Dexbotic: Open-Source Vision-Language-Action Toolbox

Dexbotic Team

Project: https://dexbotic.com **Github**: https://github.com/Dexmal/dexbotic

Huggingface: Dexbotic Collection

Abstract

In this paper, we present Dexbotic, an open-source Vision-Language-Action (VLA) model toolbox based on PyTorch. It aims to provide a one-stop VLA research service for professionals in the field of embodied intelligence. It offers a codebase that supports multiple mainstream VLA policies simultaneously, allowing users to reproduce various VLA methods with just a single environment setup. The toolbox is experiment-centric, where the users can quickly develop new VLA experiments by simply modifying the Exp script. Moreover, we provide much stronger pretrained models to achieve great performance improvements for state-of-the-art VLA policies. Dexbotic will continuously update to include more of the latest pre-trained foundation models and cutting-edge VLA models in the industry.

1. Introduction

Recently, significant progress has been made in the field of embodied intelligence with the development of Vision-Language-Action (VLA) models [2, 4, 15, 26, 27, 30]. However, research in this area is fragmented across various institutions, each using different deep learning frameworks and model architectures. This diversity creates challenges for users when comparing policies, as they must configure multiple experimental environments and data formats, making the VLA development process cumbersome. Additionally, ensuring that each policy being compared is optimized to its fullest potential is difficult, leading to unfair comparisons. Furthermore, VLA models have evolved alongside Vision-Language Models (VLMs). However, many existing VLA models [14, 16] are built on outdated VLMs [29], making most users fail to benefit from the latest advanced VLMs.

To address these challenges, we release the so-called Dexbotic Toolbox for the Embodied AI community to push VLA research forward. Reviewing the toolbox development in AI 1.0 Era [7, 32], the first step is to unify the model architecture. This process involves standardizing the structures and designs of models, which facilitated easier sharing, comparison, and improvement of algorithms across the research community. For example, mmdetection [7] disassembles the object detectors into backbone, neck and head. Similarly, existing VLA policies are uniformly divided into two parts in Dexbotic: vision-language model (VLM) and action expert (AE). The VLM part mainly includes a vision encoder, projector and large language model (LLM). It takes the observation and task prompt as inputs and produces the multi-modal tokens, which can be used to generate the discrete actions [4, 15] or serves as the input of AE. The architecture of AE can be diffusion transformer [16], Multi-layer Perception (MLP) [14] or Mixture-of-Experts (MoE) [2, 3] coupled with LLM.

Based on the unified model architecture above, Dexbotic further provides stronger pretrained models for some mainstream VLA policies, compared to the original open-source ones. Many existing VLA policies are built upon some outdated open-source VLMs or LLMs. For example, OpenVLA [15] and its follow-ups like CogACT [16] and OFT [14], are all constructed based on Llama2 [29], whose representation is much inferior than those latest LLMs, like Qwen3 [33]. In Dexbotic, we introduce the DexboticVLMs, which integrates the open-source vision encoders with latest LLMs. Based on DexboticVLMs, we first provide discrete pretrained model for general VLM-based discrete policies. Users can utilize the discrete pretrained model for initialization of VLM part. For specific continuous-representation VLA policies, we further provide the detailed implementation for various action experts. Both the single-arm and hybrid-arm continuous pretrained models are provided to initialize the whole models.

To facilitate the development of VLA experiments, Dexbotic introduces an experiment-centric development framework. Different from those codebases that build upon the *yaml* files for configuration [32], Dexbotic configurates the parameters by the *Exp* scripts based on the provided *base_exp* script. User can simply modify these parameters that differentiates from the *base_exp* script without affecting the whole configuration. Users can easily meet various needs such as modifying configurations, changing models, or adding tasks by simply altering the *Exp* script. More-

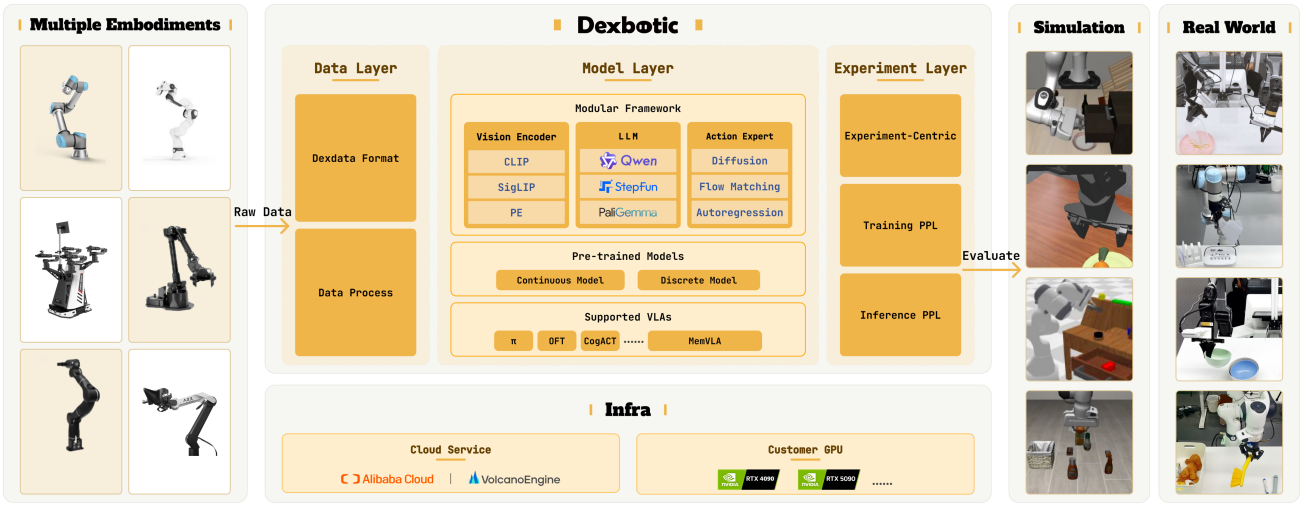


Figure 1. The overall architecture of Dexbotic. It introduces the Dexdata format to unify different embodiments. In Model Layer, Dexbotic integrates the open-source vision encoder, LLM and action expert through a unified modular VLA framework. Based on the provided DexboticVLMs, users can develop existing VLA policies and custom policies. Based on the developed policies, we further propose the Experiment Layer for fast development. Both the training pipeline and inference service are supported on some cloud service and customer GPUs.

over, Dexbotic supports the VLA training and inference on some cloud service like Alibaba Cloud as well as customer GPUs to satisfy the demand of different users. Dexbotic introduces the Dexdata format to support the training and deployment on multiple robots. The Dexdata format can save the storage for model training, compared to the LeRobot [6] and RLDS [25] format.

2. Main Features

2.1. Unified Modular VLA Framework

Dexbotic centers around VLA models and is compatible with open-source interfaces of mainstream large language models (LLM). It integrates embodied manipulation and navigation, supporting multiple leading embodied manipulation and navigation policies, while also incorporating interfaces for future whole-body control.

2.2. Powerful Pre-trained Foundation Models

For mainstream VLA policies such as π_0 [2] and CogACT [16], Dexbotic open-sources several more powerful pre-trained foundation models. These models bring significant performance improvements across various mainstream simulators, like SimplerEnv [17] and CALVIN [21], as well as the real-world robotic tasks.

2.3. Experiment-Centric Development Framework

The experimental framework of Dexbotic adopts a "layered configuration + factory registration + entry dispatch" approach. Users can easily meet various needs such as modifying configurations, changing models, or adding tasks by simply altering the experimental Exp script. This design aligns with the Open-Closed Principle, allowing for flexibility and extensibility while maintaining stability.

2.4. Cloud and Local Training Capabilities

Dexbotic fully addresses the training needs of users from different universities and enterprises. It supports large-scale cloud-based training platforms such as Alibaba Cloud and Volcano Engine. Additionally, it accommodates local training with consumer-grade GPUs, such as RTX 4090 cards.

2.5. Diverse Robot Training and Deployment

For various mainstream robots, such as UR5, Franka and ALOHA, Dexbotic offers a unified data format for training. It also provides open-source, general-purpose deployment scripts, allowing users to customize their deployments. In the future, Dexbotic will continue to support additional mainstream robotic platforms.

3. Supported VLA Policies

Unified Policy Representation: The representation of different policies can be unified for both robotic manipulation and navigation, though their components may have some differences. Usually, the VLA policies can be simply divided into two parts: *VLM* and *Action Expert*.

VLM can be regarded as the backbone of VLA policy. It is usually pretrained on multimodality data, such as the image-text pairs for VQA and image caption. It mainly includes three parts: Vision encoder (e.g., CLIP [24], SigLIP [34]) for visual token generation. Projector (e.g.,

two-layer MLPs) projects visual tokens into the textual space. LLM (e.g., Llama 2 [29]) takes as input visual and textual tokens and produces tokens for text generation.

Action Expert takes the representation from VLM, such as the multi-modal tokens or cognition token, as input and produces the action chunking. For example, CogACT employs Diffusion Transformer while π_0 uses the flow matching. Currently, Dexbotic supports these VLA policies for robotic manipulation and navigation as follows. More VLA like $\boldsymbol{\pi_{0.5}}$ [3] and VLN policies like NaVid [35] and NaVILA [9] will be supported in the near future.

• $\boldsymbol{\pi_0}$ [2] is a well-known flow matching VLA policy and built upon the PaliGemma [1] and uses the action expert to generate action chunking.

• **OpenVLA-OFT** [14] can be regarded as the improved version of OpenVLA [15] and explores fine-tuning strategies to greatly improve manipulation performance.

• **CogACT** [16] is also based on the OpenVLA and extracts the cognition token, together which the noises are input to the Diffusion Transformer for diffusion modeling.

• **MemoryVLA** [26] further introduces the concept of perceptual-cognitive memory, improving the performance on long-horizon tasks.

• **MUVLA** [11] is a recently proposed VLA policy based on map understanding for object navigation and achieves state-of-the-art performance.

4. Architecture

4.1. Overall Architecture

Fig. 2 shows the overall architecture of Dexbotic toolbox. It mainly includes three typical layers: Data Layer, Model Layer and Experiment Layer. In Data Layer (Sec. 4.2), we define the so-called Dexdata format to unify different data sources and save the storage. With Dexdata format data, dexbotic performs the data process to extract the image, text and state for training. In Model Layer (Sec. 4.3), we introduce the basic DexboticVLM as the foundation model to develop more VLA policies. DexboticVLM can be directly used for discrete VLA training, like RT-2 [4] and OpenVLA [15]. It can also serve as the base model of existing VLA policies. For current version, we support multiple VLA policies like π_0, OpenVLA-OFT and CogACT. User can directly define their own custom VLA models. The most important part of dexbotic is the Experiment Layer (Sec. 4.4). Based on different VLA models in Model Layer, we introduce various experiment scripts to support fast development for existing and custom

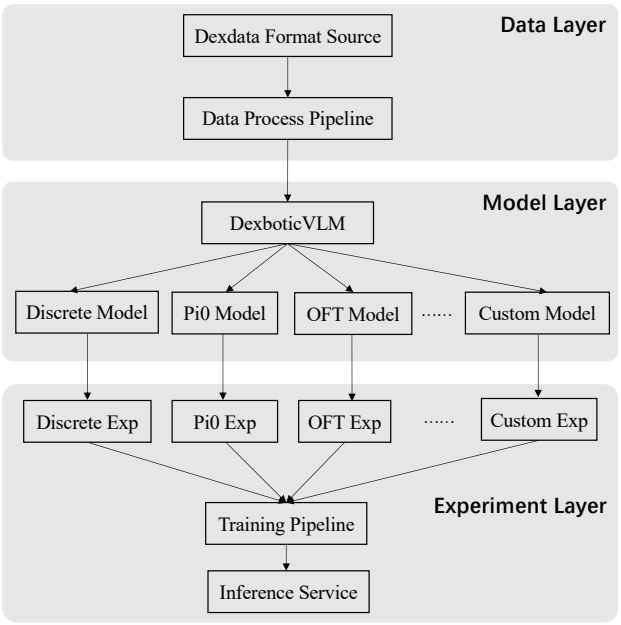


Figure 2. The overall architecture of Dexbotic. The framework is organized into three core layers including the data, model and experiment layers, that work together to provide a complete solution for training and serving VLA models.

policies while maintaining the stability of whole toolbox. With the experiment scripts, users can perform the training pipeline and the inference service using different modes.

4.2. Data Layer

Dexdata Format: We designed the Dexdata format to store robotic datasets in a unified and efficient way. As shown in Fig. 3 (a), it mainly includes two elements: *video* and *jsonl*. The *video* directory contains video files with mp4 format while *jsonl* directory includes the corresponding jsonl files. Each jsonl file contains the data for a single robot episode. In *jsonl* directory, the *index_cache.json* file, which users can ignore, stores the metadata for all episodes and is automatically generated for fast access. Fig. 3 (b) illustrates one example of one line in a given jsonl file. It provides the detailed information of a frame about the multi-view images, robot state and the textual prompt. For the detail of data process pipeline, please kindly refer to the Sec. 4.4.

4.3. Model Layer

In this section, we first introduce our pretrained VLM, called DexboticVLM. Based on such a foundation model, we further describe how to develop existing and custom VLA policies. After that, we introduce some robotic pretrained models for finetuning both manipulation and navigation tasks.

(a) Overview of Dexdata dataset

```
video
    episode1.mp4          # the first episode
    episode2.mp4          # the second episode
    ...
jsonl
    index_cache.json      # Global index of dataset_1
    episode1.jsonl        # Data for the first episode
    episode2.jsonl        # Data for the second episode
    ...
```

(b) One example of one line in json file

```
{
  "images_1": {"type": "video", "url": "url1", "frame_idx": 21},
  "images_2": {"type": "video", "url": "url2", "frame_idx": 21},
  "images_3": {"type": "video", "url": "url3", "frame_idx": 21},
  "state": [0.1, 0.2],
  "prompt": "open the door",
  "is_robot": true
}
```

Figure 3. The overview of Dexdata format.

4.3.1 DexboticVLM

For better generalization and the best use of existing open-sourced state-of-the-art LLM, we pretrain VLM from scratch. In Dexbotic, we choose to pretrain our own VLMs, called DexboticVLM. We utilize the CLIP [24] as the vision encoder, two-layer MLP as the projector and Qwen2.5 as the LLM. Similar to the LLaVA training pipeline [20], we first freeze the vision encoder and LLM parts and only train the projector for cross-modal alignment. After that, the parameters of whole network, including the vision encoder, projector and LLM, are updated. The training data sets include those of LLaVA and Cambrian [28].

4.3.2 Model Development

As described in Sec. 3, existing VLA pipeline can be unified with VLM and action expert parts. The DexboticVLM mentioned above can be directly used for discrete VLA training, the same as [4, 15]. To enable DexboticVLM to directly predict robot actions, the actions in LLM output space are represented by mapping continuous robot actions to discrete tokens used by the language model tokenizer. Each dimension of the robot actions is discretized separately into 256 bins. To reproduce the existing continuous representation VLA models, we can build different action experts to produce various policies. For example, we can add the Diffusion Transformer as the action expert to construct the CogACT [16] model. We can further introduce the memory module between DexboticVLM backbone and action expert to produce MemoryVLA [26] policy. For current version, we support multiple VLA policies like π_0 [2], OpenVLA-OFT [14] and CogACT. On the other hand, they can also develop their customized VLA policies by designing new action expert or introducing a supervision pipeline.

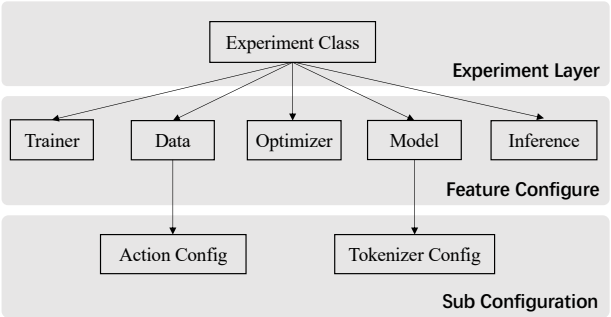


Figure 4. The layered configuration architecture in experiment layer. Each experiment class includes the configurations on trainer, data, optimizer, model and inference.

4.3.3 Pretrained Models

For different demands on various robotic arm of users, we provide two kinds of pretrained models. The first one is the pretrained discrete model for general VLA policies and the second one is the pretrained continuous model for specific VLA policies. For continuous pretrained model, we further provide two versions for single-arm and two-arm tasks. We take CogACT as example to clearly illustrate the continuous pretrained models.

Discrete Pretrained Model: Based on the DexboticVLM mentioned above, we further pretrain the discrete VLA model named *Dexbotic-Base*. It is further pre-trained on single-arm data, which includes the subset of Open-X Embodiment dataset [22], simulation data from multiple simulators such as RLBench [12], LIBERO [19] and ManiSkill2 [10], and some real robot data (e.g. UR5). The DexboticVLM is trained to predict the N discrete tokens to decode the discrete actions. Here, N is the degree of freedom (DoF). During training, continuous actions of grounding truth are divided into 256 bins, which is used to supervise the predicted discrete tokens. The pretrained discrete model *Dexbotic-Base* can be directly used to finetune any VLM-based manipulation and navigation policies. The user can directly employ it for both discrete and continuous action learning. The parameters of VLM part can be loaded from the Dexbotic-Base checkpoint. For continuous action modeling, such as the diffusion [16] and flow matching [2] processes, it can be performed by adding an action expert based on the DexboticVLM. The parameters of the action expert part can be randomly initialized.

Single-arm Continuous Model: Here, we illustrate how to perform continuous pretraining for CogACT in Dexbotic. Based on *Dexbotic-Base*, we further train the entire CogACT model, including VLM and DiT head, through continuous representation pre-training. The VLM is initialized with *Dexbotic-Base* and DiT head is randomly initialized. The GT continuous action is used to supervise the prediction of DiT. We use the data from multiple data resources,

including the subset of Open-X embodiment dataset and part of our private dataset we collected. Our private dataset includes 52 manipulation tasks collected using eight single-arm real-world robots. These single-arm robots include UR5, Franka, Unitree Z1, Realman GEN72, Realman 75-6F, UMI, ARX5 and WidowX. Note that those robots have different embodiedments with different DoF, which challenges our infrastructure capacity. The pretrained model obtained is called *Dexbotic-CogACT*.

Hybrid-arm Continuous Model: The original CogACT policy [16] does not support the multi-view and two-arm setting. To support two-arm tasks, we modify the number of noise tokens from 7 to 16, covering both 6-DoF and 7-DoF arms. The front-half tokens represent the actions of left arm while the second-half tokens denote the right arm one. We perform continue training on the robotic data of the hybrid arms based on the single-arm continuous model *Dexbotic-CogACT* mentioned above. Despite the single-arm data above, we further introduce Robomind [31], AgiBot World [5] datasets, as well as the private two-arm datasets collected from our ALOHA. For single-arm data, the single-arm actions are used to supervise the front-half tokens while the loss of second-half tokens is ignored during training. To support multi-view inputs, we share the vision encoder for multi-view images and their visual tokens are processed and concatenated as input for later LLM.

4.4. Experiment Layer

Experiment layer is the most important part in Dexbotic toolbox. In this section, we first describe how to develop new experiments from scratch. After that, the training pipeline and the inference service are presented in detail.

4.4.1 Experiment Development

Based on various models developed in Model Layer, we further develop the experiment scripts for model training and inference. As shown in Fig. 4, we adopt a layered configuration architecture. For each experiment class, it contains the feature configurations on trainer, data, optimizer, model and inference. The data and model configurations further include the action and tokenizer configurations, respectively. We first construct a *base_exp* script in dexbotic, which serves as the basis configuration of VLA policies. This script mainly contains the configuration on optimizer, trainer, action, data, model and inference.

To develop the experiment script for existing VLA policies, we inherit the *base_exp* script and modify the corresponding configuration for the target policy (e.g., *CogACT_Exp*). To run new experiments on existing policies, users can simply modify these configuration settings. As long as users inherit the corresponding configuration and override the fields, users can effortlessly fork a new set of

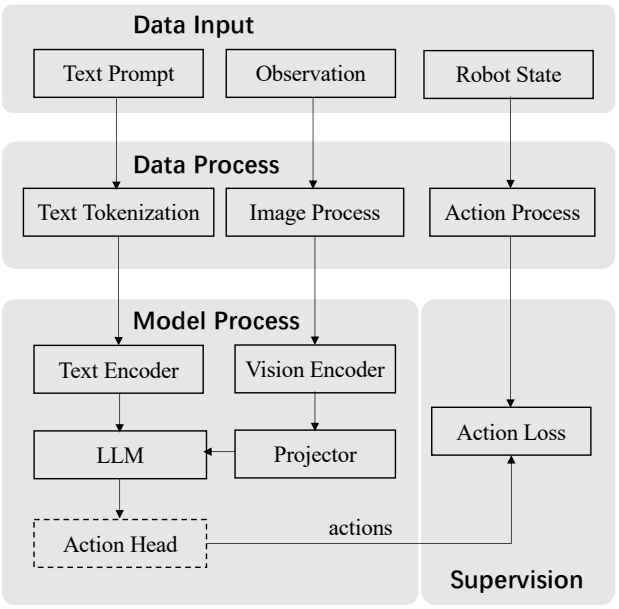


Figure 5. The training pipeline of Dexbotic.

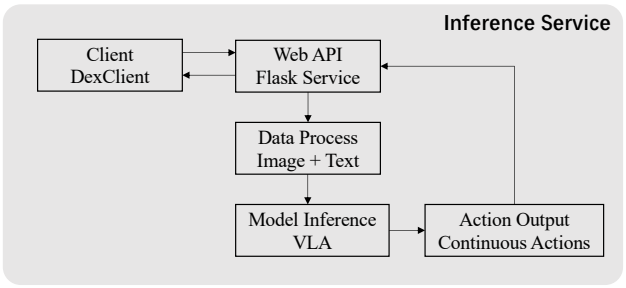


Figure 6. The inference service of Dexbotic.

hyperparameters without the need to copy the entire file. To develop the custom policies, users need to inherit and modify the configuration and model classes. Once the experiment script is correctly created, the users can directly run the experiment script to perform model training or inference like: *python xxx_exp.py -task train*. Here, the *task* denotes train or inference.

4.4.2 Training Pipeline

As shown in Fig. 5, the data inputs of Dexbotic include the observation, textual instruction and robot states. The textual prompt is tokenized and input to the text encoder to generate the text tokens. The observation image is firstly processed by the vision encoder to generate the image tokens and aligned to the text space by a light-weight MLP-based projector. The image and text tokens are concatenated together and further input to the LLM to generate discrete tokens. For discrete-representation policy, these tokens can be directly decoded into sparse actions while for the contin-

Table 1. Performance comparison on SimplerEnv-Bridge with WidowX robot between the state-of-the-art policies and dexbotic version.

Methods	Spoon on Towel	Carrot on Plate	Stack Cube	Eggplant in Basket	Avg. Suc
CogACT	71.7	50.8	15.0	67.5	51.3
DB-CogACT	87.5	65.3	29.2	95.8	**69.5** (+18.2)
OFT	12.5	4.2	4.2	100.0	30.2
DB-OFT	91.7	76.4	43.1	94.4	**76.4** (+46.2)
MemVLA	75.0	75.0	37.5	100.0	71.9
DB-MemVLA	100.0	66.7	70.8	100.0	**84.4** (+12.5)

Table 2. Performance comparison on four tasks in RoboTwin2.0 between the state-of-the-art policies and their dexbotic version.

Methods	Adjust Bottle	Grab Roller	Place Empty Cup	Place Phone Stand	Avg. Suc
CogACT	87	72	11	5	43.75
DB-CogACT	99	89	28	18	**58.5** (+14.75)

Table 3. Performance comparison on LIBERO between the state-of-the-art policies and their dexbotic version.

Methods	Spatial	Object	Goal	Long	Avg. Suc
CogACT	97.2	98.0	90.2	88.8	93.6
DB-CogACT	93.8	97.8	96.2	91.8	**94.9** (+1.3)
MemVLA	98.4	98.4	96.4	93.4	96.7
DB-MemVLA	97.2	99.2	98.4	93.2	**97.0** (+0.3)

uous representation, an action expert is usually appended to produce continuous-value action chunking. The generated action sequence is supervised by the ground-truth of actions using the corresponding action losses.

4.4.3 Inference Service

Dexbotic also provides the inference service for different developers. As shown in Fig. 6, DexClient sends a request to the Web API over the network. Web API based on the Flask Service, receives and process the request from DexClient. After receiving data from the DexClient, the system performs the data process on image and text, making the data suitable for next VLA model inference. The VLA model takes the image and text prompt as input and produces the continuous actions. The generated action sequences are then sent back to the Web API and Client sequentially. The DexClient performs corresponding actions based on these resulting actions.

5. Benchmarks

To validate the effectiveness of pretrained models we provided, we conduct the experiments and perform performance comparison on multiple simulation benchmarks: LIBERO [19], SimplerEnv [17], CALVIN [21], ManiSkill2 [10], Robotwin2.0 [8]. We first simply describe these simulation benchmarks and then show detailed benchmarking results on them.

5.1. Simulation Benchmarks

SimplerEnv aims to narrow the gap between simulation and the real world environment. It provides two embodiedments, Google robot and WidowX robot, and two task suites, Visual Matching and Variant Aggregations, for fair evaluation. In this paper, we focus mainly on the WidowX robot with only Visual Matching suite. It includes four tasks: *Put Spoon on Towel*, *Put Carrot on Plate*, *Stack Cube* and *Put Eggplant in Yellow Basket*.

CALVIN targets long-horizon language-conditioned robot manipulation tasks. We perform experiments under the standard ABC-D setting. The model is trained on environments A, B, and C, and evaluated on generalization capability on environment D. We report the average success rate over 1000 rollouts per task, along with the average number of tasks completed consecutively to accomplish five instructions.

ManiSkill2 mainly focuses on the basic pick-and-place. We evaluated the experimental results on five representative tasks: PickCube, StackCube, PickSingleYCB, PickSingleEGAD, and PickClutterYCB. Those tasks require the robot to grasp the specific object and place it in a 3D position indicated by a green marker, evaluating 3D perception and spatial reasoning.

RoboTwin2.0 is a newly-introduced simulation benchmark. It improves sim-to-real transformation and contains 50 dual-arm tasks and five robot embodiments. In this paper, we conduct the comparison based on four carefully selected tasks: adjust bottle, grab roller, place empty cup, and place phone stand.

LIBERO includes five task suites, and each suite is de-

Table 4. Performance comparison on CALVIN between the state-of-the-art policies and their dexbotic version. We perform experiments on the ABC→D split, where VLA models are trained with data from A, B, and C environments and evaluated in environment D that is unseen during training.

Methods	1	2	3	4	5	Avg. Len
CogACT	0.838	0.729	0.640	0.559	0.480	3.25
DB-CogACT	0.935	0.867	0.803	0.760	0.698	**4.06**(+0.81)
OFT	0.891	0.794	0.674	0.598	0.515	3.47
DB-OFT	0.928	0.807	0.692	0.602	0.511	**3.54**(+0.07)

Table 5. Performance comparison on five tasks in ManiSkill2 between the state-of-the-art policies and their dexbotic version.

Methods	PickCube	StackCube	PickSingleYCB	PickSingleEGAD	PickClutterYCB	Avg. Suc
CogACT	55	70	30	25	20	40
DB-CogACT	90	65	65	40	30	**58** (+18.0)
OFT	40	45	5	5	0	21
DB-OFT	90	75	55	65	30	**63** (+42.0)

signed to evaluate specific capabilities. *LIBERO-Spatial* mainly focuses on different positions to place the objects. *LIBERO-Object* involves pick and place various objects into the box within a fixed scene layout. *LIBERO-Goal* evaluates the ability to perform various operations in a fixed layout. *LIBERO-Long*, also called *LIBERO-10*, which targets 10 long-horizon tasks that involve various scenes and operations. *LIBERO-90* is an expanded version of LIBERO-10, presenting a more challenging benchmark.

5.2. Benchmark Results

SimplerEnv: As shown in Tab. 1, on the challenging SimplerEnv benchmarks, DB-CogACT with the pretrained model outperforms the official CogACT by absolute 18.2%. DB-OFT achieves absolute 46.2% improvements compared to the official OpenVLA-OFT. Moreover, we further evaluate the effectiveness of pretrained models on the MemoryVLA, a state-of-the-art VLA on SimplerEnv. The experimental result shows DB-MemoryVLA achieves 84.4% success rate, achieving more than absolute 12% improvements. The great improvements indicate the strong representation power of our pretrained models.

CALVIN: To evaluate the improvements on long-horizon tasks, we conduct the performance comparison between VLA policies and their dexbotic countparts (see Tab. 4). DB-CogACT outperforms the official CogACT on all metrics. It achieves 4.06 average length, surpassing the CogACT by 0.81. Moreover, DB-OFT also achieves better performance than the standard OpenVLA-OFT.

RoboTwin 2.0: Here we mainly take the CogACT as an example to show the effectiveness of dexbotic under the easy mode. As shown in Tab. 2, for four selected tasks: adjust the bottle, grab roller, place empty cup and place the phone stand, CogACT achieves an average success rate of 43.75%. In comparison, DB-CogACT surpasses CogACT by 14.75% absolute gains with 58.5% success rate. It demonstrates that our pretrained model can bring large performance improvements under the dual-arm embodiment.

LIBERO: On LIBERO benchmark, the performance of state-of-the-art VLA policies is nearly saturated (see Tab. 3). With our Dexbotic pre-trained models, those policies can obtain some further performance improvements for them like CogACT and MemoryVLA. Specifically, DB-CogACT boosts the average success rate on four task suites by 1.3% points, compared to the CogACT baseline.

ManiSkill2: As shown in Tab. 5, the original OpenVLA-OFT produces undesirable performance with 21% average success rate among five tasks. In comparison, DB-OFT improves the absolute performance by 42% points, which demonstrates the effectiveness of our pretrained model. Moreover, DB-CogACT further improves the average success rate by 18% points compared to the strong baseline of original CogACT.

6. Real-world Performance

To showcase what tasks users can accomplish in real world using the Dexbotic toolbox, we release the task gallery for visualization (see Fig. 8). On different robots like UR5e, ALOHA, ARX5 and Franka, we collect the real-world task data through the teleoperation. For each task, we collect 500-1000 demonstrations depending on the task difficulty and convert those data into our Dexdata format. These converted data is used to finetune the corresponding models based on our pretrained model.

The real world experiments show that Dexbotic can accomplish various daily tasks. Notably, it achieves 100% and 80% success rates for the *set the plates* and *search the green*

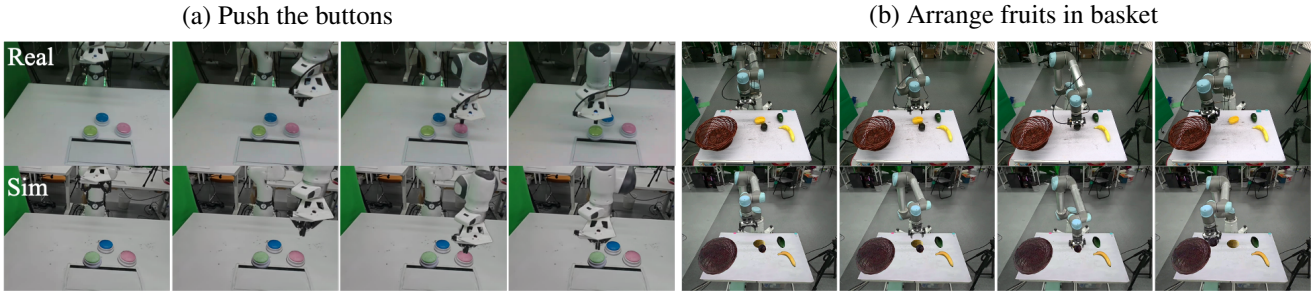


Figure 7. Comparison of real-world and simulated renderings, highlighting the high consistency in the robotic arm and objects performing the same action in both environments.

box tasks, respectively. However, for those fine-grained manipulation tasks like *Shred the scrap paper* and *Pour fries into plate*, they indeed pose challenges for existing VLA policies. Moreover, we verify that some state-of-the-art VLA policy like MemoryVLA [26] can solve the long-horizon and memory-requiring tasks, like *Push buttons sequentially*. Please see the official website of Dexbotic for more visualization on real-world tasks, and we would like to encourage the users to utilize the Dexbotic toolbox to develop more real-world robotic tasks. We would also suggest users submit more policies developed based on Dexbotic to RoboChallenge for fair comparison in real-world.

7. Real2Sim Evaluation

Real-world evaluations are often labor-intensive. To address this challenge, we propose Dexbotic Open Source-Twins (DOS-Twins), a Real2Sim2Real [18, 23] simulator developed as part of the Dexbotic ecosystem. For the publicly released real-world datasets, we reconstruct corresponding simulation environments that closely replicate the real setups. Users can leverage these datasets for model training and subsequently submit their trained models to our Real2Sim evaluation interface for comprehensive capability assessment. DOS-Twins ensures consistency between simulation and real-world across three key dimensions:

Visual Consistency. The VLA model is highly sensitive to visual alignment. This misalignment leads to discrepancies in success rates, with high success in simulation but significantly lower success in real-world tasks. Leveraging 3D Gaussian Splatting (3DGS) [13], we generate photorealistic renderings with precise alignment between rendered objects and their corresponding meshes. Accurate camera calibration guarantees that the simulated camera viewpoints perfectly match those of real-world cameras.

Motion Consistency. Motion consistency is crucial for ensuring that the simulated and real-world control systems are aligned. Proper alignment prevents incorrect visual feedback and ensures that the model can reliably execute the intended actions, as predicted by the simulation, in real-world tasks. Without consistency in motion, the robotic arm may not perform as expected, leading to discrepancies between simulated success and real-world failure. The low-level controller of the robotic arm is calibrated to match the motion dynamics and kinematic characteristics of the real hardware.

Interaction Consistency. In grasping tasks, ensuring accurate interaction between the gripper and the object is critical, as the geometric and physical alignment directly impacts the model's ability to perform tasks successfully. To achieve this, we perform high-precision 3D scanning of both the gripper and objects, ensuring that the geometric structure errors remain within a millimeter for the objects. This high level of precision minimizes interaction errors, maintaining millimeter-level accuracy during grasping. Additionally, we align the parallel structure of the gripper with that of the real hardware, ensuring consistent physical interactions between the gripper and objects. This consistency enables reliable performance in both simulated and real-world environments.

Specifically, DOS-Twins employs Isaac Sim as the backend physics engine and 3DGS as the rendering frontend, achieving high-precision reconstruction and visualization of robotic arms and their components. It supports multiple robotic arm and gripper configurations and allows rendering from any viewpoint depending on task requirements. All simulation assets are organized modularly, enabling users to reuse and customize them for specific tasks. Moreover, users can follow our simulation construction workflow and utilize the provided developer tools to build new environments. These environments can automatically monitor model performance across various robotic platforms and task settings during pre-training.

As illustrated in Fig. 7, we compare the simulated environment with the real setup through replay analysis. The results show that the manipulation behavior in our simulations closely aligns with real-world phenomena, demonstrating that users can train policies in the real world while conducting consistent evaluations within our simulation framework. Please access the official website for more visualization comparison on the visual, motion and interaction consistency.

References

[1] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024. 3

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. pi-0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024. 1, 2, 3, 4

[3] Kevin Black, Noah Brown, et al. pi-0.5: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. 1, 3

[4] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. 1, 3, 4

[5] Qingwen Bu, Jisong Cai, Li Chen, Xiuqi Cui, Yan Ding, Siyuan Feng, Shenyuan Gao, Xindong He, Xuan Hu, Xu Huang, et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. *arXiv preprint arXiv:2503.06669*, 2025. 5

[6] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. https://github.com/huggingface/lerobot, 2024. 2

[7] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, et al. Mmdetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019. 1

[8] Tianxing Chen, Zanxin Chen, Baijun Chen, Zijian Cai, Yibin Liu, Qiwei Liang, Zixuan Li, Xianliang Lin, Yiheng Ge, Zhenyu Gu, et al. Robotwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025. 6

[9] An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Zaitian Gongye, Xueyan Zou, Jan Kautz, Erdem Bıyık, Hongxu Yin, Sifei Liu, and Xiaolong Wang. Navila: Legged robot vision-language-action model for navigation. *arXiv preprint arXiv:2412.04453*, 2024. 3

[10] Jiayuan Gu, Fanbo Xiang, Xuanlin Li, Zhan Ling, Xiqiang Liu, Tongzhou Mu, Yihe Tang, Stone Tao, Xinyue Wei, Yunchao Yao, et al. Maniskill2: A unified benchmark for generalizable manipulation skills. *arXiv preprint arXiv:2302.04659*, 2023. 4, 6

[11] Peilong Han, Fan Jia, Min Zhang, Yutao Qiu, Hongyao Tang, Yan Zheng, Tiancai Wang, and Jianye Hao. Muvla: Learning to explore object navigation via map understanding. *arXiv preprint arXiv:2509.25966*, 2025. 3

[12] Stephen James, Zicong Ma, David Rovick Arrojo, and Andrew J Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020. 4

[13] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1, 2023. 8

[14] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025. 1, 3, 4

[15] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024. 1, 3, 4

[16] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024. 1, 2, 3, 4, 5

[17] Xuanlin Li, Kyle Hsu, Jiayuan Gu, Karl Pertsch, Oier Mees, Homer Rich Walke, Chuyuan Fu, Ishikaa Lunawat, Isabel Sieh, Sean Kirmani, et al. Evaluating real-world robot manipulation policies in simulation. *arXiv preprint arXiv:2405.05941*, 2024. 2, 6

[18] Xinhai Li, Jialin Li, Ziheng Zhang, Rui Zhang, Fan Jia, Tiancai Wang, Haoqiang Fan, Kuo-Kun Tseng, and Ruiping Wang. Robogsim: A real2sim2real robotic gaussian splatting simulator. *arXiv preprint arXiv:2411.11839*, 2024. 8

[19] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023. 4, 6

[20] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023. 4

[21] Oier Mees, Lukas Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7(3):7327–7334, 2022. 2, 6

[22] Abby O'Neill, Abdul Rehman, Abhiram Maddukuri, Abhishek Gupta, Abhishek Padalkar, Abraham Lee, Acorn Pooley, Agrim Gupta, Ajay Mandlekar, Ajinkya Jain, et al. Open x-embodiment: Robotic learning datasets and rt-x models: Open x-embodiment collaboration 0. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6892–6903. IEEE, 2024. 4

[23] M Nomaan Qureshi, Sparsh Garg, Francisco Yandun, David Held, George Kantor, and Abhisesh Silwal. Splatsim: Zero-shot sim2real transfer of rgb manipulation policies using gaussian splatting. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6502–6509. IEEE, 2025. 8

[24] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021. 2, 4

[25] Sabela Ramos, Sertan Girgin, Léonard Hussenot, Damien Vincent, Hanna Yakubovich, Daniel Toyama, Anita Gergely, Piotr Stanczyk, Raphael Marinier, Jeremiah Harmsen, Olivier Pietquin, and Nikola Momchev. Rlds: an ecosystem to generate, share and use datasets in reinforcement learning, 2021. 2

[26] Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2508.19236*, 2025. 1, 3, 4, 8

[27] Lin Sun, Bin Xie, Yingfei Liu, Hao Shi, Tiancai Wang, and Jiale Cao. Geovla: Empowering 3d representations in vision-language-action models. *arXiv preprint arXiv:2508.09071*, 2025. 1

[28] Peter Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Adithya Jairam Vedagiri IYER, Sai Charitha Akula, Shusheng Yang, Jihan Yang, Manoj Middepogu, Ziteng Wang, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *Advances in Neural Information Processing Systems*, 37:87310–87356, 2024. 4

[29] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023. 1, 3

[30] Yuqing Wen, Hebei Li, Kefan Gu, Yucheng Zhao, Tiancai Wang, and Xiaoyan Sun. Llada-vla: Vision language diffusion action models. *arXiv preprint arXiv:2509.06932*, 2025. 1

[31] Kun Wu, Chengkai Hou, Jiaming Liu, Zhengping Che, Xiaozhu Ju, Zhuqin Yang, Meng Li, Yinuo Zhao, Zhiyuan Xu, Guang Yang, et al. Robomind: Benchmark on multi-embodiment intelligence normative data for robot manipulation. *arXiv preprint arXiv:2412.13877*, 2024. 5

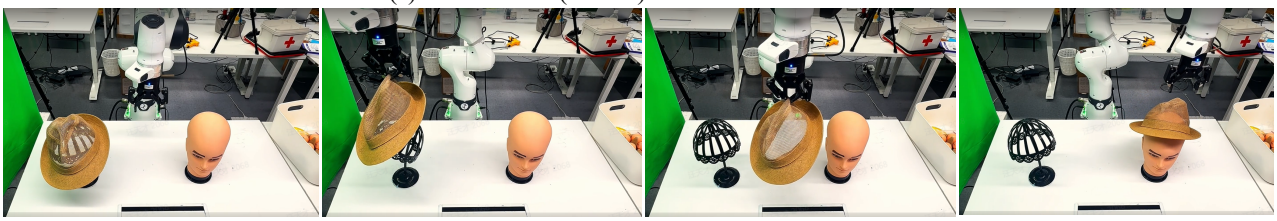
[32] Yucin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019. 1

[33] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025. 1

[34] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11975–11986, 2023. 2

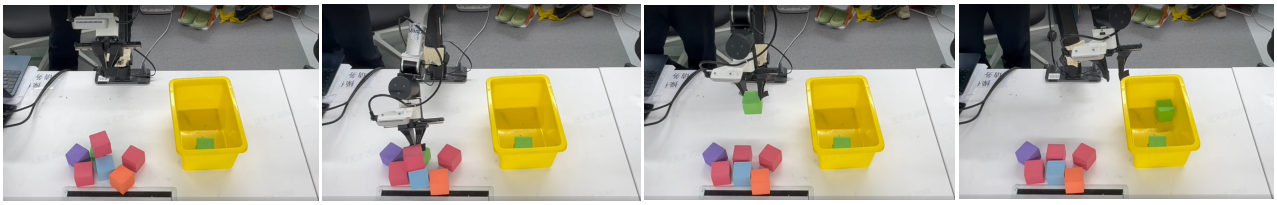
[35] Jiazhao Zhang, Kunyu Wang, Rongtao Xu, Gengze Zhou, Yicong Hong, Xiaomeng Fang, Qi Wu, Zhizheng Zhang, and He Wang. Navid: Video-based vlm plans the next step for vision-and-language navigation. *arXiv preprint arXiv:2402.15852*, 2024. 3

(a) Wear the hat (Franka) Success rate: 70%

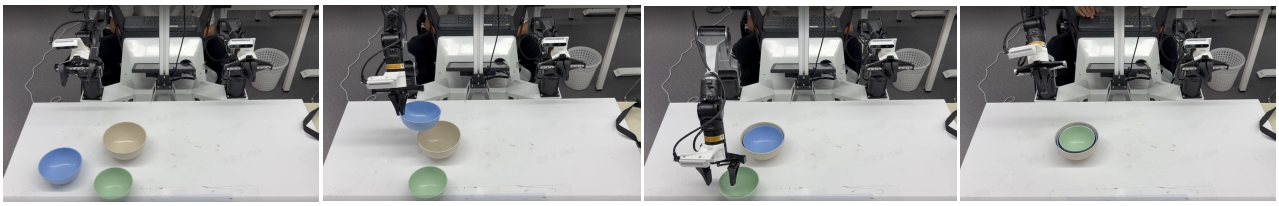


(b) Set the plates (UR5e) Success rate: 100%

(c) Search green box (ARX5) Success rate: 80%

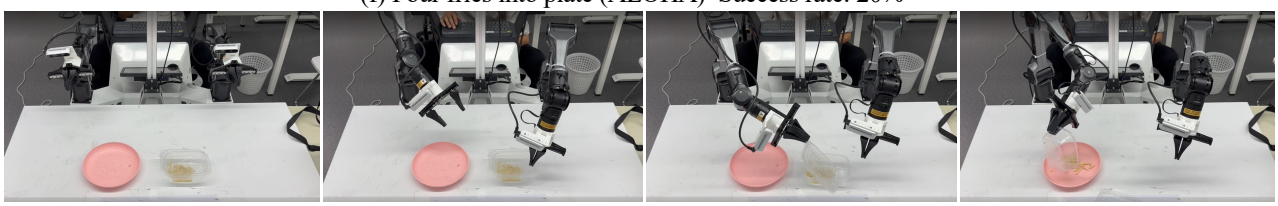


(d) Stack the bowls (ALOHA) Success rate: 90%



(e) Push the buttons (Franka) Success rate: 60%

(f) Pour fries into plate (ALOHA) Success rate: 20%



(g) Shred scrap paper (UR5e) Success rate: 40%

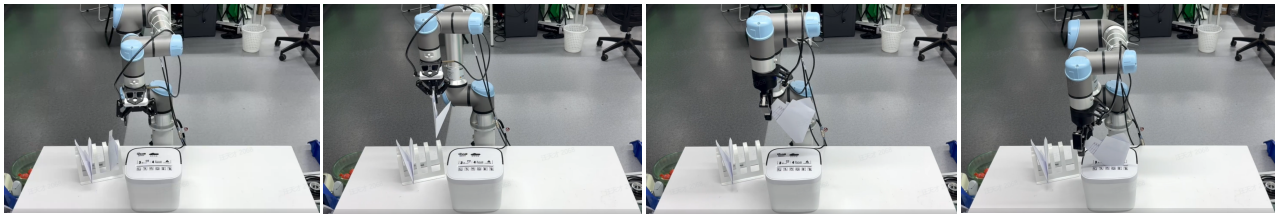


Figure 8. The video gallery produced by Dexbotic toolbox.