zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

Field Notes from a Low-Cost Robot Learning Lab: Hardware Churn, Software Whiplash, and What Actually Went Wrong

Iliya Radulov[1]

[1]Independent researcher

October 2026

Abstract

This is the informal companion to two more rigorous papers from the same project. Over roughly seven weeks we trained imitation-learning policies on a self-built SO-101 robot arm, and along the way we moved the work between machines more often than planned, rebuilt the camera setup, fought a fast-moving open-source framework, and lost an embarrassing amount of time to causes that turned out to be a loose power cable, a swapped port, or a missing audio package. The quantified findings are reported elsewhere. Here we tell the rest: what the hardware and software path actually looked like, which incidents cost us the most time and why, what we would warn a newcomer about, and which threads we deliberately left unfinished. We write for people building something similar on a hobbyist budget, not for reviewers, and we try to be as candid about our mistakes as about our results.

1 Introduction

Papers about robot learning usually describe the clean version of the story: a dataset, a policy, a success rate. The version that happens at a desk is messier. Cables come loose, a framework updates and a command that worked on Monday fails on Friday, a camera that works perfectly on its own crashes the moment the arm starts moving, and a policy that trains without a single error turns out to be useless because of something visible in the footage that nobody had looked at.

This paper is the record of that messier version. It belongs to a single-operator project built around the open-source SO-101 arm and the LeRobot framework, which began on 5 August 2026. Two companion papers report the results in the form a journal would expect: a study of trigger-based attacks on robot policies [1] and a findings paper on camera-position artifacts, lighting, and background generalization [2]. Where a story here has a rigorous, quantified counterpart there, we point to it and do not repeat the numbers. What belongs here is everything else: the machines, the versions, the incidents, the dead ends, and the lessons we would pass on.

We have two reasons to write it down. The first is practical. Nearly every problem described below is documented nowhere we could find, and each one cost hours. The second is that we found, while writing the findings paper, that our own memory of what happened and when was unreliable. The only reason we can reconstruct it now is that we kept dated session notes throughout, and we think that habit deserves a paragraph of advocacy by itself (Section 7).

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

2 Hardware: Where We Were Headed, and the Detours on the Way

2.1 The destination

The project was never really about a stationary arm. The goal from the start was a mobile manipulator: first a single-arm mobile platform of the LeKiwi type, and eventually a XLeRobot-class robot with two arms on a moving base. Because we had no previous experience with robot arms, we deliberately started with one: learn to operate a single arm, then a second one, then add mobility, and only then combine the two. The computing plan followed from the destination. A Raspberry Pi 5 was meant to be the onboard computer, since it is small enough to ride on a mobile base, and everything was supposed to run on it from the beginning.

2.2 Starting on a Pi 4

Then reality intervened: there was a market shortage and no Pi 5 to be had. We bought a Raspberry Pi 4 (1 GB) instead, as a stand-in and as an experiment to see whether the whole pipeline would run on a Pi at all. We built and calibrated the arms on a Mac first, so that "is the arm built correctly" was settled separately from "can the Pi handle it", and then moved the setup to the Pi. Teleoperation held 60 Hz there, even with both arms and a webcam running.

Recording was the problem. With arm control, camera capture, and real-time video encoding all running together, the Pi 4 sustained only about 4–6 Hz against a 30 Hz target. A test recording that should have taken 20 seconds took 63, and the encoder processed roughly 570 examples per second against about 7,300 on the Mac, a factor of about twelve. Lowering the resolution and frame rate and switching on streaming encoding did not fix it, and we did not want to keep degrading the data to suit the machine. This was a hardware ceiling, not a configuration mistake.

So we moved recording and training back to the Mac and waited for a Pi 5. We also looked at a Ryzen 7 laptop we had at hand, but its integrated graphics had no practical PyTorch acceleration path, which would have made training CPU-only; the Mac's GPU backend (MPS) was the usable option. Practically all of the August work, including the first autonomous success on 5 August, was done on the Mac.

2.3 The Mac era

A laptop has very few ports. Two arm controllers and two USB cameras all went through external hubs, so everything shared one pool of USB bandwidth, and the symptoms were maddening because they were intermittent and load-dependent. Both cameras worked perfectly when tested one at a time, and then crashed with a "camera too slow" error during a real rollout, when the arm, the cameras, and the policy were all active at once. It took about two hours of swapping ports and cables to establish that the hub itself was not communicating reliably under load. A recording campaign in August showed the same thing from the other side: episodes and the pauses between them were taking far longer than configured, and the cause was that both arms and both webcams had ended up on a single USB 2.0 hub, confirmed with a device listing. Moving them to the available USB 3.1 hub restored normal speed. Even earlier, a camera that kept disconnecting had turned out to be an underpowered hub; a powered one fixed it.

Training speed was the other limit. One 300-episode, 80,000-step run took almost 28 hours of wall-clock time on the Mac's GPU backend. A quick test of CPU-only training measured about 506 seconds per step, which settled that question in thirty steps.

2.4 The Pi 5 arrives

When a Pi 5 finally became available, the USB problem made the move more attractive. It has four native USB ports (two USB 3.0, two USB 2.0), enough for each camera to have a direct port of its own and for the hub to leave the high-bandwidth path. We migrated the SO-101 rig to it in early September, and with it ran the policy that later reached a confirmed 86% success rate on its trained positions [2].

One lesson from that migration surprised us. Cameras need direct ports, but the arm's serial controllers behaved better *behind* a hub, the opposite of our first assumption. Hubs are harmless for low-bandwidth serial traffic and harmful for video, and we had treated all USB devices as one category. The working configuration ended up as: both cameras on direct USB 3.0 ports, both arm controllers on the hub.

The Pi 5 also kept giving us small administrative surprises. It dropped off the network once and needed a monitor and keyboard attached to re-enter the Wi-Fi password locally, after about 30 minutes of checking everything else first. Its IP address drifted between sub-networks. We now treat "re-confirm the address, do not assume it" as a standing rule for every session.

The Pi 5 was fine for recording and for running ACT, but it could not run SmolVLA on its own CPU. Measured, it managed about 0.55 inference steps per second against a 30 Hz control target, roughly 54 times too slow. Strangely, the task still completed, and the run went unnoticed for about an hour before we saw the numbers. Inference therefore moved to the desktop, and the Pi became a thin client that captures cameras and drives the arm at about 5–6% CPU. Making that split fast enough, with asynchronous inference and real-time action chunking, is the subject of a later paper, so we only note it here (Section 4).

2.5 The voice-trigger setup

The Pi 4 did not disappear. One Pi 4, called `voice`, was kept for the voice side of the project. The design was a two-computer arrangement: one machine listens, the other operates. Speech recognition (Vosk) and speech synthesis (Piper) run on the listening side, a small network service built on Flask carries the trigger to the machine hosting the arm, and that machine starts the rollout. This first stage reached its goal of a voice-triggered autonomous rollout, and the same trigger path is meant to be reused later for a language-conditioned policy. It is the seed of the voice-and-vision robot we eventually want on the mobile platform.

2.6 Renting a GPU, then buying one

Training a small vision-and-state policy like ACT on a Mac is tolerable. Training a vision-language-action model such as SmolVLA is not. Before spending money on hardware we rented an RTX A6000 cloud GPU for roughly $0.54 per hour and used it to validate the whole SmolVLA pipeline end to end with a 100-step test whose only purpose was to prove the plumbing worked. That exercise cost somewhere between $0.50 and $0.70 and told us two useful things: the pipeline worked, and SmolVLA is dramatically heavier than ACT (on the order of 27–38 seconds per step against 1–2.5 for ACT in our setup), partly because automatic mixed precision was not enabled by default and our batch size was smaller than it should have been.

With that evidence, we decided to buy a local training machine rather than keep renting: a Ryzen 7 9700X with an RTX 5060 Ti (16 GB) and 32 GB of RAM, for about 1,750 euros. (The board and CPU were later upgraded to a Ryzen 9 9900X; what follows describes the machine as originally purchased.) The practical effect was immediate. The 96-episode re-recording of the

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

camera-fixed dataset trained for 50,000 steps in about an hour and 40 minutes, at roughly 8 steps per second, against about one step per second on the Mac.

2.7 What comes later

The mobile platform itself is a separate chapter. It will get its own, more capable Raspberry Pi 5, distinct from the one that runs the SO-101 rig, so that the two setups do not share machines or get tangled together. Its story, and the work of making a language-conditioned policy run well on it, belong to later papers, and we deliberately keep them out of this one.

3 Software: A Stack That Moved Under Us

LeRobot is a young, fast-moving framework, and we were glad of that: new features arrived constantly. The cost is that, in our experience, two machines installing it a few days apart could behave differently. We list the traps that cost us the most time.

Python version. The framework needs Python 3.12 specifically. 3.11 and 3.14 both caused trouble, and on 3.14 the configuration library `draccus` breaks outright.

Install from the repository, not from PyPI. Installing an extras package directly from PyPI did not give us what the repository contained. The reliable route was always an editable install from a local checkout (`pip install -e ".[extras]"`).

The wrong interpreter. One merge script failed with a missing `draccus` dependency, which sounded like an installation problem. The actual cause was that the script was being run with the system Python instead of the virtual environment's interpreter. Check `which python` before debugging anything else.

The schema bug we hit more than once. Dataset merge scripts repeatedly crashed with `KeyError: 'dtype'`. The reason is that the policy configuration and the dataset metadata both describe features, but in different formats, and it is easy to hand the wrong one to `LeRobotDataset.create()`. The same mistake appeared in the background-diversity merge and again in the merge of the final 96-episode dataset. Two other bugs in that last script were equally mundane: one of the three recording batches was missing from the list, and the task string said "box" where the recorded data said "target".

Crashes while encoding episodes. Early recordings sometimes died with a `BrokenProcessPool` error, because the default mode holds a whole episode in memory before encoding it in one go and the resulting spike was too big. Four recording flags fixed it (streaming encoding on, two encoder threads, no image-writer processes, two image-writer threads per camera), and from then on we used them in every recording command. We forgot them once, for the 120-episode dataset that later turned out to be unusable, and while we never proved that the omission contributed to that failure, we now think of those flags as part of the recording command rather than an option.

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

Small behaviors that bite. Resuming a recording requires passing the dataset root explicitly. The recording tool appends a timestamp to the folder name even when you give it a clean one, so scripts that guess folder names fail. And the rollout tool does not read the policy's input and output features from the checkpoint; they have to be passed on the command line to match the checkpoint's own configuration. The camera-name mapping used at training time is likewise not stored in the checkpoint, so the rename has to be repeated at rollout. A policy that expects three camera slots will not start with two; we filled the third with the Mac's built-in webcam, and only later learned that the third slot is not normalized and the stand-in was unnecessary. Copying a directory with `scp` when the destination already exists nests it one level deeper, so the files land where nothing looks for them. And after a folder reorganization, a policy's tokenizer path still pointed at the old location; a `.bak` file next to the patched script proved only that a fix had been applied once, not that the path in it was still valid.

The takeaway. None of these is a bug report. They are what building on a moving target looks like. What helped was writing the exact working command down every time, and keeping a short "environment facts" note that we read at the start of each session.

4 War Stories

Each of the following looked serious at the time. Each had a mundane cause. We tell them in the order they happened.

4.1 The small stuff from the first weeks

Before the dramatic incidents there was a long tail of small ones, each worth a sentence because each is easy to hit. An installation failed with "no space left on device" although the disk was nearly empty: on the Pi, pip's temporary directory sits in a small RAM-backed filesystem that the large PyTorch download did not fit into, and pointing the temporary directory at the main disk fixed it. A camera scan listed six video devices although only one camera was attached; the other five were the Pi's own internal camera pipeline, and listing the devices with the Video4Linux and USB tools identified the real one. Serial port names such as `/dev/ttyUSB0` and `/dev/ttyUSB1` are assigned by the operating system and can swap after a reconnect; stable identifiers under `/dev/serial/by-path` do not. The very first recording ran, saved, and contained no arm movement at all, which we put down to nerves, and a second one came out blurred because the room was dim and the camera's shutter had slowed to compensate, a first hint that lighting would matter more than we expected. On the Mac, a run died with "device not configured" because the hub was bumped while the script was running; the lesson, which we then had to relearn, is to touch cables only between runs. And a camera scan on the Mac failed because the operating system had not yet been told that the terminal was allowed to use the camera.

4.2 The arm that connected but would not move

The follower arm powered up, the software connected to it without complaint, and nothing moved. We checked ports, calibration, and cables for a long time. The cause was that the servo bus needs its own power supply: the arm was drawing only what the USB port could give, which is not enough to move it. The supply was an old laptop adapter, and its cable was pushed into the adapter but slightly withdrawn, so that it looked connected and was not. Everything was mechanically stable and nothing was visibly wrong; only testing each element in turn found it. The software reported

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

no error because, from its point of view, nothing was wrong. We later saw the same pattern again after a cable-management session during the Pi 5 migration: "no error in the terminal" does not mean "no problem", and the power supply is the first thing to check.

4.3 Overload errors on shutdown

Several times the arm printed a motor overload error as we shut it down. It looked like a hardware fault. It was gravity: at the instant torque is released, each joint carries the weight of the arm above it, and the controller reports that as an overload. It is benign. (An overload error *during* a rollout, after the arm has contacted something, is a different matter, and in the poisoning study we excluded two trials for exactly that reason.)

4.4 The swapped ports

On 20 September, during a teleoperation test, the leader and follower controller ports were accidentally reversed relative to the confirmed mapping. The arm moved erratically. We cut power immediately and then inspected the arm by hand; no damage. The corrected command ran cleanly. A device-path mapping that you "know" is correct is the first thing to re-verify at the start of a session, because device numbers are not stable across reboots and re-plugging.

4.5 The audio cue that cost an afternoon

We wanted the machine to speak episode boundaries aloud during recording. The speech command failed silently. Tracing it took us through a missing speech service, a missing audio backend, and the discovery that the Pi 5 exposes two independent HDMI audio devices of which only one was connected. An audio-configuration file pointing at the right card fixed part of it. The episode-boundary cue was still not working on 20 September, and recording simply proceeded with terminal text instead. We note it here as unresolved and, in the end, not worth more time.

4.6 The flag that looked right and was not

A recording command that had worked before failed after we pasted it from a note. It looked identical to the working one. The cause was an invisible Unicode character that had come along with the pasted text and sat inside one flag, so the flag was silently not recognized as the one we meant. Retyping it by hand fixed it. We never found out whether the culprit was the copy and paste itself or our German keyboard layout. We now paste commands through a plain-text editor, and when a command misbehaves for no visible reason we suspect the characters first.

4.7 Making a vision-language-action model run at a usable speed

Running SmolVLA remotely, over the network from a small onboard computer, initially produced a control rate far below what the policy needs. The fix was LeRobot's asynchronous inference mode, which separates action-chunk prediction from action execution, combined with tuning the number of actions per chunk, a threshold for requesting the next chunk, and how overlapping chunks are blended. Adopting the official asynchronous mode gave an effective rate of about 6 Hz; further tuning traded rate against motion quality and did not by itself remove the instability. It is a real improvement and it is also still imperfect. This is the part of the project that remains most clearly unfinished, and the work of closing that gap (including real-time chunking) belongs to a later paper, so we leave the details there.

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

5 The Camera Story, Told Straight

If we had to pick one lesson from the whole project, it would come from here. The quantified version of this story is in the findings paper [2]. This is the version with the mistakes in it.

5.1 The cheap cameras that did their job

We started with ordinary, inexpensive USB webcams of low resolution (Trust Spotlight webcams, advertised at 1.3 megapixels, although the listing gives 640×480 as the hardware resolution and the Mac showed about 0.7). We chose them for the least scientific reasons possible: they were cheap and could be delivered quickly, and as complete beginners we did not know what else to look for. In hindsight they were not the wrong cameras in the sense that mattered. They produced perfectly usable images for what we asked of them, and we have no complaint about their quality. We had also considered building the vision into the Raspberry Pi itself with a dedicated AI camera, but that was far more camera than the task needed, and it is still unused.

Our first model, trained with only a single overview camera, reached toward the object but never grasped it. The reason, found by reviewing the footage, was simple: the camera never showed the gripper closing on the object, so the model had no visual signal connecting what a grasp looks like with the act of closing the gripper. A second, close-up camera at the gripper solved that.

5.2 Light we did not know we were using

The real problems were not the cameras but where we put them and what we pointed them at, and above all the light. Having no experience, we did not appreciate how much the light falling on the scene mattered. The first recordings were made in summer, in daylight, and the room was bright enough. Nothing about those recordings was controlled. They were simply made at different times of day, with the sun in different places, and the amount of light reaching the scene was never the same twice. We did not notice because nothing went wrong, until the days got shorter and the light changed, and suddenly it did.

5.3 The better cameras, and what they quietly changed

We had always intended to replace the first cameras with better ones. The new ones had a delivery time, and we did not want to wait two weeks, so we started with the simple ones and swapped later. That is where we made our biggest mistake: when the new cameras arrived, we mounted them and did not stop to think about what the change would actually cause.

Quite a lot changed at once, and some of it was a surprise. The replacements were InnoMaker USB cameras: 1080p with a high-dynamic-range sensor for the main view, and 720p for the wrist. They were far more sensitive to light than the old ones. Both also had a much wider angle of view, around 120 degrees diagonally (the old cameras do not even list one), so a great deal more of the room was now in the frame. Because we had bought a 3D printer in the meantime, we could print our own camera supports and, importantly, the mount that fixes a camera to the arm, which had been a weak point. The camera must sit rigidly on the gripper, stay out of the gripper's way, not hide the sample, and not hit anything as the arm and camera move together. We had not appreciated how many separate requirements there were until we tried to meet them.

The first episodes we recorded at the intended camera position were useless. We were recording half of the room, and our own arm and hand, moving through the scene during teleoperation, were in many frames. The policy that trained on this data converged normally and did nothing useful.

Only by watching the footage did we see why, and a repositioned camera fixed it (the numbers are in the findings paper).

The high position was not an accident. It was the official XLeRobot camera design, which we had adopted on purpose so that data and policies could later transfer between the arm and that robot. The transfer turned out to be unnecessary, since the mobile robots run SmolVLA on their own recordings. We had optimized the camera for a robot we did not yet have, and it gave unusable data for the one we did.

5.4 What nobody writes down

The part that still surprises us is that none of this is explained anywhere we looked. The documentation and the community seem to assume that you already know how to place a camera for a learning system, and that it is a solved, boring step. We made the same assumption in reverse. We bought the recommended cameras, used the recommended mounts, and expected it to be plug and play. It is not. Where a camera sits, how wide it sees, how much light reaches it, and what else appears in the frame are all design decisions that shape the dataset, and a policy can only learn from what the camera shows it. If you build one of these robots, plan to spend real time on this step, and look at your own footage before you train.

6 Dead Ends and Deliberately Deferred Threads

6.1 The orientation mystery that was never solved

A persistent weakness in one specific approach orientation (what we called O1, 0 degrees relative to the arm) showed up consistently across an entire lineage of ACT checkpoints, even as other, unrelated weaknesses in that same lineage were successfully fixed one by one. Live observation during O1 approaches showed the gripper visibly rotating slightly during descent, plausibly linked to the orientation's weaker performance. Four separate theories were tested in turn: kinematic strain specific to that approach angle, camera occlusion, wrist-roll calibration wraparound (checked directly against recorded joint values — no wraparound boundary was anywhere near the observed range, ruling this one out cleanly), and a position-correlated version of the rotation itself (the wobble appeared at some positions regardless of whether they were strong or weak performers, which did not cleanly track the O1 pattern either). None of the four produced a single, unifying explanation. The most plausible remaining candidate is plain grasp geometry: the sample we used at the time was elongated, so the gripper had more graspable length when it approached along the long side than at other angles, which would make some approach angles inherently less forgiving than others. This rests on memory rather than on the records, and we never confirmed it.

We stopped there on purpose, and the reason is worth stating honestly. The ACT lineage was never meant to be the destination; it was the stepping stone towards a vision-language-action policy (SmolVLA). The theories that were cheap to test turned out not to be related, or at least not directly related, to the result, and pushing further would have meant investing weeks in a policy we were about to leave. The investigation ended in mid-August, on the same day as the white-background finding, with SmolVLA already planned. The camera changes came afterwards. By late August a diversity report noted the arm filling much of the frame, and in mid-September the main camera moved from a position above the workspace to one in front of the arm, because from above the arm itself was the obstacle: the camera saw mostly the arm and often could not see the sample at all. Around the same time we began producing samples of different geometries, which suits a language-conditioned policy better than a single elongated object. Those later changes

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

would in any case have reshaped the orientation question. We therefore report O1 as it stands: tested, not explained, and deliberately left behind rather than solved.

6.2 The text-trigger backdoor that did not work

Summarized in full in the companion paper [1]: an attempted reproduction of a published text-triggered attack on ACT initially appeared to succeed, then contradicted itself under a more controlled setup. The actual cause, found via source-code inspection, was that ACT has no language-conditioning pathway at all — the apparent effect was object-placement variability and contradictory training labels, not the trigger. The negative result mattered because it is exactly the kind of failure that is cheap to catch early (a few minutes reading a `forward()` method) and expensive to discover after a week of confused experiments, which is how it actually went for us.

6.3 Simulation and visualization, repeatedly postponed

Two tools kept appearing on our list and kept being pushed down it. The first is simulation. A MuJoCo model of the arm was confirmed to run in late September, and domain randomization is the obvious next step, but the gaps between simulation and our real setup (action space, camera resolution, the variety of tasks) are large enough that we chose to keep the real robot simple and stable first. The second is visualization. After moving back to the Mac and installing a newer version of the framework, we found that it integrates the Rerun viewer, which shows recordings and rollouts side by side and has turned out to be very useful for judging how different policies actually behave. Both belong with the policy work of the next paper rather than here, and we mention them so that their absence from this one is not mistaken for oversight.

7 Lessons, Informally

The practitioner checklist in the Discussion section of the findings paper has eight items, written in the dry, diagnostic voice a methods section demands. Here is the same list again, in the voice it actually happened in.

Watch the footage. We mean it. The zero-success camera-artifact policy cost a full recording session and a training run before anyone actually sat down and watched the raw episodes play back. Ten minutes of watching would have shown the problem in the first thirty seconds. A smooth loss curve tells you the model learned *something* consistently; it tells you nothing about whether that something was the task.

Build for the robot in front of you, not the one you are planning to build next. The high camera mount existed for a platform that did not exist yet, running a policy family we had not started using. It cost us a full dataset and a full training run to realize the hypothetical future requirement was solving a problem we did not actually have yet.

Check the USB tree, the camera brightness, and the port mapping before you record, every single time. Three separate incidents — a saturated hub, a camera that had not finished adjusting exposure after a background swap, and leader/follower ports reversed — all cost real time, and all would have been caught by a thirty-second check before starting.

Lighting is not an afterthought you remember later. We went weeks describing lighting qualitatively ("it's fine", "it's a bit dim") before finally pointing a lux meter at the workspace. The moment we had actual numbers, two previously-confusing failure patterns separated cleanly into two different, oppositely-signed problems.

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

Pick one test object and stick with it until you have a reason not to. Object shape and color were a live confound for weeks before we fixed on one sample.

Say what actually happened, not just whether it worked. "Failed" is not a result. "Failed to find the sample", "found it but missed", "grabbed it and then let go", and "did something that looked nothing like the task" are four different problems with four different fixes, and lumping them into one failure count hides that.

One trial is a story, not a result. Numbers moved meaningfully between a 1-trial pass and a 7-trial pass at the same positions. We now treat anything under five trials as provisional, no matter how clean it looks.

When something is unexplained, save the evidence, not just the conclusion. The position-4 anomaly is still open partly because we never saved the one raw frame that would have settled whether the sample was invisible to the camera or visible and ignored.

A ninth lesson did not make the findings paper's formal checklist, because it is not a finding — it is how we were able to write any of this down accurately at all: *keep dated session notes, every session, as you go.* This paper exists because of that habit, not despite the gaps in it. Reconstructing this project's actual chronology from memory alone, seven weeks later, would have been hopeless — we discovered exactly that while assembling this document, when two different past records disagreed with each other about a hardware spec, and the only way to resolve it was checking a third, dated note that happened to capture the moment the hardware was later upgraded. Neither record was wrong; they were just taken at different points in time, and without timestamps, there was no way to tell which was which. The habit is boring and it is the only reason any of the rest of this paper is trustworthy.

8 What Comes Next

The long-term goal is a mobile, eventually bimanual robot: first the LeKiwi base, then the larger XLeRobot. The stationary SO-101 arm stays as it is. It becomes a standalone training playground, where policies, sensors and simulation are tried out before anything is moved to a mobile base, so that the Kiwi serves as a proof of concept on the way to XLeRobot rather than a place to debug basics. The mobile platform will have its own Raspberry Pi 5, kept separate from the one running the arm, so that the two setups do not get tangled into each other's debugging sessions. Its story belongs to later papers.

The more active thread right now is making a vision-language-action policy (SmolVLA) usable in real time. Switching to it was planned as a quick step and proved to be a larger project. The first SmolVLA training run was in late September, and the policy went from zero to roughly 50–60% success on the trained positions, which we count as a real result. It is not yet as stable as ACT; but ACT started at full success and then hit a wall, whereas SmolVLA started from nothing and keeps improving. Inference now runs on the local training desktop, with the policy served over the network to the Pi driving the arm. Closing the gap between "works" and "works well" is substantial enough to be its own paper, and we stop here.

Code and Data Availability

Command templates and merge scripts are given in the appendices of the companion papers [2].

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697

References

[1] I. Radulov. *When Does a Trigger Reach a Robot Policy? Architecture, Fine-Tuning, and a Correctly Targeted Visual Backdoor.* Zenodo, 2026 (all versions). https://doi.org/10.5281/zenodo.23143960

[2] I. Radulov. *Distribution Mismatch, Lighting, and Background Generalization in a Low-Cost Imitation-Learning Pick-and-Place Policy.* Zenodo, 2026. https://doi.org/10.5281/zenodo.23212210

zenodo:23259697 [cs.RO] Oct 2026 DOI: 10.5281/zenodo.23259697