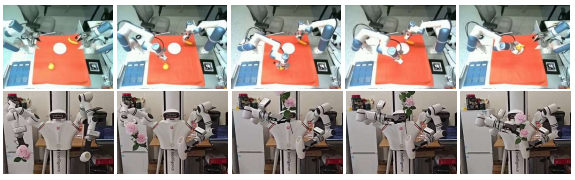
A_0: An Affordance-Aware Hierarchical Model for General Robotic Manipulation

To All: We sincerely thank the reviewers for their appreciation: clear motivation and structure (R1-99Qp, R2-V5ED, R3-WKEr), strong contribution (R2, R3), cross-robot validation (R3), effective hierarchical design (R1, R2), real-world applicability (R2), and large-scale dataset with thorough experiments (R3). All comments on grammar, notations and diagrams will be adopted to further improve the readability.

R1 *Q1* and R3 *Q3* Orientation-sensitive tasks. Our method focuses on high-level spatial understanding. The action execution module is fully modular and can be replaced. For tasks requiring precise orientation and nuanced 6D manipulation—such as liquid pouring and revolute-drawer opening—We will choose an optimal observation viewpoint that allows A_0 to predict the necessary waypoints. As demonstrated in prior work (e.g., ATM: Anypoint trajectory modeling), a track-guided policy can be trained with only a few demonstrations to achieve accurate control. For more complex tasks, VLMs like GPT-4o can be employed to decompose the task into a sequence of subtasks and A_0 can address each step individually.

R1 *Q2* and R3 *Q2* Long-horizon planning. Our method faces this common limitation shared by affordance-based and modular approaches. Even current VLA models struggle with long-horizon tasks. In future work, we will address this by leveraging VLMs such as GPT-4o to decompose long-horizon tasks into a sequence of shorter subtasks (e.g., $\pi_{0.5}$), which can then be executed stage-by-stage using A0. We will include a discussion of this limitation in the revised manuscript. While MOKA and ReKep handle multiple objects via prompt VLM, we have incorporated VLM for high-level planning in our new experiments. This enables our model to perform more complex tasks, such as inserting flowers into the bottle and putting fruits on plate, as shown in the demo. We will include this in the revised version.



R1 *Q3* Performance gap on real robots. First, when deploying our method in real-world environments, the testing setups for the two robotic arms (kinova and franka) were not exactly identical. differences in lighting conditions, object placements, and scene configurations may lead to slight variations in the model's predictions. Second, although we tried our best to ensure accurate calibration between each robot and its corresponding camera, such calibration is not guaranteed to be perfect. this imperfection can cause deviations in the predicted trajectories based on camera observations. This issue is especially critical in the button-pressing task, where success is defined by the arm accurately pressing a small red button. in some cases, the kinova robot may end up just a few centimeters away from the button, which is still counted as a failure according to our strict success criterion.

R1 *Q4* More comparisons with VLA methods. We also conducted comparisons with VLA methods on other tasks with the same settings as in the paper, and we will include these results in the revised version. The results are shown in the following table.

	RDT	π_0	π_0 + FAST	A_0
Place Object	20	40	35	**60**
Open Drawer	0	20	10	**65**
Press Button	25	10	30	**40**
Wipe Board	0	10	0	**50**
Avg. Success	11.25	20.00	18.75	**53.75**

R2 *Q1* and R3 *Q1* Execution speed. The "execution steps" refer to the full rollout process of task execution on the robot. In VLA methods, the robot performs many fine-grained actions, requiring a model inference at each step—typically 25–50 steps. In contrast, our method predicts 4–5 waypoints at the beginning of the task, reducing the rollout to only 4–5 steps. We will clarify this in the paper and include a comparison table for better transparency.

R2 *Q2* Training details. We utilized four A100 80GB GPUs with a batch size of 200. The 1B model required 73 GB of memory per card. For pre-training, the model was trained for 80,000 steps over 5 days. For post-training, it was trained for 30,000 steps, taking 50 hours. We will release both the code and the pretrained model weights.

R2 *Q3* Typos and readability issues. We will fix the typos, enlarge the text in Figures 5 and 7, and improve all unclear parts for better readability.

R3 *Q1* "Object-centric" definition. MOKA uses a scene-level approach by prompting the VLM to mark different objects in the entire scene. Similarly, ReKep also operates at the scene level, considering the entire scene for constraint optimization. In contrast, our method focuses on object-centric reasoning, which is different from these scene-level approaches. We will clarify the definition of "object-centric" in the paper.

R3 *Q4* Rollout time. MOKA requires grounding with SAM and two prompts to GPT-4v, while ReKep uses DINOv2, SAM, and GPT-4o. In contrast, our method only requires one A0 and one GPT-4v prompt. The rollout time comparison is shown below.

Method	MOKA	ReKep	A_0
Rollout Time	144.75 s	40.33 s	29.79 s