Preprints.org

Article

Not peer-reviewed version

An Open-World Predictive Architecture Beside a Vision-Language-Action Model: Declared Rules, Prediction and Repair for Robot Manipulation

Jean-Jacques Dubray *

Posted Date: 3 November 2026

Keywords: open-world predictive architecture; vision-language-action models; world models; declared world model; robot manipulation; LIBERO; large language models; runtime verification

Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC, OpenAlex.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Article

An Open-World Predictive Architecture Beside a Vision-Language-Action Model: Declared Rules, Prediction and Repair for Robot Manipulation

Jean-Jacques Dubray

Cognitive Fab LLC, USA; jdubray@gmail.com

Abstract

Vision-language-action models (VLAs) turn camera images and an instruction into robot motions. They keep no explicit model of the world: they cannot say what a step should achieve, notice that it did not, or be held to a rule. We place an open-world predictive architecture (OWPA) beside an unmodified VLA. A language model writes a declared world model of the task as readable rules; a deterministic engine runs it to plan and to predict the facts each step should produce; a monitor compares prediction and scene after every action chunk, re-instructs a stalled policy and repairs rules that prove wrong. We know of no earlier system that pairs a VLA with a world model written by a language model and repaired at run time. Tested with two open VLAs on LIBERO, in nominal scenes and in scenes changed during the episode, it gives three findings. The declared model is a reliable observer: it notices most failures early, names the fact that did not change, and costs nothing when the scene is as trained. Correction through language does not follow: better instructions, a repaired or perfect model, or a correct plan leave success unchanged, because the instruction selects among learned behaviours and does not compose new ones. A small fine-tune on demonstrations of the changed scenes is the only intervention that moved success. Detection and correction are separate problems; a declared world model solves the first and finds the data for the second.

Keywords: open-world predictive architecture; vision-language-action models; world models; declared world model; robot manipulation; LIBERO; large language models; runtime verification

---

1. Introduction

A vision-language-action model (VLA) maps camera images and an instruction to robot motions [6–9]. Trained on demonstrations, the best open VLAs complete most of the long kitchen tasks of the LIBERO benchmark [11]. What they do not have is an explicit model of the world. A VLA does not state what it expects a step to achieve, so it cannot notice that the step failed; it cannot be told a rule ("nothing on a lit stove") and be held to it; and when the world changes in a way the demonstrations never showed, nothing in it can be corrected except by retraining.

Our previous papers built and tested an open-world predictive architecture (OWPA) [2–4]. Perception supplies facts. A language model writes a declared world model: the rules of the scene, as text a person can read. A deterministic engine runs the model to predict. When a prediction is confirmed wrong, the language model repairs the rules, and the repair is kept. The architecture follows the modules of LeCun's proposal for autonomous intelligence [1] (perception, world model, actor, critic) with one difference: its world model is declared, not learned. Terra Incognita II [4] ran it on video in a grid world, and its future work proposed to place the same architecture beside a VLA. This paper does so.

The division of labour is the one robotics has long used between a planner and a controller. The VLA keeps the motor control: it receives an instruction and produces the motion. The OWPA works above it. It writes the task's rules from the instruction and the first scene, plans a sequence

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

of instructions, predicts the facts each step should produce, compares them with what the simulator reports, and acts on the difference: a shorter instruction to a stalled policy, a repair of a rule that was wrong, a veto of an action that breaks a declared constraint. The VLA's weights are never changed.

The question is what such a layer can do for a VLA in scenes its demonstrations never contained: notice its failures, correct them, or both. Its corrections are compared with the usual answer to a failure, more demonstrations.

The contributions are:

- a test bed in which two open VLAs, Pi0.5 and SmolVLA, reproduce their published LIBERO-10 success rates, with the simulator's state read as symbolic facts after every action chunk;
- a language model that writes the declared model of each task from its instruction and first scene, with a valid plan for 99 of 100 scenes;
- the observation that a check of a written model against itself cannot find a rule that is wrong about the world: only acting can;
- a declared monitor that flags most failures of the VLA in changed scenes, with more than half the episode left, and costs nothing in the nominal ones;
- a comparison of eight arms that separates detection from correction: no correction through language raises success in the changed scenes, and a small fine-tune on demonstrations of them is the only intervention that moved it.

2. Related Work

Vision-language-action models.

RT-2 [6] showed that a vision-language model fine-tuned on robot data can output actions. π_0 [7] and $\pi_{0.5}$ [8] generate chunks of continuous actions with a flow-matching head; SmolVLA [9] does the same with a model small enough for one consumer GPU. LeRobot [10] publishes checkpoints of both fine-tuned on LIBERO. None of these models keeps an explicit model of the world: what they expect is implicit in their weights.

Learned world models that check a VLA.

CheckVLA [16] predicts the effect of an action chunk with an action-conditioned world model built on V-JEPA 2 features [14] and asks the VLA to rewrite the chunk when the observed effect drifts from the prediction. FOREWARN [17] steers a policy with a latent world model and a vision-language model; Pre-VLA [18] scores candidate chunks with a learned verifier before they run. These world models are learned: they detect a drift but do not state its cause, and what they learn cannot be read or edited.

Physics simulators in the loop.

SIMPACT [15] builds a physics simulation from one RGB-D image and lets a vision-language model refine its plans by rollout. The physics is given; what is wrong in the simulation stays wrong unless it is observed again.

Symbolic planners above learned skills, and world models written as code.

Hierarchical systems such as τ_0-VLA [19] propose subtasks to a low-level policy, with both levels learned. LODGE [22] has a language model write hierarchical planning domains and refine them through grounded execution in the environment. WorldCoder [21] writes the world model as a program and revises it from experience in grid worlds. WALL-E 2.0 [20] is the closest on the repair mechanism: a language model compares predicted and observed trajectories and writes rules that correct its predictions, in text environments without a robot policy. Language models exploring physics simulations in which one law was changed, with no task and no hint, rarely discover it: one of twelve models succeeded in 16.7% of runs and the others never did, often observing the anomaly and continuing their plan [23]. That benchmark scores a law as correct and as supported by experiment

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

separately; the repairs in this study are counted, not scored that way. A survey of world action models [24] classifies rollout, joint generation and prediction heads; it does not treat declared world models as a category.

A search of 2024 to 2026 work found no system that pairs a VLA with a world model whose rules are written by a language model and repaired at run time from confirmed prediction mismatches. The parts exist separately; what this work adds is their combination beside an unmodified VLA, with declared vetoes, measured on perturbations the demonstrations never contained.

3. The Architecture Beside a VLA

Figure 1 shows the loop. It has five steps.

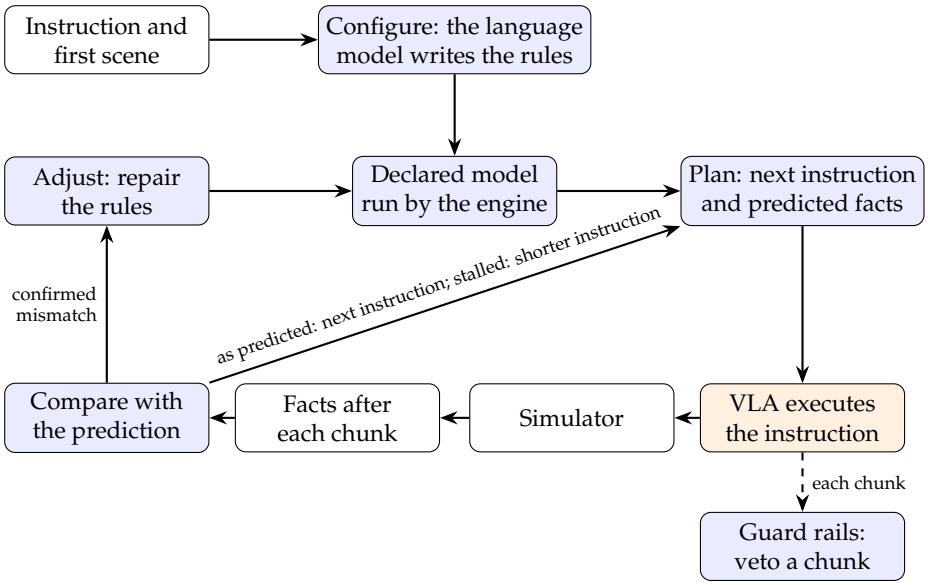

Figure 1. The OWPA beside a VLA. Blue: the declared layer; orange: the unmodified VLA. The VLA does the motor control; the declared layer plans, predicts, compares, repairs and vetoes.

1. **Configure.** A language model writes the task's model from the instruction and the facts of the first scene: the goal, one rule per instruction (what it needs and what it changes) and a plan. It does not see the benchmark's goal definition.
2. **Plan.** The engine searches for a sequence of instructions that reaches the goal under the written rules and predicts the facts after each one.
3. **Execute and compare.** The VLA runs one instruction. After every chunk of 10 actions the facts are read and compared with the prediction.
4. **Act on the difference.** A stall leads to a shorter instruction (the sub-goal of the current plan step). A confirmed mismatch (a prediction false for two readings in a row) calls Adjust, which repairs the rules; the planner then replans from the current facts.
5. **Guard rails.** Declared constraints are checked before each chunk runs; a chunk that would break one is stopped.

Figure 2 shows a simplified declared model for one task, and how it reads a scene in which one step is already done.

The order of the responses is declared: a stall never rewrites a rule, and only a confirmed mismatch does. This keeps the rules from drifting when the policy is merely slow.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

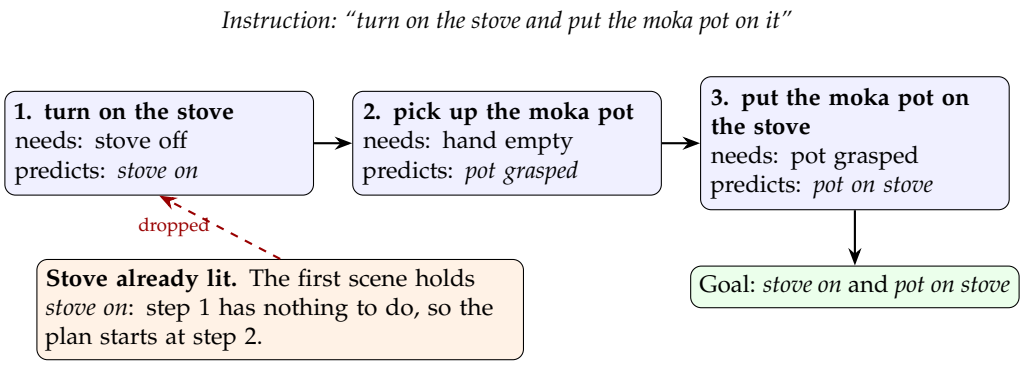

Figure 2. A simplified declared model for task 2. Each plan step is a sentence sent to the VLA, with what it needs and the facts it should produce. The monitor checks these facts after every chunk; a step whose facts do not appear is a stall or a failed prediction.

4. The Test Bed

Tasks.

LIBERO-10 [11] holds ten long kitchen tasks, each a composition of two or three short ones: put two items in a basket, turn on the stove and put the moka pot on it, put a bowl in a drawer and close it, put a mug in the microwave and close it. The simulator is robosuite [12] on MuJoCo [13], run through LeRobot [10]. An episode ends at success or at 520 steps.

Two VLAs.

Pi0.5 [8] (about 3 billion parameters) is the primary VLA and SmolVLA [9] (450 million) the second, both with the LIBERO checkpoints published by LeRobot. Both emit a chunk of 10 actions per call (the first 10 of the 50 that Pi0.5 predicts) and run synchronously: the simulator waits for each chunk, and a new instruction applies from the first action of the next chunk. The LeRobot port of Pi0.5 conditions its action expert directly on the instruction; it does not run the high-level inference of the original system, which first predicts a sub-task in language. In this study the declared plan takes that role. Each runs at the observation size of its reproduction (Table 1).

Reproduction gate.

No arm runs until arm A, the VLA alone, reproduces the published LIBERO-10 rate within its 95% Wilson interval [28]. Both pass (Table 1). The two VLAs differ where it matters for this work: Pi0.5 fails 4 episodes in 100 and leaves little to rescue in the nominal worlds, so its comparison rests on the perturbed worlds; SmolVLA fails 33 and shows the effect on the nominal tasks.

Table 1. Arm A, the VLA alone, on LIBERO-10: successes in 10 episodes per task, and the suite rate with its 95% Wilson interval. Pi0.5 at the LeRobot default observation size, SmolVLA at 256 pixels, both with chunks of 10 actions.

Task	0	1	2	3	4	5	6	7	8	9	Suite	Published
Pi0.5	10	10	10	10	9	10	10	10	7	10	96 (0.90 to 0.98)	96.0
SmolVLA	2	9	9	9	2	8	5	9	5	9	67 (0.57 to 0.75)	71

Compute.

The VLAs and the simulator run on NVIDIA T4 GPUs: the reproductions and pilots on Kaggle, the main runs on Amazon Web Services (g4dn instances), with the Kaggle runs of the same cells kept as a replication. Before any main run on the second platform, one cell (Pi0.5 alone, the task with the most failures, 10 initial states) was run on both: 8 against 7 successes, mean steps within 2.7% on the episodes both completed, the same outcome in 7 of 10 episodes. The platform is recorded for every cell. The engine, the monitor and the scoring run on a CPU. The simulator gives bit-identical states on

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

two different CPU machines for the same actions (954 steps on four tasks), so CPU-side simulation and the GPU runs can be compared.

5. Perception: Facts from the Simulator State

In this first paper perception reads the simulator's true state, as in the oracle condition of our earlier work [2]; reading the facts from the camera is the step taken in Terra Incognita II [4] and left for a later paper here. The state is turned into a fixed vocabulary of facts after every chunk: *in*, *on*, *open*, *closed*, *grasped*, *lifted*, *stove on*, *stove off*, and two facts of the monitor's own.

Facts LIBERO provides and facts it does not.

LIBERO's own predicates give *in*, *on*, *up* and *turned on*. Its *turned off* reads false at the stove's rest position, so "off" is read as "not on". It has no *grasped*: the simulator's own grasp test misses 22% of the frames in which an object is carried. Combined with a second test (fingers closed, the object at least 4 cm above its rest height and within 10 cm of the gripper), it misses 4%, with no false grasp while the gripper is open.

Ajar.

LIBERO reads a drawer as closed only when it is fully shut and as open only when it is nearly fully out. Anything in between is neither. The monitor names this state *ajar* when the drawer or door is one the goal must open or close. A drawer at rest ajar is a failure the benchmark cannot name: the task looks nearly done and is not.

No progress.

No progress holds when none of the progress facts has changed for a fixed number of chunks. The threshold was set on the SmolVLA episodes: it raises 2 false alarms in 67 successes and detects 32 of 33 failures at some point.

Validation.

On 48 scenes labelled by hand from the saved camera frames, the extracted facts agreed with the labels on 59 of 64 facts for two tasks and 36 of 38 for the other eight; most disagreements were labelling errors or drawers between open and closed. The labeller and the author of the extractor were the same model, and most labelled facts were negatives.

6. Configure: the Written Model

Claude Opus 5.5 wrote the goal, the rules and the plan of each task from the instruction and the facts of the first scene, three times per scene, for the ten tasks and for the moved and obstacle worlds of two of them (42 models). Each model was scored against the oracle rules derived from the benchmark's goal definition (Table 2).

Table 2. Models written by the language model on the scenes used to develop the writer, scored against the oracle. Every valid plan was also the shortest.

Scenes	Models	Goal exact	All rules as the oracle	Plan valid under the oracle
Nominal (10 tasks)	30	24	17	29
Obstacle (2 tasks)	6	6	6	6
Moved (2 tasks)	6	6	4	4

For the main runs the model of each episode was written once, from the first scene of each of the 100 episodes (10 tasks, 10 initial states each), before any run. The goal matched the oracle in 83 of 100, the rules in 62 of 100, and the plan was valid and shortest in 99 of 100.

The plans are good; the rules differ from the oracle's for three reasons, each repeatable:

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

- **An extra precondition.** The writer requires a container to be open before something is placed in it. That is true of the drawer and the microwave, which start open, and false of the basket, which has no door. In a few samples it adds an "open the basket" step, and the plan fails under the oracle.
- **A different reading of the goal.** "Put the pudding to the right of the plate" is written as *in* the region right of the plate; the benchmark checks *on*.
- **A goal the instruction does not state.** "Put both moka pots on the stove" does not say that the stove must be on, and the benchmark requires it. Here the instruction says less than the benchmark checks; a person reading it would miss the same fact.

The engine's check and its limit.

Before anything runs, the engine executes the written plan under the written rules; a plan that does not reach the written goal is returned to the writer with the engine's message, up to three times. The check flagged none of the 42 models, and none of 84 models written in two runs, although 8 of them have a plan the oracle rejects. Each wrong model is consistent with itself: under a rule that the basket must be opened, "open the basket" is a sound step. A check of the model against itself can catch a model that contradicts itself; it cannot catch a rule that is wrong about the world. That error shows only when the step is tried and the prediction fails, which is what Adjust is for. The writer prompt was left unchanged, so that the run-time repair of this rule is measured in the main runs.

7. The Monitor on Recorded Episodes

Before any live run, the monitor was replayed on the 200 recorded arm A episodes (Table 3). The replay shows when the declared layer would have spoken and what it would have done; it cannot show whether a shorter instruction would have rescued the episode, because the recorded VLAs acted on the full instruction throughout. The detectors were set on these runs, so the main runs use new seeds and initial states.

Table 3. The declared monitor replayed on the recorded arm A episodes.

	Pi0.5	SmolVLA
Failures with an event	4 of 4	32 of 33
First event: no progress / ajar / failed prediction	2 / 0 / 2	26 / 1 / 5
Response: shorter instruction / repair	2 / 2	27 / 5
Median steps left after the first event	180	240
False alarms in successes	2 of 96	2 of 67

Most SmolVLA failures are a policy that never starts a step or freezes early, which calls for a shorter instruction, not a change of rules. Two of the four Pi0.5 failures put one moka pot on the stove and never go for the second; no progress fires only near the step limit there, so an idle-arm rule (the arm nearly still for a fixed time while a plan step is undone) was declared, with its acceptance test fixed in advance: at most 5% false alarms in successes, and firing before no progress in at least half of the failures both detect. On the 150 arm A episodes of the main runs it raised no false alarm in 120 successes and fired first in 13 of 16 such failures, a median of 130 steps earlier. It was adopted for arm B.

The vocabulary of shorter instructions.

Both checkpoints were fine-tuned on the same 40 task sentences. Only 7 of the 35 plan steps of the ten tasks have a trained sentence of their own; for the others the declared layer can re-issue the full instruction or a new sentence built on a trained pattern ("put the moka pot on the stove"). Whether each VLA follows such sentences was measured before the main runs: 8 sub-goals, 5 trained and 3 composed, 3 episodes each, an episode counting as followed when the sub-goal's facts hold. Pi0.5 followed 23 of 24 (Wilson 0.80 to 0.99), composed sentences included. SmolVLA followed 5 of 24 (0.09 to 0.41) and none of the trained sentences, which it had learned in other scenes; the measure mixes

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

non-compliance with SmolVLA's ordinary failure rate on these tasks. The re-instruction vocabulary was therefore fixed per VLA before the main runs: sub-goals for Pi0.5, the full instruction only for SmolVLA.

8. Arms and the Comparison

Arm G, fine-tuning on demonstrations.

The usual answer to a failure is more demonstrations. Pi0.5 is fine-tuned on up to five successful episodes per perturbed world, taken from its own scripted-plan runs (arm C) on initial states no other run uses: 19 episodes, 6,441 frames (1 for the step already done, 4 or 5 for the others). The vision-language backbone is frozen and the action expert is trained (693 million of 4.1 billion parameters) for about five passes over the frames, with the optimiser and schedule of the published fine-tuning; these settings were fixed on a probe before any evaluation. The model is evaluated on the main-run states and on nominal tasks 0 and 4, whose perturbed versions supplied the demonstrations, to check that the new skill does not cost an old one.

Table 4. The arms. Figure 3 shows where each acts on the VLA.

Arm	What runs	What it tests
A	the VLA alone	the policy as published
B	the VLA with the OWPA beside it	this work
B^{o}	B with the oracle rules	the cost of writing the rules
B^{-}	B without repair (monitor and shorter instructions only)	what repair adds
G	the VLA fine-tuned on a few demonstrations per perturbed world	the usual fix: more data
C	the VLA with a scripted plan of instructions	a correct plan without prediction or repair
B2	B, re-instructing with the joined pick-and-place sentence of the current step	a better sentence on a stall
C2	C with the joined sentences from the first step	the right words from the start

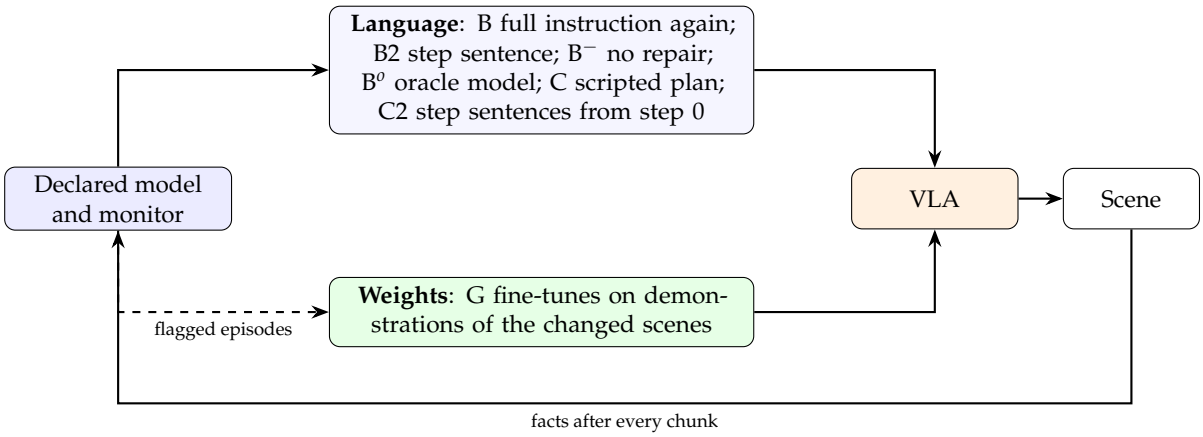

Figure 3. Where each arm acts on the VLA. Six arms act through the policy's language input and arm G through its weights. Arm A is the VLA alone.

Fair comparison.

All arms use the same checkpoints, settings, tasks, seeds and step limit. The main runs use seed 2000 and initial states 10 to 19 of each task, which no earlier run used, with 10 episodes per task and world. The declaration was frozen with file hashes before any run of arms B to G; every later change is a numbered amendment committed before the run it affects.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

9. The Worlds

Nominal.

The ten LIBERO-10 tasks.

Perturbed.

Changes the demonstrations never contained, made by the harness during the episode (Figure 4).

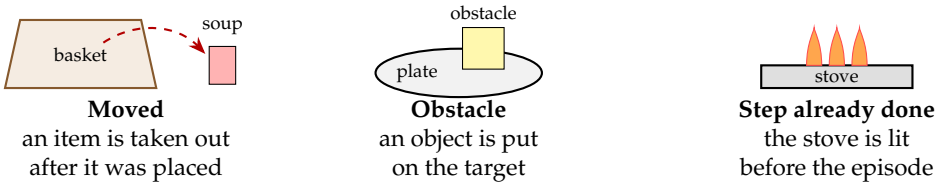

Figure 4. The three changes kept. None appears in the demonstrations; each lowers the success of the VLA alone.

A perturbation is kept only if it lowers arm A by at least 3 of 10 episodes or with a one-sided Fisher p below 0.10, a rule fixed before the pilot. Two passed on Pi0.5: an object *moved* after it was placed (3 of 10 against 10 of 10, $p = 0.002$) and an *obstacle* on the target (4 of 10 against 9 of 10, $p = 0.03$). In the failures the policy goes on or idles; it does not redo the undone step. A third change, a drawer that resists the first closing push, was tested and not kept: the lock held in 10 of 10 episodes and Pi0.5 succeeded in 10 of 10, retrying after the first push.

A step already done.

In the task "turn on the stove and put the moka pot on it", the stove was lit before the episode began. Pi0.5 succeeded in 3 of 10 episodes, against 10 of 10 with the stove unlit on the same initial states (one-sided Fisher $p = 0.0015$); in every failure the stove stayed lit and the pot never reached it. The scene no longer matches the demonstrations, because one step is already done. A declared model reads the lit stove as a fact, drops the step from its plan (Figure 2) and sends "put the moka pot on the stove", a sentence Pi0.5 follows on the nominal task. This world is kept.

Forbidden states.

Two forbidden states were declared: the stove lit with nothing on it at the end, and an object not named by the goal on a lit stove at any time. The nominal runs almost never enter them (1 of 40 episodes). In the world with the stove lit at the start the first state occurred in 7 of 10 episodes, each of them a failure, and the second in none; as a measure it repeats the success rate. It is kept as a secondary count for every arm. The guard rail declared for these worlds (no release of an object not named by the goal over a lit stove) would not have fired once in 8519 chunks of 470 recorded episodes, so arm B without guard rails would equal arm B here; that arm was dropped, and the test of guard rails is left to a world built for it.

10. Results

The VLA alone.

In the Kaggle replication, on the main-run seeds and the first 10 initial states, Pi0.5 alone succeeded in 94 of 100 nominal episodes (Wilson 0.88 to 0.97) and in 26 of 50 perturbed ones (0.39 to 0.65). The nominal rate agrees with the reproduction (96 of 100). In the main runs on AWS it succeeded in 94 of 100 nominal and 38 of 100 perturbed episodes. The changes cost the policy more than half its episodes, and the step already done cost the most.

Success per arm.

On the perturbed worlds every arm that corrects through language lies within 3 episodes of arm A's 38 of 100, apart from C2 (18 of 50 against 21 of 50); the fine-tuned VLA reaches 46. No difference is

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

significant (Figure 5; two-sided Fisher $p \geq 0.68$). With 100 episodes per arm the smallest difference the comparison detects (80% power) is about 17 points, so gains of a few points cannot be excluded; gains of 17 points or more are unlikely. Single cells move in both directions by up to 4 of 20, which is the spread of separate rollouts from the same states: arm A itself, run twice on states 10 to 19 (once on each platform), succeeded in 26 and 21 of 50, a difference of 10 points with nothing changed. On the nominal tasks the OWPA costs nothing: 93 of 100 against 94 (Fisher $p = 1.0$), and the oracle model 96.

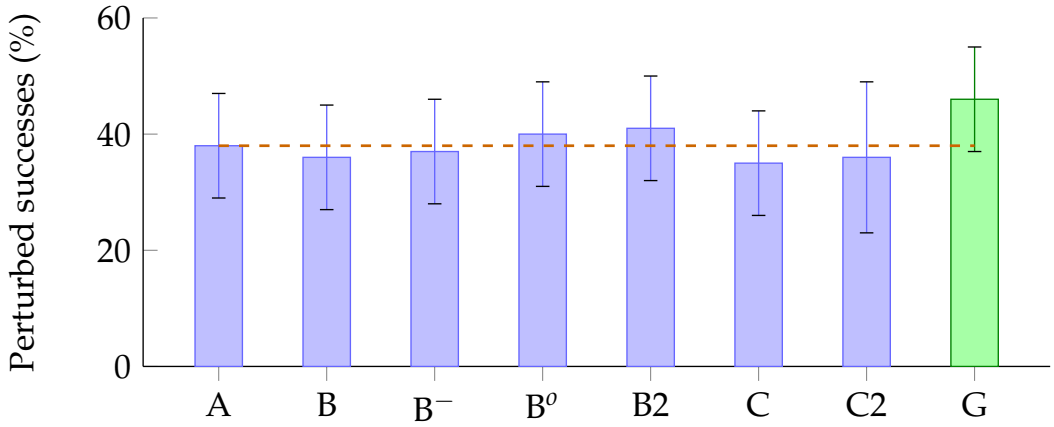

Figure 5. Pi0.5 on the perturbed worlds (Table 5): success rate per arm with 95% Wilson intervals (100 episodes per arm; C2 50). The dashed line is the VLA alone. Every correction through language lies on it; the fine-tuned VLA (G, green) lies above it, within the interval.

Table 5. Pi0.5, main runs on AWS: successes per arm and world (seed 2000; initial states 10 to 19, and 40 to 49 in the perturbed worlds). B2 re-instructs a stalled policy with the joined pick-and-place sentence of the current step; C2 sends the joined sentences of the scripted plan from the first step, on states 10 to 19 only (cells of 10, total of 50; arm A has 21 of 50 on the same states). G is the VLA fine-tuned on demonstrations of the perturbed worlds; on nominal tasks 0 and 4 it succeeded in 20 of 20 (arm A 17). Empty cells were not run.

World	A	B	B⁻	B⁰	B2	C	C2	G
Nominal, 10 tasks (of 100)	94	93		96				
Moved, task 0 (of 20)	13	11	14	12	13	12	3	16
Obstacle, task 0 (of 20)	8	6	8	6	9	5	4	10
Step already done, task 2 (of 20)	4	3	2	3	4	3	2	3
Moved, task 4 (of 20)	6	10	9	12	5	9	4	11
Obstacle, task 4 (of 20)	7	6	4	7	10	6	5	6
Perturbed, total (of 100)	38	36	37	40	41	35	18	46

Detection of failures.

The monitor was replayed offline on arm A's own perturbed episodes, which record the world facts at every chunk boundary, with the settings frozen for arm B. It flagged 54 of the 62 failures (Figure 6), with a median of 360 of the 520 steps still available (minimum 10) and a median of 150 steps after the perturbation. The first event was an idle arm in 33 and a failed prediction in 21. Every failure on the step already done (16 of 16) and on the moved objects (21 of 21) was flagged; the misses are on the obstacles (8 of 25), where the policy goes on moving without a fact changing. The written and the oracle models gave identical events on all 100 episodes. In 23 of the 38 perturbed successes the monitor also raised an event, mostly a failed prediction when an object was moved: the change was real and the policy recovered on its own. On arm A's unperturbed runs, where nothing changed, 7 of 94 successes had an event (9 events: 8 a door or drawer left ajar, 1 failed prediction). Arm B's own rollouts on the same states give the same count (54 of 62). For SmolVLA the monitor flagged 85 of 90 failures (median 320 steps left) and raised an event in 8 of 68 unperturbed successes. Figure 7 shows

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

two failures as the monitor reads them. Only the first event is measured this way: in a live run the monitor's own actions change what follows.

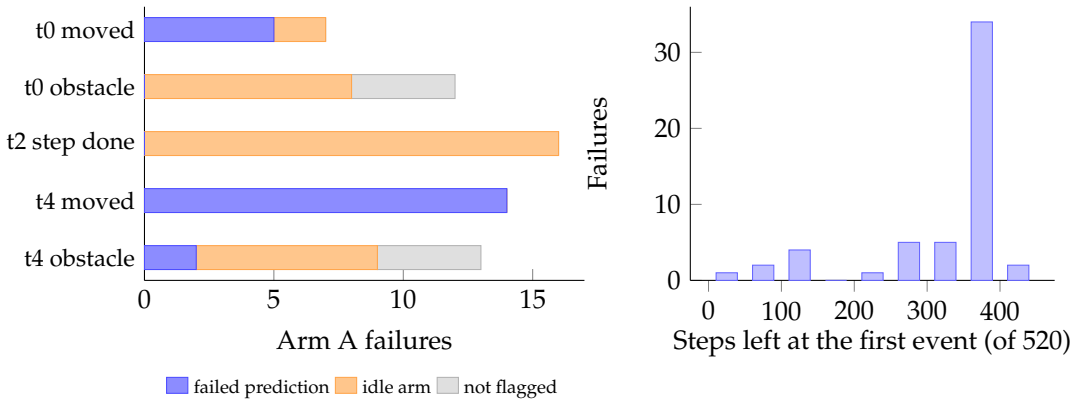

Figure 6. The monitor replayed on arm A's own perturbed episodes (Pi0.5, 62 failures). Left: the first event per failure, by cell. Right: steps still available when it fired.

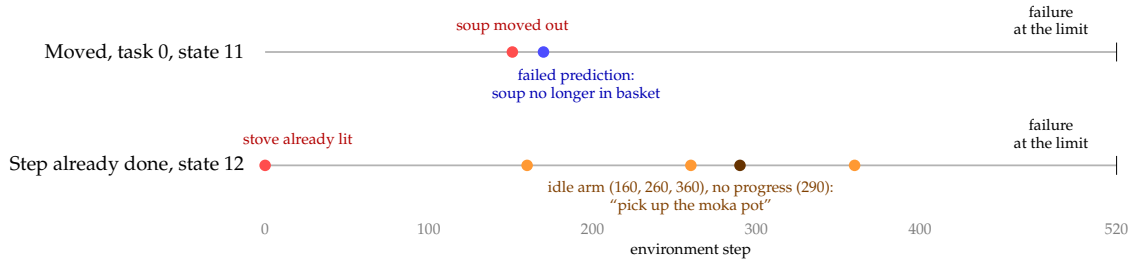

Figure 7. Two arm A failures as the monitor reads them. In the moved world the failed prediction names the fact that changed 19 steps after the change. With the stove already lit the arm stalls, and the monitor names the stalled step (picking up the pot) 360 steps before the limit.

Recovery after detection.

Arm B succeeded on 14 of the 62 states where arm A failed (0.23, Wilson 0.14 to 0.34), B2 on 17 and the scripted plan on 16. Over the same states each arm also failed where arm A had succeeded (B 16 of 38), so these swaps are the run-to-run noise of separate rollouts, not rescues the monitor caused. When arm B re-instructed, it sent the full task instruction in 98% of cases, because for most plan steps the policy's trained vocabulary has no shorter sentence; B2 sent a step sentence in 91% and did no better. The right sentence from the first step (C2) did no better either. On the step already done, B2 sent "put the moka pot on the stove", a sentence Pi0.5 carries out on the nominal task, and succeeded in 4 of 20: the sentence was followed in the nominal scene and not in the changed one.

Language sensitivity.

To test whether the policy reads the instruction at all, Pi0.5 was given another task's instruction in task 0's scene (soup and tomato sauce into the basket), 10 episodes per case on the same states as arm A. With task 1's instruction (cream cheese box and butter into the basket), a task trained in the same scene, it put the cream cheese box and the butter in the basket in 10 of 10 episodes and never did task 0. With task 7's instruction (soup and cream cheese box), a combination trained only in another scene, it never reached that goal: it did task 0 in 4 episodes, task 1 in 4, and neither in 2. In the moved world the same instruction lowered task 0 from 8 of 10 to 2 of 10, again without reaching the instructed goal. The policy reads the instruction, and the instruction chooses among the behaviours trained in that scene; a combination it was not trained on in that scene is not composed.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

Repair and the oracle model.

Arm B called the writer to repair its model in 43 of 100 perturbed episodes (49 calls, none failed) and in 3 of 100 nominal ones, where nothing in the world had changed: 3 false repairs. Removing repair (B^{-}, 37 of 100) and replacing the written model with the oracle model (B^{o}, 40 of 100) leave the totals where they were. The plan changed in 10 of the 43 perturbed episodes with a repair. The perturbations leave every task rule true (the soup still belongs in the basket), so a repair of the task rules had little to correct. The model is therefore not the limit: a perfect model, read and acted on, does not raise success either. Whether each repair was correct and supported by the mismatches that triggered it could not be scored on these runs, because the episode records kept the outcome of each repair and not the rules it wrote.

The model as a predictor.

The written model was also scored as a forecaster, offline on arm A's episodes: at each chunk boundary, the fact changes it predicted for the current plan step against those observed. On the nominal tasks 92% of the predicted changes occurred while the model was on that step, and 79% of the transitions in which a progress fact changed contained at least one predicted change (47% contained only predicted changes). The unpredicted changes are almost all outside the model's vocabulary: 46% of the observed changes are facts it has no rule for, such as an object resting on a fixture or a drawer passing through ajar, and only 2% are facts it knows but did not expect at that moment. After a perturbation, recall fell to 62%, facts outside the vocabulary rose to 61% and known facts missed to 9%. The oracle model scored the same (nominal recall 97%, precision 49%). The declared model forecasts what the plan should achieve, which is what it was written to do; it does not describe everything that happens in the scene.

SmolVLA.

SmolVLA alone succeeded in 68 of 100 nominal episodes and in 10 of 100 perturbed ones. With the OWPA it succeeded in 66 of 100 nominal episodes and in 13 of 80 perturbed ones (arm A has 10 on the same 80; the two task 4 cells on states 40 to 49 were not run). It follows 5 of 24 sub-goal sentences, so it can only be given its full instruction again, and its success in the changed scenes is too low for a comparison of arms.

Fine-tuning on demonstrations.

The fine-tuned VLA (arm G) succeeded in 46 of 100 perturbed episodes against 38 for the VLA alone on the same states (Wilson 0.37 to 0.56; Fisher $p = 0.32$), 26 of 50 against 21 on states 10 to 19 and 20 of 50 against 17 on states 40 to 49. It is the largest gain of any arm and the only one in the same direction on both sets of states, but it is within the 17 points the comparison can detect, and the gain on states 10 to 19 equals the difference between two runs of arm A on those states, so it is a trend, not a result. It is also a lower bound for more data: only the action expert was trained, with the vision-language backbone frozen, for five passes over the policy's own successful rollouts, 1 to 5 per cell. In OpenVLA [25], freezing the vision encoder during fine-tuning gave poor results even with the language model trained, and a narrow task used about 50 demonstrations; a changed scene must be seen before it can be acted on. Training on its own successes reinforces what the policy can already sometimes do, which is consistent with where the gain appears. The gain is where the demonstrations were (an observation, not a declared test): on the four cells with 4 or 5 demonstrations, 43 of 80 against 34; on the step already done, which had a single demonstration, 3 of 20 against 4. On nominal tasks 0 and 4 the fine-tuned VLA succeeded in 20 of 20 against 17, so the new skill did not cost the old one.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

11. Discussion

Detection and correction.

The declared model does what it was built to do: it states what each step should achieve, and when the world changes it notices, in most failures with more than half the episode left. What it cannot do is make the policy act on that knowledge. Every channel through language was tried: the full instruction again, a step sentence, a repaired model, the oracle model, a correct plan, a correct plan from the first step. None raised success in the changed scenes, while the same sentences are followed in the nominal ones. The limit is the policy's skill in a scene its demonstrations did not contain, not the words it is given or the model that chooses them. The language interface of these policies selects among the behaviours they learned; it does not compose new ones. The language test shows this directly: given another task's instruction, Pi0.5 carried out the task trained with that instruction in the same scene (10 of 10), but not a combination trained only elsewhere (0 of 10). The text reaches the policy; it selects a learned behaviour. A policy co-trained on broad vision-language data, as in RT-2 [6], may compose further.

Operators the controller cannot execute.

The declared model plans over operators, the sentences it sends to the VLA, and only 7 of the 35 plan steps are sentences the policy was trained on. A plan is only as good as the operators the controller can carry out, and in the changed scenes even the trained ones failed. The failures are also regular: the policy fails on an obstacle on the target and on a step already done. These are preconditions of the controller's skills, not of the task. The repair in this study changes task rules; a repair that learned skill preconditions from the flagged failures ("placing on the plate requires the plate clear") could plan around them, for instance by first sending the step that removes the obstacle. This is the grounding of operators in skills that domain-learning methods such as LODGE [22] perform by execution, and it is the step this study leaves open. Two further gaps follow from the same view. The monitor had no strategy for exploring: on a stall it sent the full instruction again in 98% of cases, the same action in the same situation, which yields no new evidence. And the rules are deterministic, while the controller's skills succeed with a probability that depends on the scene; a model that kept a success estimate per step and scene, updated from the flagged failures, would know which steps to avoid, which to retry and which to replace.

What a declared model adds to a VLA.

It adds an observer that costs nothing on the nominal tasks and turns a silent failure into a named event: which fact did not change, at which step, in which scene. That event is what a deployed robot needs to stop, ask for help or log the case. The declared goal also tells when the task is done, which a VLA trained by imitation does not know: it stops when its episode times out, and a drawer left ajar looks finished to it. It is also a label: the episodes it flags are the situations the policy cannot handle, collected without a person watching, as large-scale robot data collection with declared safety rules requires [27]. Fine-tuning on the cases where a policy fails is the usual remedy, but finding those cases takes a person who watches the robot and judges where it failed, as in DAgger [26]; the declared monitor does that work and names the fact that failed. Arm G took the next step of that loop with 19 demonstrations of the changed scenes, and is the only arm that moved success upward: by 8 points, not significant at this size. The words had no effect; a little data had a small one. Whether more demonstrations of the flagged situations close the gap is a question of scale, which this study does not answer.

A task model above a learned controller.

The declared model does not predict how an object moves, only which facts a step should change: that the pot ends on the stove, not the path it takes or how it might slip. It is a model of the task's causal structure, and it leaves dynamics, contact and geometry to the VLA. This is the division between

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

levels that hierarchical proposals for autonomous intelligence call for [1]: an abstract level that plans over long horizons, above a level that acts in continuous space. Here the abstract level is declared rather than learned, and its vocabulary of facts was chosen by hand. Whether a language model can choose that vocabulary for a new domain, and whether it can be read from the camera instead of the simulator, are the next questions; the second was answered in a grid world in Terra Incognita II [4].

Two layers of checking.

The engine's check before acting and the comparison after acting find different errors. The first finds a model that contradicts itself and found none. The second finds a model that is wrong about the world. A wrong rule that is consistent with itself (the basket that must be opened) passes the first and can only be caught by the second. This is the reason the architecture repairs its rules from confirmed mismatches at run time rather than validating them once before deployment. The prediction is also what makes a wrong rule noticed at all: language models left to explore without one tend to observe an anomaly and carry on [23].

The instruction and the benchmark.

Two of the writer's three error types come from the instruction, not the writer: it does not say which spatial relation is checked, or that the stove must be on. A model written from the instruction can only be as complete as the instruction. In this study that cost was not visible: the oracle model (B^o) did no better than the written one.

Steering through language.

Pi0.5 carried out 23 of 24 sub-goals on the nominal tasks, SmolVLA 5 of 24. The first can be steered step by step in scenes it knows; the second can only be given its full instruction again. In changed scenes neither could be steered. The original Pi0.5 predicts its own sub-tasks before acting; whether that built-in level can steer the policy where the declared plan could not is left open, since the port used here does not include it. How far a declared layer can correct a policy therefore depends on how well the policy follows language in the scene at hand, which is a property to measure before the layer is placed.

12. Limitations

- Perception reads the simulator's state, not the camera, into a vocabulary of facts chosen by hand. A camera reader adds errors this paper does not measure, and a new domain needs a new vocabulary.
- The declared model predicts symbolic facts, not continuous motion; it cannot foresee how an action will fail, only that a fact did not change.
- The thresholds of the monitor were set on the recorded episodes it was replayed on; the main runs use new seeds and initial states for that reason.
- Three kinds of change are kept (a moved object, an obstacle, a step already done); a fourth, a resisting drawer, did not lower the primary VLA. Guard rails are not tested against a world built to tempt a violation.
- The trained instruction vocabulary covers 7 of 35 plan steps; for the rest the declared layer relies on the full instruction or on composed sentences whose compliance is measured, not assumed.
- One writer model was used, and the writer was not tuned to the task set.
- Arm G is a light fine-tune: the vision-language backbone, vision encoder included, was frozen, and each perturbed world had 1 to 5 demonstrations. Its result bounds what more data can do from below; it does not measure it.
- Both VLAs are LIBERO-fine-tuned checkpoints; the finding about correction through language is established for such policies, not for VLAs trained more broadly. The language test used one scene and two instructions.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

- The perturbations are made by the harness, on three tasks, with 20 episodes per arm and perturbed cell. A standard robustness suite such as LIBERO-plus was not used, so the numbers are not directly comparable with other work.
- The thresholds for no progress and an idle arm are engineering choices, fixed before the main runs, not derived from a principle.
- A repair calls the language model inside the episode: in a first cell each call took 6.5 to 8.9 s, against about 1 s of simulation and policy time per chunk. In simulation the world waits during the call; on a robot the arm would have to hold, and that pause is not measured here.
- Ten episodes per task and world give wide intervals per task; comparisons are made on suite totals, which detect differences of about 17 points, not smaller ones.
- The rules written by each repair were not recorded in the main runs, so repairs are counted, not scored as correct.

13. Future Work

- Read the facts from the camera with a vision model, as in Terra Incognita II [4], and measure the cost of perception errors.
- Keep repaired rules and skill success estimates across episodes and tasks, so that one corrected failure improves every later episode of its kind, and measure their reuse.
- Carry the declared model to a real robot, where no exact simulator exists, the facts must come from the cameras, and the arm must hold while a repair is written; a low-cost arm supported by LeRobot is the natural first platform.
- Fine-tune the whole policy, vision encoder included (for instance with low-rank adapters on all layers [25]), on a realistic number of demonstrations of the flagged situations (tens per situation, not 1 to 5), including the failures and their corrections, and measure the gain against the amount of data.
- Repeat the comparison on a standard robustness suite such as LIBERO-plus, and with policies that condition their visual features on the instruction.
- Learn the preconditions of the controller's skills from the flagged failures, and plan around them before collecting new data.
- Close the loop from detection to data: collect the episodes the monitor flags, add demonstrations of those situations, retrain, and measure how many rounds a new kind of change needs.
- Compare the declared monitor with a learned verifier of action chunks in the style of CheckVLA [16], on the same episodes.
- Use the declared model to plan in action space, with a simulator or a learned model checking the contact details it cannot state, in the style of SIMPACT [15].

14. Conclusions

A VLA can be given an explicit, readable model of its world without changing its weights. A language model writes that model from the instruction and the first scene, and an engine runs it to plan and to predict what each step should achieve. Beside two open VLAs on LIBERO, the declared model was a reliable observer: in scenes the demonstrations did not contain, it noticed most failures with more than half the episode left, named the fact that did not change, and cost nothing when the scene was as trained. It could not make the policy recover. A repeated instruction, a better sentence, a repaired model, a perfect model and a correct plan from the first step all left success where the VLA alone put it. Detection and correction are separate problems. The first is solved by a declared world model; the second lies in the policy's skill in the changed scene, which words select but do not create. A small fine-tune on demonstrations of the changed scenes was the only intervention that moved success, a trend this sample cannot confirm. The open-world predictive architecture beside a VLA is therefore an observer that knows what should happen, says when it did not, and collects the situations from which the policy must learn.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

Data Availability Statement: The engine, the writer prompt and the monitor are proprietary to Cognitive Fab. They are available for review under a non-disclosure agreement. **Use of AI tools:** Claude (Anthropic) assisted the author in writing the experiment code, the analysis scripts, the figures and the text. Claude Opus 5.5 is also a component of the system studied: it writes and repairs the declared models. The author designed the study, checked every reported value against the committed results, and is responsible for the content.

Acknowledgments: The main runs used AWS; the replication runs used GPUs provided by Kaggle. The language model runs were made with API credits provided by Anthropic. The VLA checkpoints and the evaluation code are those published by Hugging Face (LeRobot) and Physical Intelligence (OpenPI); the tasks are those of the LIBERO benchmark.

Conflicts of Interest: The author is the founder of Cognitive Fab LLC, which develops the engine and the declared world model used in this work. The model runs used API credits provided by Anthropic, the developer of the model used as the writer.

References

1. Y. LeCun. A path towards autonomous machine intelligence. OpenReview, version 0.9.2, 2022.
2. J.-J. Dubray. A declared world model: language models write the rules of the scene they are shown. Preprints.org, doi:10.20944/preprints202610.1139.v1, 2026.
3. J.-J. Dubray. Terra Incognita: a declared world model that assumes what it cannot see and repairs its rules. Preprints.org, 2026.
4. J.-J. Dubray. Terra Incognita II: perception, prediction and repair in an open-world predictive architecture. Preprints.org, 2026.
5. J.-J. Dubray. An open-world predictive architecture for business: a macroeconomic digital twin that reads data and text. Draft, 2026.
6. A. Brohan et al. RT-2: vision-language-action models transfer web knowledge to robotic control. arXiv:2307.15818, 2023.
7. K. Black et al. π_0: a vision-language-action flow model for general robot control. arXiv:2410.24164, 2024.
8. Physical Intelligence. $\pi_{0.5}$: a vision-language-action model with open-world generalization. arXiv:2504.16054, 2025.
9. M. Shukor et al. SmolVLA: a vision-language-action model for affordable and efficient robotics. arXiv:2506.01844, 2025.
10. R. Cadene et al. LeRobot: state-of-the-art machine learning for real-world robotics in PyTorch. GitHub, huggingface/lerobot, 2024.
11. B. Liu et al. LIBERO: benchmarking knowledge transfer for lifelong robot learning. NeurIPS Datasets and Benchmarks, arXiv:2306.03310, 2023.
12. Y. Zhu et al. robosuite: a modular simulation framework and benchmark for robot learning. arXiv:2009.12293, 2020.
13. E. Todorov, T. Erez and Y. Tassa. MuJoCo: a physics engine for model-based control. IROS, 2012.
14. M. Assran et al. V-JEPA 2: self-supervised video models enable understanding, prediction and planning. arXiv:2506.09985, 2025.
15. H. Liu, S. Yao, H. Chen, J. Gao, J. Mao, J.-B. Huang and Y. Du. SIMPACT: simulation-enabled action planning using vision-language models. CVPR, arXiv:2512.05955, 2026.
16. Y. Liu et al. CheckVLA: execution-time verification with action-conditioned world model for long-horizon mobile manipulation. arXiv:2607.26789, 2026.
17. Y. Wu et al. From foresight to forethought: VLM-in-the-loop policy steering via latent alignment. arXiv:2502.01828, 2025.
18. Z. Sun et al. Pre-VLA: preemptive runtime verification for reliable vision-language-action and world-model rollouts. arXiv:2605.22446, 2026.
19. X. Cai et al. τ_0-VLA: a hierarchical robot foundation model with world-model-guided test-time computation. arXiv:2608.16885, 2026.
20. S. Zhou, T. Zhou, Y. Yang, G. Long, D. Ye, J. Jiang and C. Zhang. WALL-E 2.0: world alignment by neurosymbolic learning improves world model-based LLM agents. arXiv:2504.15785, 2025.
21. H. Tang, D. Key and K. Ellis. WorldCoder, a model-based LLM agent: building world models by writing code and interacting with the environment. NeurIPS, arXiv:2402.12275, 2024.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.

Preprints.org | NOT PEER-REVIEWED | Posted: 3 November 2026 https://doi.org/10.20944/preprints202611.0227.v1

22. C. Kienle, B. Alt, O. Arenz and J. Peters. Learning hierarchical domain models through environment-grounded interaction (LODGE). arXiv:2505.13497, 2025.
23. Z. Yang, W. Wang, X. Xu and Y.-C. Chen. Can LLMs uncover hidden physical laws through free exploration? arXiv:2610.37649, 2026.
24. Q. Shen et al. World action models: a survey. arXiv:2606.20781, 2026.
25. M. J. Kim et al. OpenVLA: an open-source vision-language-action model. arXiv:2406.09246, 2024.
26. S. Ross, G. Gordon and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. AISTATS, 2011.
27. M. Ahn et al. AutoRT: embodied foundation models for large scale orchestration of robotic agents. arXiv:2401.12963, 2024.
28. E. B. Wilson. Probable inference, the law of succession, and statistical inference. Journal of the American Statistical Association 22(158), 209 to 212, 1927.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.

© 2026 by the author(s). Distributed under a Creative Commons CC BY license.