

RC-NF: Robot-Conditioned Normalizing Flow for Real-Time Anomaly Detection in Robotic Manipulation

Supplementary Material

1. Experimental Demonstration Videos

Simulated Environment Results.mp4 presents three anomalies of LIBERO-Anomaly-10: Gripper Open, Gripper Slippage, and Spatial Misalignment, along with the corresponding anomaly scores inferred by RC-NF.

Real World Experiments Results.mp4 presents real-world experiments, exhibiting how the RC-NF monitoring mechanism enhances the robustness and adaptability of the model π_0 , in task-level and state-level OOD situations.

2. Further details on Task Embedding

Implementation Details of Task Embedding. This section explains the implementation of task embedding τ from Sec. 3.2. Fig. 8 (left) shows the ten tasks in the LIBERO-10. In deployment, the first step is to obtain 10 vectors uniformly distributed on a hypersphere with radius $R = 5$ and dimension $T = 12$ using an optimization algorithm. Then, these 10 task instructions are mapped one-to-one to the spherical vectors, resulting in $[\tau_1, \tau_2, \dots, \tau_{12}]$. Fig. 8 (right) provides detailed examples to clarify the mapping process. When $T = 3$ and the number of tasks is 6, this results in 6 spherical vectors on a 3-dimensional sphere, intersecting the x , y , and z -axes, as represented by $[\tau_1, \tau_2, \tau_3]$.

Role of Task Embedding. As discussed in Sec. 4.3, removing the task embedding limits the model to detecting only dataset-level OOD samples, preventing it from identifying task-specific anomalies. To illustrate this, consider the Spatial Misalignment setup: during training, the book is placed in the back compartment, while in the test scenario, the robotic arm places the book in the left, front, or right compartments, referred to as the 3 tasks (see Fig. 9(right)). Tables 1 and 2 show that without these tasks in the training set, the action trajectories are out-of-distribution, and Spatial Misalignment is classified as a dataset-level OOD anomaly. When these 3 tasks are included in the training set, the action trajectories become in-domain, and detecting Spatial Misalignment is recognized as a task-specific anomaly. Fig. 9 compares the metrics for this scenario. After adding the 3 tasks, the performance of the RC-NF model shows almost no decrease. However, without the task embedding, the RC-NF monitoring performance drops to around 0.6, emphasizing the importance of task embedding in distinguishing tasks within the training set. In contrast, FailDetect’s performance drops to 0.5, highlighting its inability to differentiate tasks within the same training set.

Algorithm 1 Training Loop with BalancedHardSampler

```
1: for epoch  $\leftarrow$  startEpoch to numEpochs do
2:   model.train()
3:   for data in trainLoader do
4:     ... // process data, compute
       loss, update model
5:   end for
6:   if epoch = NextStageEpoch then
7:     model.eval()
8:     for data in trainLoader do
9:       update_weights(model(data))
10:    end for
11:    sampler  $\leftarrow$  BalancedHardSampler(weights)
12:  end if
13:  if epoch  $\geq$  NextStageEpoch then
14:    update_train_loader(sampler)
15:  end if
16: end for
```

3. Training Data Debiasing

The Training Data Debiasing Procedure aims to address the issue of imbalanced sample distribution during robotic manipulation. Specifically, in the initial stages of a robotic arm’s motion, the motion trajectories tend to be more consistent, whereas near the object grasping phase, the trajectories exhibit greater variation. This results in a higher frequency of similar trajectories from the initial movement phase, causing an uneven sample distribution. To mitigate this, we design the **BalancedHardSampler** class, designed to equalize the distribution of these samples, ensuring a more balanced and robust training process.

Algorithm 1 illustrates the training procedure, which is divided into two stages. In the first stage, standard sampling is used to train RC-NF. Upon reaching the NextStageEpoch, inference results are collected for sample evaluation. The BalancedHardSampler is then initialized to reduce redundant samples while increasing the diversity of underrepresented samples, ensuring a more balanced sample distribution throughout the demonstration. After the NextStageEpoch, the BalancedHardSampler is continuously employed for sample collection during the remaining stages.

4. Implementation Details for Simulations

As mentioned in the Implementation Details in Sec. 4.1, computer graphics techniques are used in the simulation

environment to generate the first frame’s bounding box for SAM2. This method of obtaining the bounding box is stable, low-cost, and ensures reproducibility. The following section provides a detailed explanation of the process.

Generating Sample Points in Geometry. For each object, we generate a grid of sample points within the geometry. Assuming the geometry size is $\text{geom_size} = [W, H, D]$, where W , H , and D represent the width, height, and depth of the geometry, respectively, the offset for each sample in a $5 \times 5 \times 5$ grid is computed as:

$$\text{local_offset} = \begin{bmatrix} dx \cdot W \\ dy \cdot H \\ dz \cdot D \end{bmatrix}.$$

Here, dx, dy, dz are the offsets in the grid along the width, height, and depth axes of the geometry.

These local offsets are then transformed into world coordinates using the geometry’s position in the world geom_pos and its rotation matrix R :

$$P_{\text{world}} = \text{geom_pos} + R \cdot \text{local_offset}.$$

In this equation, geom_pos is the position of the geometry in the world frame (a 3D vector representing the center of the geometry), and R is the rotation matrix that transforms the local offset into the world frame.

Projection of 3D Points to 2D. After obtaining the world coordinates of all sample points P_{world} , we project them onto the 2D image plane using the camera’s intrinsic matrix K and the extrinsic matrix Extrinsic :

$$P_{\text{image}} = K \cdot (\text{Extrinsic} \cdot P_{\text{world}}).$$

In this equation, Extrinsic represents the transformation from the world coordinates to the camera coordinates (a combination of rotation and translation), and $P_{\text{image}} = [x, y]^T$ represents the 2D coordinates of the projected point on the image plane.

Filtering Invalid Projections. We filter out invalid projections, such as those behind the camera or outside the image boundaries. For each projected point $P_{\text{image}} = [x, y]$:

$$0 \leq x < \text{image_width}, \quad 0 \leq y < \text{image_height}.$$

Bounding Box Calculation. For the valid projected points, we calculate the bounding box by finding the minimum and maximum x and y coordinates:

$$x_{\min} = \min(x), \quad x_{\max} = \max(x).$$

$$y_{\min} = \min(y), \quad y_{\max} = \max(y).$$

The bounding box is then defined as:

$$\text{bbox} = [x_{\min}, y_{\min}, x_{\max}, y_{\max}].$$

5. Implementation Details in the Real World

Bounding Box Generation. In this paper, we utilize the multimodal large model Gemini 2.5 Prof[8], which can understand the manipulated object from task instructions and generate a zero-shot bounding box for SAM2. As illustrated in Fig. 10, the process begins by extracting the OBJECT_LIST (e.g., *blue ball, drawer*) from the TASK_DESCRIPTION, such as *Place the blue ball into the open drawer*. The image and identified objects are subsequently passed to Gemini, which generates a JSON list.

Description of the Homing Procedure. The homing procedure refers to the process by which a robot returns from an unknown or arbitrary position to a predefined *home position*, which serves as the starting point for the robotic operations. We utilize the homing procedure to perform task rollback in state-level OOD situations. The intervention of RC-NF’s monitoring mechanism allows the robotic arm to adjust its trajectory without the need to completely return to the *home position* when an error is encountered.

Explanation of the Real-Time Characteristic. RC-NF can detect anomalies within 100 ms on a consumer-grade GPU, such as the Nvidia GeForce RTX 3090. Below we report the latency breakdown for a single-frame inference. At 100 ms, this performance is faster than the average human reaction time, and thus, we refer to the operations involved in RC-NF as capable of real-time anomaly detection.

SAM2 Inference	Grid Sampling	RC-NF Inference	Other Latency	Overall (RTX 3090)
50 ms	1.7 ms	30 ms	5 ms	86.7 ms

6. VLMs as anomaly scorers

In the simulation experiments presented in Sec. 4.2, we utilize VLMs such as GPT-5 [29], Gemini 2.5 Pro [8], and Claude 4.5 [4] for anomaly scoring. To reduce the effects of network latency and extended inference times on large model performance during streaming, we employ a parallel API invocation strategy and sample results at 1 Hz.

Fig. 11 illustrates the prompt provided to the model, which is an improved version derived from Sentinel, adapted for anomaly detection scoring. We provide VLMs with the current time (TIME) and the total time allocated for the task (TIME_LIMIT), enabling them to calculate the progress ratio of the task. DESCRIPTION refers to the task description, such as *Pick up the book and place it in the back compartment of the caddy*. FRAME_RATE denotes the sampling frequency, set here to 1 Hz. CURRENT represents a sequence of 256×256 images, with the number of images being $\text{TIME}/\text{FRAME_RATE}$, which captures the visual progression of the task from start to present. This enables the VLMs to capture the complete image sequence.

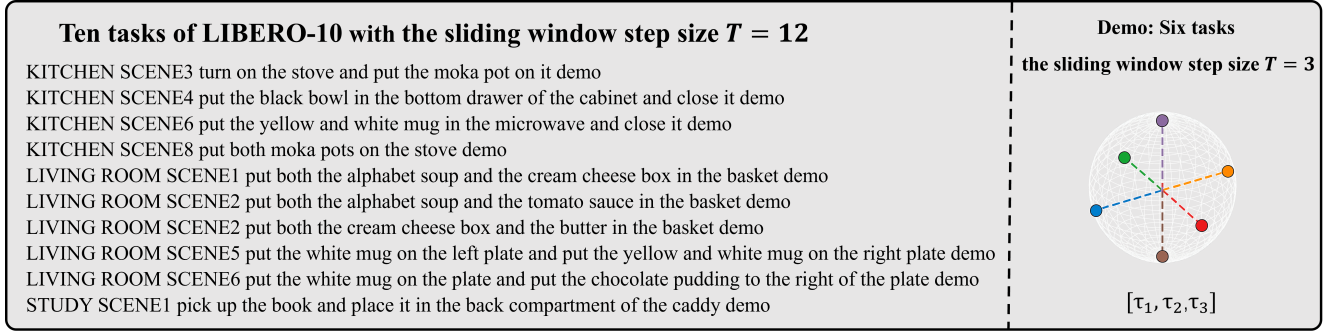


Figure 8. Diagram illustrating Spherical Uniform Encoding. The left side displays the ten tasks from the training set along with their descriptions, while the right side provides an example demonstration.

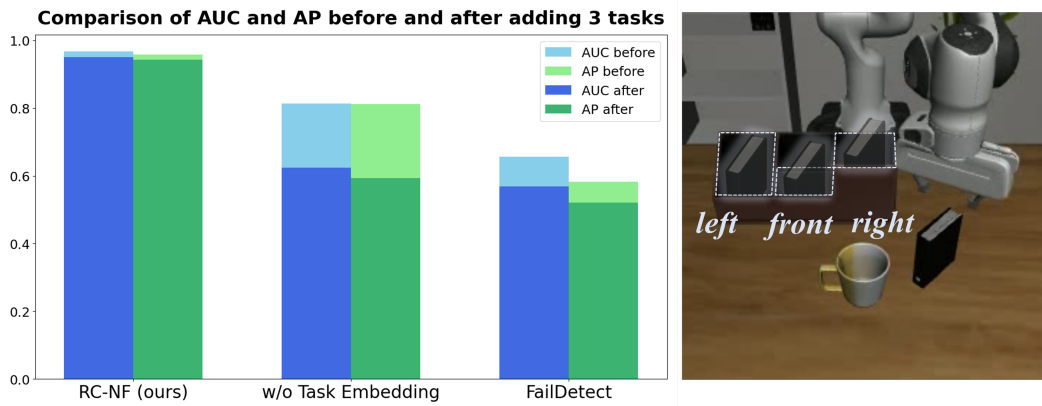


Figure 9. Comparison of performance metrics before and after incorporating three tasks with spatial misalignment (left, front and right) into the training set.

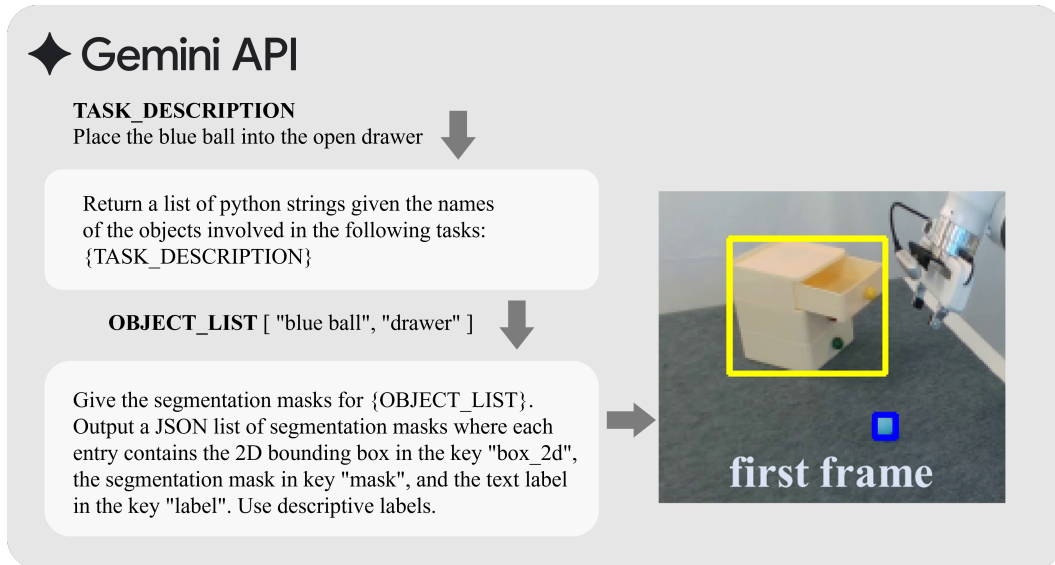


Figure 10. Prompts for generating bounding boxes with Gemini 2.5 Pro

prompts:

- role: system

content:

You are the runtime monitor for one autonomous mobile manipulator robot capable of solving common household tasks.

A camera system captures a sequence of image frames (i.e., a short video) of the robot executing its current task online.

The image frames are captured at approximately {FRAME_RATE} from a third-person view.

Note: The image resolution is relatively low, so assume that the appearance and identity of objects are correct and stable.

Focus your monitoring primarily on the execution quality of the robot's actions (e.g., movement trajectories, interactions with objects, efficiency, and safety), rather than on detecting fine-grained visual details of the objects.

As a runtime monitor, your job is to analyze the sequence of frames and assign an **anomaly score** (0–10) for the **last frame** of the sequence, where higher scores mean more abnormal behavior.

Scoring guidelines:

- **0–3** → Normal behavior (no clear anomalies, task is being executed as expected).

- **4–6** → Slightly abnormal or uncertain (some inefficiency, hesitation, or mild mistakes, but still likely recoverable).

- **7–10** → Strongly abnormal or failing (robot is acting incorrectly, unsafely, or unlikely to complete the task in time).

The robot's current task is to {DESCRIPTION}.

The robot may take up to {TIME_LIMIT} seconds to complete this task.

The current elapsed time is {TIME} seconds.

You should consider the entire sequence of frames to understand context and trends, but your final anomaly score must reflect the robot's state in the **last frame**.

You must strictly follow the output format shown below.

Do not add bold, headings, or Markdown symbols.

Do not explain your reasoning outside the required sections.

Output must always begin with [start of output] and end with [end of output]:

[start of output]

Questions: Generate a few task-relevant questions that help you assess the robot's behavior and all task-relevant objects across the video, focusing mainly on the robot's action execution.

Answers: Answer your questions, with fine-grained visual details from the sequence of frames.

Analysis: Analyze whether the robot is behaving normally or abnormally in the last frame, making use of the context from earlier frames.

Explicitly reference the scoring guidelines (0–3 normal, 4–6 slightly abnormal, 7–10 strongly abnormal) when justifying your score.

Anomaly score: {SCORE between 0 and 10}

[end of output]

Rules:

1. Always return exactly one integer anomaly score between 0 and 10. The output must strictly follow the format: Anomaly score: <number>

2. Base your scoring on both the last frame and the context from earlier frames.

3. Always follow the scoring guidelines above when assigning scores.

4. Always focus more on the robot's execution quality than on object recognition.

5. Always start with [start of output] and end with [end of output].

6. Before giving your score, please include "Anomaly score:" followed by the number.

- role: user

content:

- type: "text"

text: "Video of the robot after {TIME} seconds of executing its task (frames shown sequentially):"

- type: "video"

video: "{CURRENT}"

- type: "text"

text: "Assess the video as per the instructions. Runtime monitor output:"

Figure 11. VLMs prompts for robotic anomaly detection scoring