Research & Strategy: Synchronization and Training Setup for RoboMaster with LeRobot

1. Do I need to do Diffusion before testing SmolVLA?

No, you do not.

- **Independent Architectures:** Diffusion Policies and Vision-Language-Action (VLA) models like SmolVLA are distinct architectures. While Diffusion Policies (like those used in ALOHA) are excellent for complex, multimodal manipulation, SmolVLA is designed to be a "generalist" model that integrates language understanding directly.
- **SmolVLA specifics:** SmolVLA typically uses a VLM (Vision-Language Model) backbone (like SigLIP + generic LLM) fused with an action head (often using Flow Matching or Diffusion for the action generation itself, but integrated). You train it end-to-end.
- **Recommendation:** Since your goal is a VLA, start directly with SmolVLA. Using Diffusion first would just be a data engineering sanity check, but if your data pipeline is correct, it's not a prerequisite for the model weights.

2. Simple Drive Task: "Point A to Point B"?

Yes, but with a visual target.

- **The "Goal" Problem:** VLA models are "instruction conditioned" (e.g., "Drive to the red box"). If you just drive from A to B in an empty room, the model has no visual feature to ground the instruction "Drive forward." It might just learn to output a constant speed regardless of what it sees.
- **Better Task:** "Chase the target." Place a distinct object (e.g., a bright orange cone or box) in the room. Drive the robot towards it from various angles.
  - *Instruction:* "Drive to the orange cone."
  - *Visual Learning:* The model learns: "When the cone is small/centered -> Move Forward." "When the cone is to the left -> Rotate Left." "When the cone is huge (close) -> Stop."

3. How many records?

Start with 50 Episodes.

- **Benchmarks:** For a simple task like "Drive to object" with a pre-trained VLA foundation (like SmolVLA which has seen millions of images), 50 demonstrations are typically sufficient to get a working policy.
- **Duration:** Keep episodes short (10-20 seconds). Total data: ~15 minutes of driving.
- **Diversity:** Crucial. Do not start from the same spot every time.

    - Start 1m away, 2m away.
    - Start offset to the left, offset to the right.
    - Change the lighting (turn lights on/off) if possible.

---

4. Synchronization and Latency Evaluation (Critical)

Q: Should I synchronize capture of commands to the video feed?
A: YES.
In Imitation Learning, causal misalignment is a silent killer.

- **The Scenario:** You see a turn in the video stream. You react and move the joystick.
- **The Reality:** The video you saw actually happened ~200ms ago (Wifi latency).
- **The Data Log:** If you log Current_Time, Joystick_Value, Current_Video_Frame, you are associating a *response* (joystick) with the *stimulus* (old frame) that caused it. This is actually **good** for behavioral cloning (Human reaction matches the delayed observation).
- **The Danger:** The danger is **system latency drift** or **variable jitter**. If the wifi lag spikes from 100ms to 500ms, your data becomes inconsistent.

How to Reliably Evaluate Latency?
You cannot rely on the SDK's internal timestamps alone because the clock on the Robot (sending video) and your PC (receiving video) are not perfectly synchronized (unless you run NTP, which is hard on the closed RoboMaster firmware).
Method: The "Step Response" Visual Test
This is the gold standard for measuring "Photon-to-Action" latency without specialized hardware.
Step-by-Step Python Implementation Plan:
You will write a script that sends a command and simultaneously looks for the visual result.

1. **Setup:** Point the robot's camera at a stopwatch (on a phone screen) OR setup the robot to face a static scene.
2. **The Trigger:**
    - The script sends a "violent" command: chassis.drive_speed(x=0, y=0, z=100) (Fast rotation).
    - *Simultaneously*, record the system timestamp: t_command = time.time().
3. **The Detector:**
    - The script reads the video stream in a loop using your SDK fork.
    - It calculates the "Optical Flow" or simple pixel difference between the current frame and the previous frame.
    - When the pixel difference exceeds a threshold (motion detected), record: t_motion = time.time().
4. **The Calculation:**
    - Latency = t_motion - t_command
    - Repeat this 10 times and take the average.

Code Snippet for Latency Test:

Python

```
import time
import cv2
import numpy as np
# Assuming your SDK structure
from robomaster_sdk import Robot

def measure_latency(robot):
    camera = robot.camera
    chassis = robot.chassis

    camera.start_video_stream(display=False)
    time.sleep(2) # Warmup

    latencies =

    for i in range(10):
        print(f"Test {i+1}...")

        # 1. Get baseline frame
        frame_prev = camera.read_cv2_image()
        gray_prev = cv2.cvtColor(frame_prev, cv2.COLOR_BGR2GRAY)

        # 2. Send Command & Mark Time
        t_start = time.time()
        chassis.drive_speed(x=0, y=0, z=50) # Spin z-axis

        # 3. Watch for Motion
        motion_detected = False
        while not motion_detected:
            frame_curr = camera.read_cv2_image()
            if frame_curr is None: continue

            gray_curr = cv2.cvtColor(frame_curr, cv2.COLOR_BGR2GRAY)

            # Simple frame difference
            diff = cv2.absdiff(gray_prev, gray_curr)
            _, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)
            motion_pixels = np.count_nonzero(thresh)
            # Tune this threshold based on your resolution
            if motion_pixels > 5000:
                t_detect = time.time()
                motion_detected = True

            # Timeout safety
            if time.time() - t_start > 2.0:
                break

        # Stop robot
        chassis.drive_speed(x=0, y=0, z=0)
        time.sleep(1) # Wait for stop

        if motion_detected:
            latency = (t_detect - t_start) * 1000 # ms
            latencies.append(latency)
            print(f"Latency: {latency:.2f} ms")
        else:
            print("Motion not detected (timeout)")

    avg_latency = sum(latencies) / len(latencies)
    print(f"Average System Latency: {avg_latency:.2f} ms")
    return avg_latency
```

Interpretation for Training:

- **Latency < 100ms:** Good. You can train directly mapping Frame_t to Action_t.
- **Latency 100ms - 300ms:** Acceptable for navigation. The model will learn to "predict ahead" slightly.
- **Latency > 300ms:** Problematic. The robot will oscillate.
    - *Fix:* In your dataset loader (LeRobot configuration), you might need to "shift" the labels. You would associate Image_t with Action_{t + k} where k frames corresponds to the latency delay. This aligns the "visual state" with the "action that was actually executing" at that moment.

Summary Checklist for You

1. [ ] **Run the Latency Test Script:** Determine your system lag.
2. [ ] **Setup "Chase" Task:** Use a colored cone/box.
3. [ ] **Record 50 Episodes:** 10-20s each, varying start positions.
4. [ ] **Train SmolVLA:** Feed the data into LeRobot.

This "Action vs Telemetry" report is now ready to be generated with full details.