APPENDIX A
IMPLEMENTATION DETAILS

A. Network Architecture

Point Cloud and Robot State Encoding: The raw 3D point cloud converted from RGB-D observations is first downsampled to 1024 points using Farthest Point Sampling (FPS), and then encoded into a global visual representation via a lightweight MLP encoder. In parallel, the robot proprioceptive state, represented as current 3D robot node positions obtained through forward kinematics, is also encoded with an MLP.

Conditional Denoising: At each diffusion step k, the noisy action represented as robot node trajectories A^k is processed by a temporal Conv1D U-Net. Global conditioning information (visual features, robot state features, and sinusoidal diffusion timestep embeddings) is injected through cross-attention layers. We adopt sample prediction (rather than ϵ-prediction) to directly estimate the clean sample $\hat{A}^0$ and compute the denoised sample A^{k-1}. This procedure is iterated for K diffusion steps.

Hyper-parameter Details: Each action, robot state and observation dimension is independently normalized to the range $[-1, 1]$. Both visual and robot-state embeddings are set to 64 dimensions. During training, we use 100 diffusion steps, while at inference time we employ 10 steps with DDIM as the noise scheduler for faster sampling. Other key hyper-parameters include the observation horizon $T_o = 2$, action horizon $T_a = 8$, and execution horizon $T_e = 4$.

B. Baseline Settings

DP3-ERJ: Mazzaglia et al. [6] proposed two realizations of the ER (End-effector Redundancy) action space family, ERAngle (ERA) and ERJoint (ERJ), to enable full-body control of redundant robot arms while preserving space alignment. ERA resolves redundancy by introducing an additional elbow angle, whereas ERJ controls the redundant joint separately from the inverse kinematics. As ERJ shows superior performance in [6], we implement ERJ within the DP3 framework, treating the base joint as an independently controlled joint.

DP3-HDP: For a fair comparison, we reproduce the HDP approach by removing the high-level Perceiver-Actor [23] agent and retaining only the low-level Robot Kinematics Diffuser. Specifically, two separate diffusion branches, are employed to predict joint trajectories and end-effector pose trajectories, respectively. Differentiable forward kinematics $\mathrm{F}_{\mathrm{fk}}(\cdot)$ are used to link two diffusion branches. The total training objective is:

$$\begin{aligned}\mathcal{L} = \lambda_{\mathrm{pose}}\,\mathrm{MSE}(A_{\mathrm{pose}}, A^{\mathrm{gt}}_{\mathrm{pose}}) + \lambda_{\mathrm{joint}}\,\mathrm{MSE}(A_{\mathrm{joint}}, A^{\mathrm{gt}}_{\mathrm{joint}}) \\ +\lambda_{\mathrm{joint}\to\mathrm{pose}}\,\mathrm{MSE}\big(\mathrm{F}_{\mathrm{fk}}(\mathrm{A}_{\mathrm{joint}}), \mathrm{A}^{\mathrm{gt}}_{\mathrm{pose}}\big),\end{aligned} \tag{11}$$

where the weighting parameters are set to $\lambda_{\mathrm{pose}} = \lambda_{\mathrm{joint}} = \lambda_{\mathrm{joint}\to\mathrm{pose}} = 1$, following the official HDP implementation.

During inference, the predicted joint trajectory is refined to ensure that the corresponding end-effector poses closely match the predicted pose trajectory. A single optimization step with learning rate α is denoted as:

$$A_{\mathrm{joint}} \leftarrow A_{\mathrm{joint}} - \alpha \frac{\partial \left\| A_{\mathrm{pose}} - F_{\mathrm{fk}}(A_{\mathrm{joint}}) \right\|}{\partial\, A_{\mathrm{joint}}}. \tag{12}$$

APPENDIX B
ADDITIONAL RESULTS AND ANALYSIS

A. Details on Inverse Kinematics Mapping

Optimization-Based IK: We implement the optimization-based IK solver using SciPy's *minimize* function with the SLSQP algorithm. The gradient of the objective respect to the joint angles is computed analytically using the chain rule, by multiplying the current node position error with the Jacobian matrix ($\partial A_{\mathrm{node}}/\partial A_{\mathrm{joint}}$), which is obtained through widely-used robot kinematics toolkit Pinocchio. In practice, the solver is initialized with the joint configuration from the previous control step. Joint limits are enforced via box constraints throughout optimization. For the 7-DoF Franka Emika Panda robot, we optimize a total of 8 joints (7 arm joints plus one gripper joint) when solving IK; however, only the 7 arm joints are used to control the robot, while the gripper's binary opening-closing action is enabled through a separate action channel output by the diffusion model for improved precision. This design allows the IK solver to focus on accurate whole-arm configuration while maintaining reliable gripper actuation.

MLP-Based IK: The simple MLP network, which takes 3D node positions as input and predicts the corresponding joint angles, consists of three fully connected linear layers with nonlinear activations in between. During training, joint angles are randomly sampled within the predefined joint limits and transformed into 3D node positions via forward kinematics to form supervised training pairs. Using an $\mathcal{L}_2$ loss function between predicted and ground-truth joint configurations, the network is trained with the Adam optimizer for 3,000 epochs, where each epoch contains 10,000 randomly sampled joint configurations with a batch size of 128.

IK Error: The mean errors of the optimization-based and MLP-based IK solvers are reported in Table III. Here, *Joint Space* error denotes per-joint distance between the predicted joint angles and the ground-truth configuration, while *Node Space* error measures the per-node Euclidean distance between the node positions computed from the predicted joints and those obtained from the ground-truth joints via forward kinematics. For the optimization-based IK solver, Gaussian noise is added to the ground-truth joint angles to serve as the initial guess, reflecting realistic usage during inference.

As expected, the MLP-based IK exhibits larger errors than the optimization-based solver. However, since IK mapping is only invoked when perturbing the original sample A_0 with Gaussian noise, the relatively small approximation error of MLP-based IK can therefore be viewed as an additional stochastic perturbation, which does not alter the fundamental properties of the forward diffusion process. The training objective of the single-step denoising network remains independent of the MLP-based IK. In contrast, during inference, the policy starts from a sampled Gaussian noise and iteratively performs multi-step denoising to generate final actions. If the MLP-based IK is used, its approximation error in each step would accumulate across denoising iterations, degrading execution accuracy. Thus, the optimization-based IK is adopted during inference to ensure more precise node-to-joint mapping.

TABLE III: Comparison of the mean per-joint error and per-node Euclidean error of the optimization-based and MLP-based IK.

Error / Method	Optim-based IK	MLP-based IK
Joint-Space Error (rad)	0.00352	0.03408
Node-Space Error (m)	0.00021	0.01624

B. Computation Time Cost

We conduct a comprehensive analysis of the computational overhead introduced by incorporating FK and IK into the policy, using a single NVIDIA RTX 3090 GPU. We report the time cost of key components for both the full KADP model and KADP without kinematic constraints (*w/o KC*): observation encoding (Obs. Enc.), denoising network forward pass (Denoi. Net), forward kinematics (FK), inverse kinematics (IK MLP / IK OPT), and other operations (Others). During training, we additionally measure the loss backpropagation time.

As shown in Fig. 9, feature extraction and other operations incur negligible overhead in both training and inference. Across different batch sizes during training, the MLP-based IK adds minimal cost and FK introduces only modest overhead. Although kinematic constraints slightly increase loss backward computation time, the overall training efficiency remains largely unaffected. During inference, latency is evaluated under different numbers of DDIM steps. The denoising network accounts for more than half of the total runtime and remains nearly unchanged with or without kinematic constraints. FK and IK together contribute about 45% of the total latency, with optimization-based IK being slightly more time-consuming. We note that the current implementations of FK and the optimization-based IK have not been extensively optimized, and there remains significant room for improving their computational efficiency.

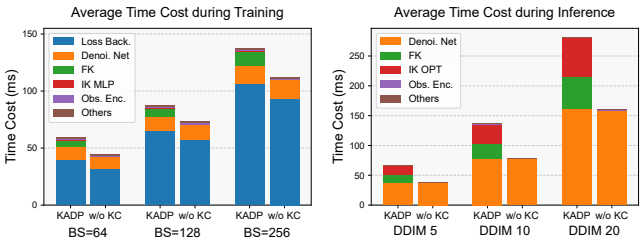
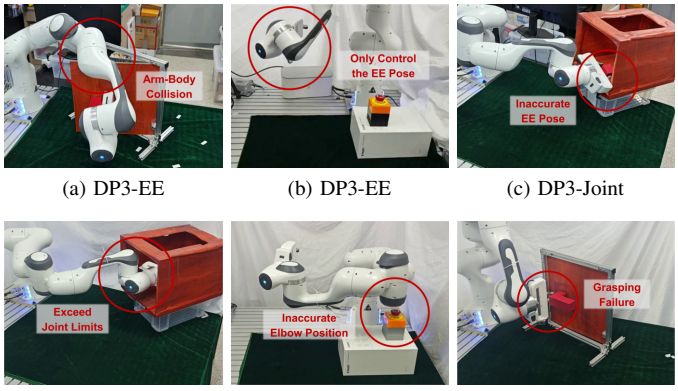


Fig. 9: Breakdown of average computation time during training (left) and inference (right) for KADP with and without kinematic constraints, illustrating the overhead of key components under varying batch sizes and DDIM steps.

C. Real-world Failure Cases

We further illustrate additional representative failure cases of the baseline methods in Fig. 10 to highlight their limitations in comparison with KADP. DP3-EE considers only the end-effector pose and lacks the ability to coordinate full-configuration motion, which often leads to collisions between the arm body and the manipulated object (Fig. 10a) and results in consistent failure on whole-arm manipulation tasks such as `push button elbow` (Fig. 10b). In contrast, DP3-Joint has access to individual joints and enables full-configuration control; however, its typical failures mainly stem from inaccurate end-effector pose during manipulation. For example, on the `put cube in cabinet` task, DP3-Joint frequently causes the gripper to collide with the cabinet side wall (Fig. 10c), highlighting the difficulty of learning the non-linear task-to-joint mapping effectively with limited data. DP3-ERJ and DP3-HDP, which both enables whole-arm control while maintaining partial space alignment to some extent, markedly outperform DP3-Joint on tasks requiring precise end-effector prediction. However, their optimization-based post-processing module, jointly leveraging predicted joint angles and end-effector poses, does not always guarantee accurate final execution. DP3-ERJ occasionally violates joint limits during optimization (Fig. 10d), primarily due to inaccurate redundant joint predictions. DP3-HDP performs slightly better but still suffers from end-effector inaccuracies after the refinement module (Fig. 10f), due to only partial task–observation–action space alignment compared to our approach. Overall, KADP succeeds in all evaluation cases presented here. Please refer to our supplementary video (also available on https://kinematics-aware-diffusion-policy.github.io) for further visualizations of success and failure cases.



(a) DP3-EE (b) DP3-EE (c) DP3-Joint

(d) DP3-ERJ (e) DP3-ERJ (f) DP3-HDP

Fig. 10: Additional failure cases of baseline methods on real-world manipulation tasks. Please refer to the supplementary video for complete execution visualizations.