.6.3 Details about Cross Model Results

Regarding the RDT+obs+obj model (Figure 11, Table 6), following the model’s approach for language and image adapters, we implement the effort projector as a two-layer MLP with a single GELU activation. This MLP maps the effort input into a 2048-dimensional vector, matching the width of RDT’s transformer backbone. The projected effort token is then concatenated after the state token. This combined sequence is further concatenated with the noisy action token, forming the input for the denoising process in RDT. We extended the state space input and output projectors and loaded pre-trained weights in a manner similar to that used in  $\pi_0$ .

#### A.6.4 Details about Cross Embodiment Results

To further assess the generalization capability of our torque-aware VLA model across different robotic embodiments, we conducted cross-embodiment experiments using the ROKAE SR robotic arm. Specifically, we trained the  $\pi_0+obs+obj$  with 200 demonstrations of the robot inserting a fast-charging connector into the charging port, using torque feedback to guide the insertion process. The model was trained for 50K gradient steps, and as shown in Figure 12(a), it successfully achieved the fast-charging insertion task with high reliability. To further evaluate generalization, we altered the task setup by replacing the fast-charging port with a slow-charging port. Without any architecture modification, the model was able to adapt seamlessly, successfully inserting the slow-charging connector into the port, as demonstrated in Figure 12(b). This result highlights the model’s capacity to generalize torque-aware manipulation strategies to new end-effector configurations, demonstrating robust cross-embodiment performance.

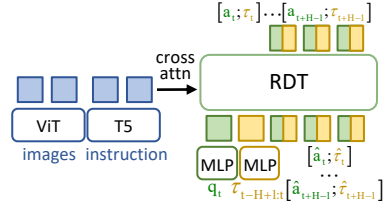


Figure 11: Architecture of RDT+obs+obj model.

#### A.7 Architectural Specifications of Baseline VLA Models ( $\pi_0$ and RDT)

$\pi_0$  is a VLA model built by adding a 300M action expert on top of the PaliGemma 3B pretrained VLM backbone and trained on a large-scale cross-embodiment robot dataset. The action expert and the VLM are two independent sets of weights within a single transformer, interacting only through the self-attention layers. During each inference step, the model receives three RGB images, a language instruction, and the robot’s proprioceptive state. The images and language instruction



are fed directly into the VLM as conditions. Proprioception is concatenated with a noisy action chunk of length 50 and input to the action expert.  $\pi_0$  uses Conditional Flow Matching to model the continuous action distribution. The action expert outputs a vector field, and the final action chunk is generated by performing 10 integration steps on this vector field.

RDT (Robotics Diffusion Transformer) is a 1B-parameter Diffusion Transformers (DiTs) model pre-trained using 1M multi-robot trajectories (including 46 datasets). It utilizes a physically interpretable 128-dimensional unified observation and action space, encoding low-dimensional inputs using MLPs with Fourier features. During each inference step, the model’s inputs include proprioception, a noisy action chunk (size 64), and the diffusion timestep. Language instruction and observations (including 2 history frames of three images, proprioceptive state, and control frequency) are input to the model as conditions. Images are processed by SigLIP, while language is processed by T5-XXL, this conditional information is alternately injected into different layers of the DiTs using cross attention. The model predicts the denoised action chunk, and the final action is produced through 5 denoising steps.

## A.8 System Efficiency

We use the Button Pushing task as an example to compare the training and inference times for  $\pi_0$ ,  $\pi_0 + \text{obs}$ ,  $\pi_0 + \text{obj}$ , and  $\pi_0 + \text{obs} + \text{obj}$ . The training speed is reported as the average throughout the training process measured on  $4 \times \text{NVIDIA L20 GPUs}$ , and the inference speed is averaged over 10 runs on an RTX 4090 GPU. The results are shown in Table 7 and Table 8, which indicate that the proposed designs do not significantly affect efficiency.

Model Variant	Training Time (s/iter)
$\pi_0$	1.703
$\pi_0 + \text{obs}$	1.628
$\pi_0 + \text{obj}$	1.648
$\pi_0 + \text{obs} + \text{obj}$	1.640

Table 7: Training Time for Different Designs

Model Variant	Inference Time (ms)
$\pi_0$	90.70
$\pi_0 + \text{obs}$	90.81
$\pi_0 + \text{obj}$	90.61
$\pi_0 + \text{obs} + \text{obj}$	93.96

Table 8: Inference Time for Different Designs