RoboLineage: Agent-Native Data Lifecycle Governance Across Robot Policy Iterations

Qian Luo[1,2] **Wentao Guo**[1,3] **Zhennan Qin**[1] **Nanchun Guo**[1]
Yunhan Zhao[1] **Yi Ma**[1,2] **Yanchao Yang**[1,2]

[1]TranscEngram [2]The University of Hong Kong [3]Beijing Institute of Technology

Project page: https://robolineage.github.io/

Abstract: We present RoboLineage, an agent-native data lifecycle governance system for robot policy iteration. Modern robot policies improve through repeated cycles of data collection and retraining. Teams still rely on local tools, scripts, and expert memory to review the evidence and decide which data to use or which policy to release. RoboLineage records how rollout evidence leads to dataset changes and how those datasets produce policy versions. Evaluation and release decisions remain linked to that history, including the plan for the next collection round. Agents interpret embodied evidence and prepare accepted data for existing training stacks. They also assess dataset readiness and carry context across iterations through typed, validated artifacts. In real-robot manipulation workflows, RoboLineage makes routine policy iteration faster and more auditable while maintaining downstream policy performance. We open source RoboLineage as a lightweight lifecycle layer for different robot embodiments and training families.

Keywords: Robot learning, Data lifecycle governance, Data lineage, Agentic systems

1 Introduction

Robot learning has become increasingly data-rich. Modern imitation learning and vision-language-action (VLA) pipelines can train capable policies from teleoperated demonstrations, in-the-wild datasets, and policy-specific corrections [1, 2, 3, 4, 5, 6, 7]. At the same time, the day-to-day work of improving a real robot policy is still surprisingly manual. After collecting a rollout batch, the team reviews the evidence and curates a training set. It then prepares inputs for the learner and retrains the policy. Evaluation determines whether to release the new version or collect more data. These steps form the data lifecycle through which a robot policy improves.

Expert robotics groups already manage this lifecycle with care. Their established procedures cover data preparation through policy release. The difficulty is that these practices rarely become a durable accountability chain across policy iterations. Human experts repeatedly reconstruct which execution evidence justified a dataset change, which data trained a checkpoint, why an evaluation was trusted, and what should be collected next. This reconstruction is fragile: evidence is scattered, decisions are remembered through local practice, and lineage across iterations remains weak until experts piece it back together. This fragility matters more as datasets span multiple robots and training moves toward generalist policies. Recent work shows that policy quality depends on which data is selected and how it is reused [8, 9, 5, 10, 11].

Robot policy iteration calls for explicit lifecycle governance. Rollouts, datasets, and trained policies need links to the observations, reviews, admission rules, and evaluations that justify their use.

10th Conference on Robot Learning (CoRL 2026), Austin, Texas, USA.

Recording these links across iterations makes it possible to audit a dataset change, trace a checkpoint to its training inputs, and recover the evidence behind a deployment decision.

To address this problem, we introduce RoboLineage, a lightweight, open-source, agent-native governance system that turns the messy data ecosystem behind robot policy iteration into an explicit data lifecycle. Each decision carries a versioned record of its supporting evidence, so later iterations can recover how a policy was produced. Figure 1 summarizes this shift. Agents write typed artifacts when they interpret a rollout or prepare data for training. Review records link admitted episodes to evidence; training records connect frozen datasets to checkpoints. Dataset-health reports identify gaps before training. Evaluation summaries then carry failures into a brief for the next collection round.

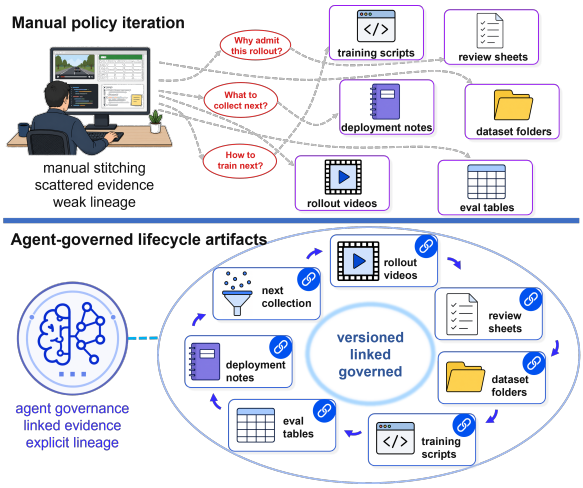


Figure 1: Manual policy iteration versus agent-governed lifecycle artifacts.

RoboLineage keeps policy iterations legible across changing robots, collection workflows, and training stacks by preserving the same lifecycle semantics. Its core abstraction is a lifecycle artifact contract for embodied policy iteration: agents may interpret evidence, adapt data, and summarize state, but consequential transitions enter the lifecycle only as validated, evidence-linked artifacts. In routine iteration, a data-collection operator can collect rollouts, inspect prepared evidence, resolve uncertain cases, and launch the next training cycle through the frontend, while RoboLineage carries the decision trail forward across policy versions.

Our contributions are:

- We formulate robot policy iteration as a data lifecycle governance problem and define a typed lineage graph for tracking how evidence, artifacts, and decisions carry across policy iterations.
- We introduce an agent-native governance architecture in which agents operate as artifact-producing lifecycle workers, turning task-grounded semantic and integration judgments into bounded, auditable lifecycle state.
- We realize RoboLineage as a lightweight open-source implementation with an operator frontend and portable lifecycle artifacts, and evaluate review reliability, routine-cycle effort, and closed-loop recollection across multiple robot and training families.

2 Related Work

Robot data collection and policy iteration. Large datasets give policies access to broader robot experience, while scalable environments make it easier to expand training coverage [8, 9, 5, 12, 6, 13]. Collection systems such as GELLO and UMI simplify demonstration collection and correction [14, 15, 16, 17, 18]. RoboMimic and MimicGen study how demonstrations can be used and generated for learning [19, 20]. Other work examines which demonstrations to retain and how dataset quality affects policy performance [21, 22, 23, 24, 25, 10]. RoboLineage follows the decisions that connect these datasets to later policy versions. It records why an episode was admitted and preserves useful failure evidence. That history also links a policy to its training data and informs recollection.

Agentic robotics systems. Language-model systems support affordance-grounded instruction following and planning from observations [26, 27]. Some express robot behavior as code or 3D value

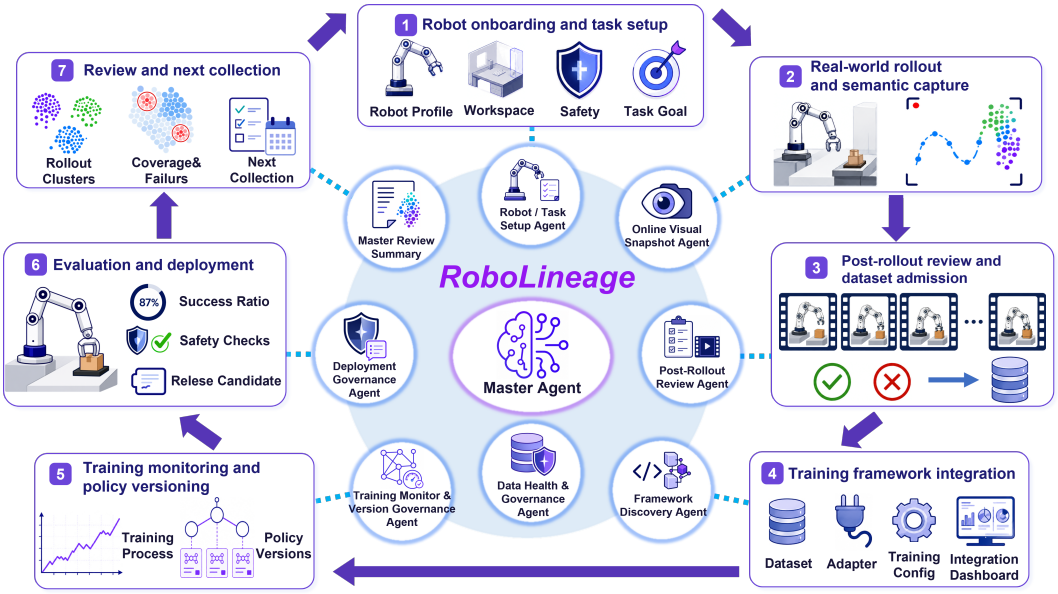


Figure 2: RoboLineage governs robot policy iteration as an explicit data lifecycle. The outer loop shows the policy-iteration cycle from robot/task grounding to next collection; the inner layer shows lifecycle agents that read and write typed artifacts at each transition.

maps [28, 29]. Other work develops embodied multimodal models and generalist robot policies [30, 4, 7, 11, 31, 32, 33]. VLA fine-tuning adapts learned policies [34], and recent agentic frameworks address longer planning and execution horizons [35, 36, 37]. These systems place model agency close to robot behavior. RoboLineage applies agentic reasoning to the data lifecycle that produces later policies. An agent may interpret execution evidence or a training repository, but its judgment enters the lifecycle as a schema-validated artifact.

MLOps, provenance, and reporting. ML systems research shows how technical debt and data cascades affect the systems around a model [38, 39]. Production pipelines and experiment trackers organize training runs and their outputs [40, 41, 42, 43]. Provenance records make data histories traceable [44, 45], while dataset documentation and model cards describe assumptions and intended use [46, 47]. Robot policy iteration needs a comparable record of decisions grounded in execution evidence. RoboLineage links a rollout's terminal evidence to its admission decision and preserves subsequent overrides. Dataset locks carry those decisions into training, and evaluation failures inform the next collection brief.

3 Method

3.1 Problem Definition: Governing a Robot Data Lifecycle

Consider a robot policy iteration loop indexed by k. Let c_k denote the collection process for that iteration: a data-collection operator, teleoperation interface, scripted routine, policy-assisted rollout protocol, intervention policy, or a mixture of these. Under embodiment b and task contract τ, c_k produces rollouts $\mathcal{R}_k = \{r_i\}$. Each rollout contains raw streams such as camera frames, robot state, actions, end-effector state, task-relevant tool signals, synchronization metadata, and operator events. Review turns a rollout into semantic labels and decisions: whether the task succeeded, where it failed, whether the episode is useful for training, and whether a human should inspect it. Accepted rollouts become dataset D_k, while routed and rejected artifacts remain linked for audit, failure analysis, and later collection. A training run T_k produces policy candidate π_k; evaluation trials $\mathcal{E}_k$ lead to deployment review, rollback, or recollection decisions G_k.

We define the robot data lifecycle as the ordered set of transformations and decisions triggered by routine operator-visible actions: demonstrate, rollout, intervene, annotate, train, evaluate, prepare release, and revert. Abstractly, one iteration is

$$(b, \tau, c_k) \to \mathcal{R}_k \to \mathcal{A}_k \to D_k \to T_k \to \pi_k \to \mathcal{E}_k \to G_k,$$

where $\mathcal{A}_k$ denotes review and admission artifacts. Lineage records the evidence, rules, and artifacts behind each transition. The transition is the natural unit of governance because policy quality is often determined between runs: which evidence was trusted, which rollout changed the dataset, which training input produced a checkpoint, and which failure changed the next collection plan. The typed lineage graph is the executable form of this accountability chain: nodes carry lifecycle artifacts, and edges record the evidence-linked transitions that allow later agents, operators, and audits to recover why a policy version exists. If collection uses a prior policy, that policy is recorded as part of c_k and the rollout metadata, alongside other collection sources.

We represent lineage as a typed, versioned artifact graph

$$\mathcal{L}_k = (\mathcal{V}_k, \mathcal{P}_k)$$

whose nodes are lifecycle artifacts and whose edges encode relations such as `observes`, `reviews`, `admits`, `freezes`, `trains`, `evaluates`, and `recommends`. Setup nodes record the robot profile, task contract, and collection context. Rollout records link raw evidence to snapshots and review annotations; admission decisions and dataset locks specify the training inputs. Training and evaluation nodes then track the resulting policy, with deployment tickets and master summaries carrying decisions into the next iteration. Agents share this graph as the governance state for successive policy versions. The outer loop in Figure 2 visualizes this formulation from robot/task grounding through next collection. Appendix C.1 gives the edge semantics and an end-to-end trace.

3.2 Agent-Native Governance Over Lifecycle Artifacts

RoboLineage provides the agent-native implementation of the inner layer of Figure 2: agents act as artifact-producing lifecycle workers, reading the current lineage context and writing governed artifacts at each transition. The governance problem is to automate enough routine work for a data-collection operator to carry an iteration forward with agent support, while keeping data and policy changes reproducible. This graph view motivates five design principles. **Traceability** requires each policy to retain the evidence and data rules that produced it. **Semantic coverage** preserves the task meaning behind raw observations. **Agent autonomy with explicit state** lets agents make context-dependent judgments through typed artifacts that later stages can inspect. **Portability** localizes robot differences in profiles and learner differences in adapters. **Operator efficiency** reduces routine review and data preparation while retaining human inspection of ambiguous cases.

Each agent has a lifecycle scope, reads the current artifact context, writes a governed artifact, and exposes uncertainty and dependencies to the next stage:

$$v_j = A_j(\mathrm{ctx}_j(\mathcal{L}_k), x_j; b, \tau)\,, \qquad v_j \in \mathcal{T}_j,$$

where v_j may be a visual snapshot, review annotation, data-health report, framework profile, evaluation summary, or master summary. Agents can therefore perform substantial semantic work while their outputs remain versioned lifecycle state, turning context-dependent expert judgments into repeatable operations with explicit inputs, uncertainty, and provenance.

Fixed rules implement schema checks, content hashes, dataset locks, and configured thresholds. Agents handle context-dependent judgments: interpreting terminal evidence, assessing whether a failure is useful, mapping a repository's data contract, and explaining a coverage gap. Their outputs enter the lifecycle through the same typed artifact boundary, so downstream workers can inspect the evidence and uncertainty behind a recommendation. This design makes the semantic work reusable across policy iterations while preserving deterministic validation at each transition.

Appendix A.2 describes the shared contract, and Appendix B.11 expands the module rationale.

3.3 Robot Onboarding and Task Grounding

During setup, RoboLineage writes a robot profile b and a task contract τ into lineage. The Robot Onboarding Agent resolves local topics and interfaces into normalized streams. The profile records how camera, robot state, and action data are bound, including coordinate frames and timing. Recorder health and auxiliary tool signals support validation, and warnings remain attached to the profile. For ROS-based systems, the operator confirms these bindings before collection. The Task Config Agent gives the execution its task meaning. It defines phases and success criteria, then specifies which events require closer inspection and how episodes should be admitted. The contract also specifies the collection policy. Downstream agents use these contracts without interpreting each robot's local naming conventions again. Details are in Appendix B.1 and Appendix B.2.

The operator enters the task goal and recording topics through the UI. Agents inspect topic types and sample messages, complete and validate stream bindings, and serialize reusable profile YAML and task-contract artifacts. Goal and success criteria seed the phase vocabulary for operator validation. A profile therefore captures a robot's stream interface, while the task contract captures the meanings used to interpret its execution; both can be reused across routine cycles. This boundary keeps robot-specific naming and timing separate from semantic review and learner-specific data preparation.

3.4 Rollout Interpretation with Online VSA and Post-Rollout Review

Rollout interpretation is the transition where raw collection becomes lifecycle evidence. During collection, the operator benefits from timely feedback about phase, risk, and obvious failure. Before dataset admission, the system needs a more careful account of outcome, terminal state, failure phase, and training usefulness. RoboLineage therefore separates Online Visual Snapshot Agent (VSA) from asynchronous post-rollout review while preserving raw capture as the source of truth.

The mechanism uses three coordinated lanes (Figure 3). The raw capture lane records streams, operator events, and timing continuously, so model timeouts or review backlogs cannot block the source of truth. The Online Visual Snapshot Agent schedules sparse task-aware windows during collection and writes semantic anchors for feedback, later review, and uncertainty routing. After capture closes and queued VSA windows drain, the Post-Rollout Review Agent builds an evidence index and packetizes terminal state, post-terminal stability, temporal context, and operator/log context into a stricter final interpretation. Appendix D and Appendix E give the windowing and packet-review details.

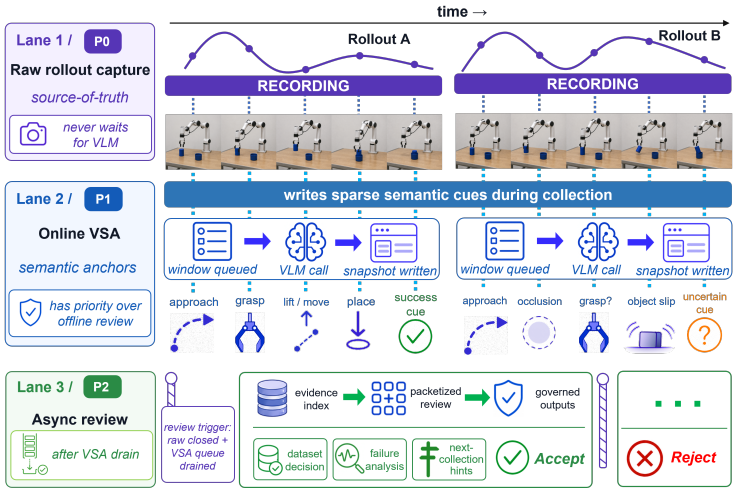


Figure 3: Rollout interpretation uses three lanes. Raw capture preserves the training source of truth, online VSA provides sparse semantic anchors during collection, and asynchronous post-rollout review turns evidence into governed lifecycle artifacts.

The review lane produces a final annotation, phase timeline, failure analysis, and dataset decision. The Data Governance Agent separates training eligibility, human-review priority, failure-pool routing, and exclusion reasons. Only rollouts marked `accepted_for_training` enter the primary training manifest. Useful failures and ambiguous successes remain linked as governed lineage evidence. Appendix B.5 describes the admission fields and override trace.

VSA supplies evidence anchors and has no dataset-admission authority. Schema-valid post-review with sufficient confidence can supersede an online phase label while retaining both records. Severe phase drift combined with low-confidence review triggers human inspection and withholds primary-training admission. Training eligibility and phase-label confidence are recorded separately, so a phase disagreement alone does not determine admission. A model may confuse similar objects or lose evidence under occlusion. A mismatched task contract or missing terminal observation can also lead to an incorrect review. Schema checks catch malformed responses, and failed requests are retried. Unresolved evidence failures receive failure or review records and remain outside primary training. Raw collection continues during asynchronous review.

3.5 Data Health, Training Integration, and Version Governance

The next lifecycle transition turns reviewed rollouts into a training dataset and then into a policy candidate. The Data Governance Agent first determines whether each rollout is eligible for training. Review priority and failure-pool routing remain separate from that decision. The Data Health Agent examines the resulting collection, using rejected and routed episodes as well as accepted data to identify weak task phases. Phase coverage and failure distributions show where data are sparse. Imbalance or missing evidence can leave a dataset unready even when individual examples were correctly admitted. The readiness artifact records these gaps and guides later recollection.

Training integration adapts this governed dataset to the host learner. The Framework Discovery Agent writes a framework profile from the target repository and the user's normal commands, and the Dataset Adapter Agent materializes accepted rollouts into that contract while preserving rollout identifiers, review artifacts, and hashes. Adapter outputs are validated before training. The Version Governance Agent freezes the selected episodes and adapter context in a dataset lock before training begins. The Training Monitor Agent records progress and metrics alongside checkpoint candidates, including warnings and failures. Policy metadata binds the chosen checkpoint to this lock and its parent policy; the framework profile and code revision identify how it was trained. The adapter checks camera, state, and action streams for modality and timestamp alignment without changing the learner's training command. These reusable artifacts make routine preparation distinct from the one-time work of integrating a new robot or training repository. Details are in Appendix B.5, Appendix B.6, Appendix B.7, Appendix B.8, and Appendix G.

3.6 Evaluation, Deployment Governance, and Master Synthesis

RoboLineage treats evaluation and deployment as agent-interpreted lifecycle stages. The Policy Evaluation Agent writes evaluation summaries from trial evidence: success, failure phase, safety issue, weak task phase, and regression against policy ancestry. The Deployment Governance Agent then converts the summary into the next lifecycle transition: deploy, hold, roll back, or collect more.

The Master Agent provides cross-iteration synthesis. It reads compact lineage artifacts in place of raw videos or full logs, then writes the current lifecycle state, a human-readable review, persistent memory, and a suggested next action. The Data Governance Agent handles rollout admission and the Data Health Agent handles data readiness; the Master Agent turns local lifecycle artifacts into cross-iteration memory and operator guidance. The cross-iteration node in Figure 2 carries this context forward, explaining what changed and which collection target the evidence supports. See Appendix B.9 and Appendix B.10.

3.7 Lightweight Implementation and Frontend

RoboLineage is implemented as a lightweight backend and frontend around existing robotics and learning stacks. The backend stores ordinary schema-validated artifacts with content hashes. The frontend lets operators inspect rollouts and resolve review decisions. It also exposes dataset readiness and follows training through evaluation to deployment review. The Master Agent view presents the next collection brief alongside the current iteration state. The release includes the runtime and schemas, with sample artifacts and scoring scripts for reproduction. Representative prompts docu-

ment the agents' contracts. ROS2-based robots can be adapted through profile-guided bindings for camera, state, action, and task-relevant streams. Appendix H describes the operator-facing flow; Appendix A lists the ordered components; Appendix A.3 describes the release package and reproduction boundaries, and summarizes the core prompt contracts.

4 Experiments

We evaluate RoboLineage around existing policy learners with three questions: (i) does the VSA/post-review pipeline improve rollout review and admission, (ii) does the lifecycle interface preserve policy quality while reducing routine pre-training labor, and (iii) can lineage-guided recommendations drive targeted recollection?

4.1 Experimental Setup

Experiments cover ARX with ACT-style action chunking, Realman with Diffusion Policy, and GALBOT G1 with a VLA/LeRobot-style workflow (Appendix Figure 9). The tasks are tabletop or visually grounded manipulation settings where cameras and robot state provide the main evidence. The review benchmark contains 25 tasks and 500 rollouts. The closed-loop case studies use three recollection rounds and an independent 60-trial final test. Labeling, task-contract, timing, and model-backend protocols are in Appendix I.

Table 1: VSA and post-rollout review over 25 tasks and 500 rollouts. All values are percentages. Outcome and failure-phase accuracy measure semantic review. Admission accuracy is exact agreement with expert consensus over normalized data-use routes across all 500 rollouts, before operator overrides. Leakage is the percentage of 194 expert-rejected rollouts admitted to the primary training manifest. Bold marks the best automated method. †Human agreement is a pre-consensus inter-annotator reference on the same 500-rollout benchmark, so leakage is not applicable.

Method	Outcome acc. ↑	Failure-phase acc. ↑	Admission acc. ↑	Leakage ↓
Direct video-to-VLM	78.4	58.0	79.6	16.0
VLM + schema only	82.6	63.4	84.0	11.3
VSA-only (no post-review)	84.0	68.0	85.6	9.8
Post-review only (no VSA)	86.6	72.0	88.4	7.2
No final observation / terminal focus	89.0	78.0	91.0	6.2
No packetized review	92.0	83.6	93.4	4.6
Full RoboLineage	**96.0**	**88.0**	**96.4**	**2.1**
Human agreement reference†	*97.0*	*91.0*	*98.0*	–

4.2 Review Reliability and Ablations

Table 1 isolates the review pipeline. Baselines use the same rollout pool and model route but remove the artifact structure, online anchors, packetization, or terminal evidence; Appendix I defines the annotation rubric and baseline controls, and Appendix J.1 summarizes the 25-task coverage.

Full RoboLineage reaches the best automated outcome and admission accuracy, the best automated failure-phase localization, and the lowest bad-data leakage. Direct video review is weakest on failure phase and false admission; schema-only review improves formatting but lacks event anchors; VSA-only review is useful for feedback and indexing but not strict enough for final admission. The post-review-only and evidence-ablation rows measure the semantic-evidence path: online anchors, packetized review, and terminal evidence improve review under the same model and rollout pool. Cross-round memory is evaluated separately in Section 4.4. Before operator overrides, the full pipeline matches consensus on 482 of 500 routes; 74 cases are routed for inspection and 18 later receive operator changes. Appendix I details baseline definitions, routing diagnostics, and containment cases (Tables 7, 8, and 9). Appendix J.1 gives the task roster, per-family and per-robot breakdowns, model-backend replay, and qualitative VSA/post-review plates (Tables 13, 14, 15, and 12; Figures 6–8).

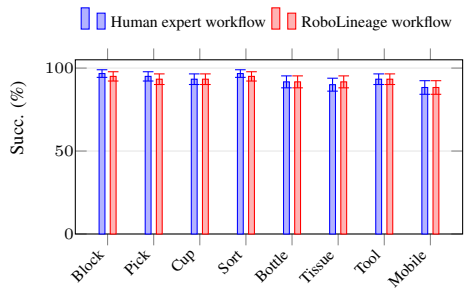


Figure 4: **Policy quality.** Each task uses a shared 100-rollout candidate pool. Each workflow trains on 60 selected demonstrations and evaluates on 60 trials per task; error bars show binomial standard error. Success counts agree on four tasks and differ by one trial on the other four. Mean success is 93.1% for the expert workflow and 92.7% for RoboLineage.

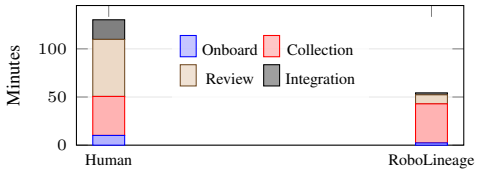


Figure 5: **Routine-cycle effort.** Mean active pre-training time per 30-rollout cycle over 12 matched sessions. RoboLineage reduces review and integration labor.

Table 2: **Runtime and API footprint.** Costs are estimated from logged token/image usage for one 30-rollout iteration.

Stage	When	Median	p90	Cost/cycle
VSA snapshot	During rollout	2.1 s	4.8 s	$6.10
Post-review	Async	18.4 s	34.0 s	$4.80
Summaries	After session	7.2 s	13.6 s	$0.70
30-rollout cycle	Mixed	–	–	$11.60

4.3 Policy Quality and Effort

We ask whether RoboLineage can reduce expert-mediated lifecycle labor while matching the policy quality of an expert-managed workflow in the measured setting. For each of eight tasks distinct from the closed-loop studies, we record a fixed 100-rollout candidate pool while RoboLineage's online VSA runs during collection. The expert-managed workflow and RoboLineage then independently select 60 accepted demonstrations from the same pool, using the same policy family, training command, and evaluation protocol. Mean success is 92.7% for RoboLineage and 93.1% for the expert-managed workflow, a mean difference of 0.4 percentage points (Figure 4). The workflows have equal success counts on four tasks; the remaining four differ by one trial out of 60. Three of these favor the expert workflow and one favors RoboLineage. These results show comparable policy performance when both workflows prepare training sets from the same candidate pool. Appendix I.3 gives protocol details.

Two trained operators run both workflows over 12 matched 30-rollout sessions, four per robot family, with condition order alternated after practice. The manual baseline uses the lab's maintained scripts, review sheets, and conversion workflow. Review includes full-video viewing, phase and transition marking, outcome/failure diagnosis, rationale, and training-manifest assembly: 59.2 minutes per session, or 1.97 minutes per rollout. Online VSA runs during recording, and asynchronous post-review overlaps subsequent collection; The 9.4-minute review stage in RoboLineage measures residual blocking work, uncertain-case inspection, and manifest confirmation. Active time also includes onboarding and framework preparation, with physical collection similar at 40.5 and 40.6 minutes. Non-collection effort falls from 89.7 to 13.7 minutes; total effort falls from 130.2 to 54.3 minutes, a 2.4× reduction. These routine timings follow one-time robot/learner integration and exclude GPU training time. Appendix I.4 and Tables 10 and 11 give scope and variance. Table 2 reports $11.60 in hosted model calls per cycle.

4.4 Closed-Loop Recollection Case Studies

Lineage-guided improvement turns failure analysis into data-collection action. We test this mechanism on two closed-loop real-robot tasks: cube stacking probes visual coverage gaps, and drawer opening probes contact-rich phase transitions. Each method is evaluated over three independent runs. In each run, three recollection rounds each add 30 accepted demonstrations and use 30 diagnostic trials. After the third round, the policy receives an independent 60-trial final test. The initial policy π_0 is also evaluated on 60 trials per run, for 180 trials per task. The heuristic counts recent failure phases; RoboLineage compares current failures with accepted data and cross-round lifecycle

state to decide what to collect next. RoboLineage tracks human-expert guidance and outperforms the other planners (Table 3). All planners receive the current diagnostic rollouts. The no-memory planner uses the same current VSA/post-review evidence. Its planning context is reset between rounds to exclude earlier lifecycle memory and policy ancestry.

The recommendations target the dominant failure mode at each round. In stacking, they move from out-of-distribution grasp locations to centered pre-release placement and then bounded-dwell release examples. Relative to the no-memory planner, final success increases from 70.0% to 83.3% in stacking and from 66.7% to 78.9% in drawer opening. Appendix I.7 gives the full recollection protocol, and Appendix J.2 gives the visual evidence plate and round-by-round recommendations.

Table 3: Closed-loop recollection trajectories. Each cell reports mean success rate (%) over three independent runs. Initial and final policies use 60 trials per run; Iter.1–3 use 30. Iter.1/Iter.2 appear on the first line and Iter.3/Final on the second. Phase denotes the failure-phase heuristic; No memory denotes the no-lineage-memory planner. Bold marks RoboLineage.

Task	Random	Phase	No memory	RoboLineage	Human expert
Stack	30.0/33.3	44.4/56.7	47.8/60.0	**52.2/68.9**	55.6/72.2
(π_0=25.0)	37.8/36.7	67.8/66.7	72.2/70.0	**84.4/83.3**	87.8/86.7
Drawer	31.1/34.4	42.2/53.3	45.6/57.8	**50.0/65.6**	53.3/68.9
(π_0=26.7)	37.8/37.8	63.3/62.2	67.8/66.7	**80.0/78.9**	83.3/81.1

4.5 Storage and Local Model Serving

Across the 500-rollout benchmark, raw records average 72.6 MiB per rollout. Structured lineage adds 68 KiB (0.09% of raw storage); the regenerable VSA keyframe cache adds 16.3 MiB (22.5%). We also replay the fixed 200-rollout backend subset locally with Qwen3-VL-32B-Instruct through vLLM on two RTX 4090 GPUs modified to 48 GB each, using the same prompts, contracts, schemas, and thresholds without retuning. Outcome, phase, and admission accuracy are 82.5%, 73.0%, and 83.5%; false accepts are 6/100 and review routes 53/200. Local serving provides an open-weight route with lower accuracy and more review than the strongest hosted backend. It removes per-call API fees; amortized cost depends on hardware utilization. Appendix I.5 and Table 12 give storage accounting and backend results.

5 Limitations and Discussion

RoboLineage reduces routine workflow effort and achieves policy quality comparable to expert-managed workflows under matched budgets. It also uses failure analysis to guide recollection. Our evidence comes from three configured robot/learner pairs on visually grounded manipulation tasks, with existing policy learners and lab safety procedures. Transfer to navigation or tasks with little visual evidence remains untested.

The ablations examine semantic review and memory across policy iterations. Provenance records make decisions traceable; dataset locks and deployment tickets govern training and release. We evaluate these mechanisms together and have not isolated their individual contributions.

Admission decisions can still go wrong if the model misreads an observation or the task contract is incomplete. Missing terminal evidence may also force a rollout into human review. As workload grows, the review queue could become a bottleneck. Sustained operation with thousands of rollouts therefore needs latency and backlog measurements. Distributed fleets and edge deployment have not been tested. Appendix K discusses these limitations and future directions.

Acknowledgments

This work is supported by the Early Career Scheme of the Research Grants Council (RGC) grant # 27207224, the HKU-100 Award, a donation from the Musketeers Foundation, and in part by the JC STEM Lab of Autonomous Intelligent Systems funded by The Hong Kong Jockey Club Charities Trust and computing resource provided by HKU-Shanghai Intelligent Computing Research Center.

We thank the robot operators and collaborators who supported data collection, system testing, and real-robot experiments.

References

[1] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi:10.15607/RSS.2023.XIX.016. URL https://doi.org/10.15607/RSS.2023.XIX.016.

[2] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. In K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi:10.15607/RSS.2023.XIX.026. URL https://doi.org/10.15607/RSS.2023.XIX.026.

[3] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, T. Jackson, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, K. Lee, S. Levine, Y. Lu, U. Malla, D. Manjunath, I. Mordatch, O. Nachum, C. Parada, J. Peralta, E. Perez, K. Pertsch, J. Quiambao, K. Rao, M. S. Ryoo, G. Salazar, P. R. Sanketi, K. Sayed, J. Singh, S. Sontakke, A. Stone, C. Tan, H. T. Tran, V. Vanhoucke, S. Vega, Q. Vuong, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich. RT-1: robotics transformer for real-world control at scale. In K. E. Bekris, K. Hauser, S. L. Herbert, and J. Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi:10.15607/RSS.2023.XIX.025. URL https://doi.org/10.15607/RSS.2023.XIX.025.

[4] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid, Q. Vuong, V. Vanhoucke, H. T. Tran, R. Soricut, A. Singh, J. Singh, P. Sermanet, P. R. Sanketi, G. Salazar, M. S. Ryoo, K. Reymann, K. Rao, K. Pertsch, I. Mordatch, H. Michalewski, Y. Lu, S. Levine, L. Lee, T. E. Lee, I. Leal, Y. Kuang, D. Kalashnikov, R. Julian, N. J. Joshi, A. Irpan, B. Ichter, J. Hsu, A. Herzog, K. Hausman, K. Gopalakrishnan, C. Fu, P. Florence, C. Finn, K. A. Dubey, D. Driess, T. Ding, K. M. Choromanski, X. Chen, Y. Chebotar, J. Carbajal, N. Brown, A. Brohan, M. G. Arenas, and K. Han. RT-2: vision-language-action models transfer web knowledge to robotic control. In J. Tan, M. Toussaint, and K. Darvish, editors, *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, Proceedings of Machine Learning Research, pages 2165–2183. PMLR, 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

[5] A. O'Neill, A. Rehman, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, A. Tung, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Gupta, A. E. Wang, A. Singh, A. Garg, A. Kembhavi, A. Xie, A. Brohan, A. Raffin, A. Sharma, A. Yavary, A. Jain, A. Balakrishna, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Wulfe, B. Ichter, C. Lu, C. Xu, C. Le, C. Finn, C. Wang, C. Xu, C. Chi, C. Huang, C. Chan, C. Agia, C. Pan, C. Fu, C. Devin, D. Xu, D. Morton, D. Driess, D. Chen, D. Pathak, D. Shah, D. Büchler, D. Jayaraman, D. Kalashnikov, D. Sadigh, E. Johns, E. P. Foster, F. Liu, F. Ceola, F. Xia, F. Zhao, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Feng, G. Schiavi, G. Berseth, G. Kahn, G. Wang, H. Su, H. Fang, H. Shi, H. Bao, H. B. Amor, H. I. Christensen, H. Furuta, H. Walke, H. Fang, H. Ha, I. Mordatch, I. Radosavovic,

I. Leal, J. Liang, J. Abou-Chakra, J. Kim, J. Drake, J. Peters, J. Schneider, J. Hsu, J. Bohg, J. T. Bingham, J. Wu, J. Gao, J. Hu, J. Wu, J. Wu, J. Sun, J. Luo, J. Gu, J. Tan, J. Oh, J. Wu, J. Lu, J. Yang, J. Malik, J. Silvério, J. Hejna, J. Booher, J. Tompson, J. Yang, J. Salvador, J. J. Lim, J. Han, K. Wang, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Black, K. Lin, K. Zhang, K. Ehsani, K. Lekkala, K. Ellis, K. Rana, K. Srinivasan, K. Fang, K. P. Singh, K. Zeng, K. Hatch, K. Hsu, L. Itti, L. Y. Chen, L. Pinto, L. Fei-Fei, L. Tan, L. J. Fan, L. Ott, L. Lee, L. Weihs, M. Chen, M. Lepert, M. Memmel, M. Tomizuka, M. Itkina, M. G. Castro, M. Spero, M. Du, M. Ahn, M. C. Yip, M. Zhang, M. Ding, M. Heo, M. K. Srirama, M. Sharma, M. J. Kim, N. Kanazawa, N. Hansen, N. Heess, N. J. Joshi, N. Sünderhauf, N. Liu, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, O. Bastani, P. R. Sanketi, P. T. Miller, P. Yin, P. Wohlhart, P. Xu, P. D. Fagan, P. Mitrano, P. Sermanet, P. Abbeel, P. Sundaresan, Q. Chen, Q. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Martín-Martín, R. Baijal, R. Scalise, R. Hendrix, R. Lin, R. Qian, R. Zhang, R. Mendonca, R. Shah, R. Hoque, R. Julian, S. Bustamante-Gomez, S. Kirmani, S. Levine, S. Lin, S. Moore, S. Bahl, S. Dass, S. D. Sonawani, S. Song, S. Xu, S. Haldar, S. Karamcheti, S. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Ramamoorthy, S. Dasari, S. Belkhale, S. Park, S. Nair, S. Mirchandani, T. Osa, T. Gupta, T. Harada, T. Matsushima, T. Xiao, T. Kollar, T. Yu, T. Ding, T. Davchev, T. Z. Zhao, T. Armstrong, T. Darrell, T. Chung, V. Jain, V. Vanhoucke, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Wang, X. Zhu, X. Geng, X. Liu, L. Xu, X. Li, Y. Lu, Y. J. Ma, Y. Kim, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Wu, Y. Xu, Y. Wang, Y. Bisk, Y. Cho, Y. Lee, Y. Cui, Y. Cao, Y. Wu, Y. Tang, Y. Zhu, Y. Zhang, Y. Jiang, Y. Li, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Ma, Z. Xu, Z. J. Cui, Z. Zhang, and Z. Lin. Open x-embodiment: Robotic learning datasets and RT-X models : Open x-embodiment collaboration. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, pages 6892–6903. IEEE, 2024. doi:10.1109/ICRA57147.2024.10611477. URL https://doi.org/10.1109/ICRA57147.2024.10611477.

[6] A. Khazatsky, K. Pertsch, S. Nair, A. Balakrishna, S. Dasari, S. Karamcheti, S. Nasiriany, M. K. Srirama, L. Y. Chen, K. Ellis, P. D. Fagan, J. Hejna, M. Itkina, M. Lepert, Y. J. Ma, P. T. Miller, J. Wu, S. Belkhale, S. Dass, H. Ha, A. Jain, A. Lee, Y. Lee, M. Memmel, S. Park, I. Radosavovic, K. Wang, A. Zhan, K. Black, C. Chi, K. B. Hatch, S. Lin, J. Lu, J. Mercat, A. Rehman, P. R. Sanketi, A. Sharma, C. Simpson, Q. Vuong, H. R. Walke, B. Wulfe, T. Xiao, J. H. Yang, A. Yavary, T. Z. Zhao, C. Agia, R. Baijal, M. G. Castro, D. Chen, Q. Chen, T. Chung, J. Drake, E. P. Foster, J. Gao, D. A. Herrera, M. Heo, K. Hsu, J. Hu, D. Jackson, C. Le, Y. Li, R. Lin, Z. Ma, A. Maddukuri, S. Mirchandani, D. Morton, T. Nguyen, A. O'Neill, R. Scalise, D. Seale, V. Son, S. Tian, E. Tran, A. E. Wang, Y. Wu, A. Xie, J. Yang, P. Yin, Y. Zhang, O. Bastani, G. Berseth, J. Bohg, K. Goldberg, A. Gupta, A. Gupta, D. Jayaraman, J. J. Lim, J. Malik, R. Martín-Martín, S. Ramamoorthy, D. Sadigh, S. Song, J. Wu, M. C. Yip, Y. Zhu, T. Kollar, S. Levine, and C. Finn. DROID: A large-scale in-the-wild robot manipulation dataset. In D. Kulic, G. Venture, K. E. Bekris, and E. Coronado, editors, *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi:10.15607/RSS.2024.XX.120. URL https://doi.org/10.15607/RSS.2024.XX.120.

[7] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. P. Foster, P. R. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn. Openvla: An open-source vision-language-action model. In P. Agrawal, O. Kroemer, and W. Burgard, editors, *Conference on Robot Learning, 6-9 November 2024, Munich, Germany*, Proceedings of Machine Learning Research, pages 2679–2713. PMLR, 2024. URL https://proceedings.mlr.press/v270/kim25c.html.

[8] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, Proceedings of Machine Learning Research, pages 885–897. PMLR, 2019. URL http://proceedings.mlr.press/v100/dasari20a.html.

[9] H. Fang, H. Fang, Z. Tang, J. Liu, C. Wang, J. Wang, H. Zhu, and C. Lu. RH20T: A comprehensive robotic dataset for learning diverse skills in one-shot. In *IEEE International Conference on Robotics and Automation, ICRA 2024, Yokohama, Japan, May 13-17, 2024*, pages 653–660. IEEE, 2024. doi:10.1109/ICRA57147.2024.10611615. URL https://doi.org/10.1109/ICRA57147.2024.10611615.

[10] F. Lin, Y. Hu, P. Sheng, C. Wen, J. You, and Y. Gao. Data scaling laws in imitation learning for robotic manipulation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=pISLZG7ktL.

[11] D. Ghosh, H. R. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, J. Luo, Y. L. Tan, L. Y. Chen, Q. Vuong, T. Xiao, P. R. Sanketi, D. Sadigh, C. Finn, and S. Levine. Octo: An open-source generalist robot policy. In D. Kulic, G. Venture, K. E. Bekris, and E. Coronado, editors, *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi:10.15607/RSS.2024.XX.090. URL https://doi.org/10.15607/RSS.2024.XX.090.

[12] H. R. Walke, K. Black, T. Z. Zhao, Q. Vuong, C. Zheng, P. Hansen-Estruch, A. W. He, V. Myers, M. J. Kim, M. Du, A. Lee, K. Fang, C. Finn, and S. Levine. Bridgedata V2: A dataset for robot learning at scale. In J. Tan, M. Toussaint, and K. Darvish, editors, *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, Proceedings of Machine Learning Research, pages 1723–1736. PMLR, 2023. URL https://proceedings.mlr.press/v229/walke23a.html.

[13] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of household tasks for generalist robots. In D. Kulic, G. Venture, K. E. Bekris, and E. Coronado, editors, *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi:10.15607/RSS.2024.XX.050. URL https://doi.org/10.15607/RSS.2024.XX.050.

[14] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei. ROBOTURK: A crowdsourcing platform for robotic skill learning through imitation. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, Proceedings of Machine Learning Research, pages 879–893. PMLR, 2018. URL http://proceedings.mlr.press/v87/mandlekar18a.html.

[15] P. Wu, Y. Shentu, Z. Yi, X. Lin, and P. Abbeel. GELLO: A general, low-cost, and intuitive teleoperation framework for robot manipulators. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2024, Abu Dhabi, United Arab Emirates, October 14-18, 2024*, pages 12156–12163. IEEE, 2024. doi:10.1109/IROS58592.2024.10801581. URL https://doi.org/10.1109/IROS58592.2024.10801581.

[16] C. Chi, Z. Xu, C. Pan, E. Cousineau, B. Burchfiel, S. Feng, R. Tedrake, and S. Song. Universal manipulation interface: In-the-wild robot teaching without in-the-wild robots. In D. Kulic, G. Venture, K. E. Bekris, and E. Coronado, editors, *Robotics: Science and Systems XX, Delft, The Netherlands, July 15-19, 2024*, 2024. doi:10.15607/RSS.2024.XX.045. URL https://doi.org/10.15607/RSS.2024.XX.045.

[17] S. Chen, C. Wang, K. Nguyen, L. Fei-Fei, and C. K. Liu. Arcap: Collecting high-quality human demonstrations for robot learning with augmented reality feedback. In *IEEE International Conference on Robotics and Automation, ICRA 2025, Atlanta, GA, USA, May 19-23, 2025*, pages 8291–8298. IEEE, 2025. doi:10.1109/ICRA55743.2025.11128717. URL https://doi.org/10.1109/ICRA55743.2025.11128717.

[18] J. Fang, W. Chen, H. Xue, F. Zhou, T. Le, Y. Wang, Y. Zhang, J. Lv, C. Wen, and C. Lu. Robopocket: Improve robot policies instantly with your phone, 2026. URL https://arxiv.org/abs/2603.05504.

[19] A. Mandlekar, D. Xu, J. Wong, S. Nasiriany, C. Wang, R. Kulkarni, L. Fei-Fei, S. Savarese, Y. Zhu, and R. Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In A. Faust, D. Hsu, and G. Neumann, editors, *Conference on Robot Learning, 8-11 November 2021, London, UK*, Proceedings of Machine Learning Research, pages 1678–1690. PMLR, 2021. URL https://proceedings.mlr.press/v164/mandlekar22a.html.

[20] A. Mandlekar, S. Nasiriany, B. Wen, I. Akinola, Y. Narang, L. Fan, Y. Zhu, and D. Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In J. Tan, M. Toussaint, and K. Darvish, editors, *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, Proceedings of Machine Learning Research, pages 1820–1864. PMLR, 2023. URL https://proceedings.mlr.press/v229/mandlekar23a.html.

[21] S. Belkhale, Y. Cui, and D. Sadigh. Data quality in imitation learning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/fe692980c5d9732cf153ce27947653a7-Abstract-Conference.html.

[22] C. Agia, R. Sinha, J. Yang, R. Antonova, M. Pavone, H. Nishimura, M. Itkina, and J. Bohg. Cupid: Curating data your robot loves with influence functions. In J. Lim, S. Song, and H.-W. Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 2907–2932. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/agia25a.html.

[23] S. Dass, A. Khaddaj, L. Engstrom, A. Madry, A. Ilyas, and R. Martín-Martín. Datamil: Selecting data for robot imitation learning with datamodels, 2025. URL https://arxiv.org/abs/2505.09603.

[24] J. Hejna, S. Mirchandani, A. Balakrishna, A. Xie, A. Wahid, J. Tompson, P. R. Sanketi, D. Shah, C. M. Devin, and D. Sadigh. Robot Data Curation with Mutual Information Estimators. In *Proceedings of Robotics: Science and Systems*, LosAngeles, CA, USA, June 2025. doi:10.15607/RSS.2025.XXI.023.

[25] Y. Zhang, Y. Xie, H. Liu, R. Shah, M. Wan, L. Fan, and Y. Zhu. Scizor: A self-supervised approach to data curation for large-scale imitation learning, 2025. URL https://arxiv.org/abs/2505.22626.

[26] B. Ichter, A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, D. Kalashnikov, S. Levine, Y. Lu, C. Parada, K. Rao, P. Sermanet, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, M. Yan, N. Brown, M. Ahn, O. Cortes, N. Sievers, C. Tan, S. Xu, D. Reyes, J. Rettinghouse, J. Quiambao, P. Pastor, L. Luu, K. Lee, Y. Kuang, S. Jesmonth, N. J. Joshi, K. Jeffrey, R. J. Ruano, J. Hsu, K. Gopalakrishnan, B. David, A. Zeng, and C. K. Fu. Do as I can, not as I say: Grounding language in robotic affordances. In K. Liu, D. Kulic, and J. Ichnowski, editors, *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, Proceedings of Machine Learning Research, pages 287–318. PMLR, 2022. URL https://proceedings.mlr.press/v205/ichter23a.html.

[27] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, P. Sermanet, T. Jackson, N. Brown, L. Luu, S. Levine, K. Hausman, and B. Ichter. Inner monologue: Embodied reasoning through planning with language models. In K. Liu,

D. Kulic, and J. Ichnowski, editors, *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, Proceedings of Machine Learning Research, pages 1769–1782. PMLR, 2022. URL https://proceedings.mlr.press/v205/huang23c.html.

[28] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 9493–9500. IEEE, 2023. doi:10.1109/ICRA48891.2023.10160591. URL https://doi.org/10.1109/ICRA48891.2023.10160591.

[29] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. In J. Tan, M. Toussaint, and K. Darvish, editors, *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, Proceedings of Machine Learning Research, pages 540–562. PMLR, 2023. URL https://proceedings.mlr.press/v229/huang23b.html.

[30] D. Driess, F. Xia, M. S. M. Sajjadi, C. Lynch, A. Chowdhery, B. Ichter, A. Wahid, J. Tompson, Q. Vuong, T. Yu, W. Huang, Y. Chebotar, P. Sermanet, D. Duckworth, S. Levine, V. Vanhoucke, K. Hausman, M. Toussaint, K. Greff, A. Zeng, I. Mordatch, and P. Florence. Palm-e: An embodied multimodal language model. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, Proceedings of Machine Learning Research, pages 8469–8488. PMLR, 2023. URL https://proceedings.mlr.press/v202/driess23a.html.

[31] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2026. URL https://arxiv.org/abs/2410.24164.

[32] K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. R. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, b. ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In J. Lim, S. Song, and H.-W. Park, editors, *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pages 17–40. PMLR, 27–30 Sep 2025. URL https://proceedings.mlr.press/v305/black25a.html.

[33] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, K. Xu, H. Su, and J. Zhu. RDT-1B: a diffusion foundation model for bimanual manipulation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=yAzN4tz7oI.

[34] M. J. Kim, C. Finn, and P. Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

[35] J. Duan, W. Pumacay, N. Kumar, Y. R. Wang, S. Tian, W. Yuan, R. Krishna, D. Fox, A. Mandlekar, and Y. Guo. AHA: A vision-language-model for detecting and reasoning over failures in robotic manipulation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=JVkdSi7Ekg.

[36] R. Li, Y. Zhou, Y. Zhu, K. Chen, J. Wang, S. Wang, K. Hu, M. Yu, B. Jiang, Z. Su, J. Ma, X. He, Y. Shen, Y. Yang, G. Ren, M. Yao, W. Wang, and Y. Mu. Roboclaw: An agentic framework for scalable long-horizon robotic tasks, 2026. URL https://arxiv.org/abs/2603.11558.

[37] W. Liang, S. Wang, H.-J. Wang, O. Bastani, Y. J. Ma, and D. Jayaraman. Tether: Autonomous functional play with correspondence-driven trajectory warping, 2026. URL https://arxiv.org/abs/2603.03278.

[38] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J. Crespo, and D. Dennison. Hidden technical debt in machine learning systems. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2503–2511, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/86df7dcfd896fcaf2674f757a2463eba-Abstract.html.

[39] N. Sambasivan, S. Kapania, H. Highfill, D. Akrong, P. K. Paritosh, and L. Aroyo. "everyone wants to do the model work, not the data work": Data cascades in high-stakes AI. In Y. Kitamura, A. Quigley, K. Isbister, T. Igarashi, P. Bjørn, and S. M. Drucker, editors, *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, pages 39:1–39:15. ACM, 2021. doi:10.1145/3411764.3445518. URL https://doi.org/10.1145/3411764.3445518.

[40] D. Baylor, E. Breck, H. Cheng, N. Fiedel, C. Y. Foo, Z. Haque, S. Haykal, M. Ispir, V. Jain, L. Koc, C. Y. Koo, L. Lew, C. Mewald, A. N. Modi, N. Polyzotis, S. Ramesh, S. Roy, S. E. Whang, M. Wicke, J. Wilkiewicz, X. Zhang, and M. Zinkevich. TFX: A tensorflow-based production-scale machine learning platform. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*, pages 1387–1395. ACM, 2017. doi:10.1145/3097983.3098021. URL https://doi.org/10.1145/3097983.3098021.

[41] M. Zaharia, A. Chen, A. Davidson, A. Ghodsi, S. A. Hong, A. Konwinski, S. Murching, T. Nykodym, P. Ogilvie, M. Parkhe, F. Xie, and C. Zumar. Accelerating the machine learning lifecycle with mlflow. *IEEE Data Eng. Bull.*, 41(4):39–45, 2018. URL http://sites.computer.org/debull/A18dec/p39.pdf.

[42] M. Vartak. MODELDB: A system for machine learning model management. In *8th Biennial Conference on Innovative Data Systems Research, CIDR 2017, Chaminade, CA, USA, January 8-11, 2017, Online Proceedings*. www.cidrdb.org, 2017. URL http://cidrdb.org/cidr2017/gongshow/abstracts/cidr2017_112.pdf.

[43] M. Schlegel and K. Sattler. Management of machine learning lifecycle artifacts: A survey. *SIGMOD Rec.*, 51(4):18–35, 2022. doi:10.1145/3582302.3582306. URL https://doi.org/10.1145/3582302.3582306.

[44] R. Souza, P. Valduriez, M. Mattoso, R. Cerqueira, M. A. S. Netto, L. Azevedo, V. Lourenço, E. F. de Souza Soares, R. Thiago, R. Brandão, D. Civitarese, E. V. Brazil, and M. F. Moreno. Provenance data in the machine learning lifecycle in computational science and engineering. In *2019 IEEE/ACM Workflows in Support of Large-Scale Science, WORKS@SC 2019, Denver, CO, USA, November 17, 2019*, pages 1–10. IEEE, 2019. doi:10.1109/WORKS49585.2019.00006. URL https://doi.org/10.1109/WORKS49585.2019.00006.

[45] M. Schlegel and K. Sattler. Capturing end-to-end provenance for machine learning pipelines. *Inf. Syst.*, 132:102495, 2025. doi:10.1016/J.IS.2024.102495. URL https://doi.org/10.1016/j.is.2024.102495.

[46] T. Gebru, J. Morgenstern, B. Vecchione, J. W. Vaughan, H. M. Wallach, H. D. III, and K. Crawford. Datasheets for datasets. *Commun. ACM*, 64(12):86–92, 2021. doi:10.1145/3458723. URL https://doi.org/10.1145/3458723.

[47] M. Mitchell, S. Wu, A. Zaldivar, P. Barnes, L. Vasserman, B. Hutchinson, E. Spitzer, I. D. Raji, and T. Gebru. Model cards for model reporting. In danah boyd and J. H. Morgenstern, editors, *Proceedings of the Conference on Fairness, Accountability, and Transparency, FAT* 2019, Atlanta, GA, USA, January 29-31, 2019*, pages 220–229. ACM, 2019. doi:10.1145/3287560.3287596. URL https://doi.org/10.1145/3287560.3287596.

A System Implementation Details

Agent	Role	Main lineage artifact
Robot Onboarding	Normalize embodiment profile, streams, frames, and capabilities	`robot.profile.yaml`
Task Config	Define phases, success criteria, risks, and collection policy	`task.config.json`
Online Visual Snapshot	Convert sparse execution events into stable semantic evidence	`snapshot.jsonl`, keyframes
Post-Rollout Review	Aggregate multimodal evidence into outcome, phase timeline, failure analysis, and L1 annotation	`annotation.final.json`
Data Governance	Route rollouts to training, review, failure pool, or rejection	`dataset_decision.json`
Data Health	Summarize coverage, phase balance, missing evidence, and failure distribution	`dataset.health.json`
Framework Discovery	Discover training command, data contract, and adapter path	`framework.profile.json`
Dataset Adapter	Materialize accepted rollouts for the target learner	`dataset_manifest.json`
Training Monitor	Track training progress, errors, and metrics	`train.run.json`
Version Governance	Lock the dataset and bind policy metadata	`dataset.lock.json`, `policy.meta.json`
Policy Evaluation	Review evaluation trials and aggregate quality	`eval.summary.json`
Deployment Governance	Recommend deploy, hold, rollback, or collect more	`deploy.ticket.json`
Master Agent	Maintain lifecycle state, memory, risks, and next-collection summaries	`master.state.json`, `master.memory.jsonl`, `master.review.json`

Table 4: Ordered agent list. The main paper introduces the lifecycle contract; this appendix records the implementation-level roles and artifacts.

A.1 Artifact Store

RoboLineage stores lifecycle state as structured artifacts with explicit schemas and hashes. This format fits robotics labs: artifacts can be versioned, inspected, copied with datasets, and debugged with ordinary tools. Each artifact records a schema version, creation time, parent identifiers, upstream content hashes, and a compact human-readable summary. The most common artifacts are:

- `robot.profile.yaml`: robot embodiment, topic bindings, cameras, state streams, action interfaces, frames, safety limits, and onboarding warnings.
- `task.config.json`: task phases, success criteria, failure criteria, risk events, snapshot policy, collection policy, and admission rules.
- `snapshot.jsonl`: online VSA snapshots, including event window, selected keyframes, predicted phase, progress, risk flags, uncertainty, and evidence references.
- `annotation.final.json`: final rollout outcome, phase timeline, L1 annotation, failure analysis, evidence packets, and review confidence.
- `dataset_decision.json`: structured admission decision, reason codes, review routing, training eligibility, and failure-pool routing.
- `dataset.health.json`: coverage, imbalance, missing data, failure distribution, and dataset-level warnings.
- `framework.profile.json`: discovered training entry point, expected data contract, command templates, adapter strategy, and repository notes.

- `dataset.lock.json`: immutable manifest of episodes, artifact hashes, adapter version, and task/robot hashes used by a training run.
- `policy.meta.json`: policy identifier, parent policy, checkpoint hash, dataset lock hash, training metrics, code revision, and framework profile.
- `eval.summary.json`: evaluation trials, success metrics, failure phases, safety issues, regressions, and evidence links.
- `deploy.ticket.json`: advisory deployment recommendation, threshold checks, approval state, policy hash, and rollback target.

A.2 Agent Contract Pattern

All agents follow a common contract: structured inputs, schema-validated outputs, explicit uncertainty, and a declared lifecycle scope. An agent may call a multimodal or language model, and the result enters lineage only after schema validation. Across the system, model outputs provide semantic perception, agents translate that perception into task-grounded interpretation, and lineage artifacts carry the versioned governance state.

A.3 Open-Source Release and Key Prompt Contracts

The release includes the backend runtime and frontend console. Schemas define the artifacts they exchange, and representative prompts document the agents' contracts. Tests and sample artifacts support inspection; analysis scripts regenerate the paper tables. Hardware profiles and training-framework examples serve as replaceable adapters. The stable release boundary is the lifecycle artifact contract.

The scoring scripts in the package regenerate the reported metrics from frozen labels, frozen model outputs, sampled keyframes, and hashed raw manifests, without requiring private full-resolution videos. Re-running semantic agents from raw evidence requires an equivalent model route and API access, so the package separates artifact-level reproduction from model-backend reproduction. From released artifacts, reviewers can regenerate the aggregate review tables, timing summaries, model-call logs, and closed-loop tables. Re-training policies, re-running robots, or auditing every full-resolution video requires local hardware or access to private lab data.

Package item	Contents and purpose
Backend runtime	Session service, raw capture orchestration, online VSA, post-rollout review, dataset governance, training integration, evaluation, deployment governance, and Master Agent code.
Frontend console	Operator-facing lifecycle views for onboarding, rollout capture, review, dataset health, training integration, and master summaries.
Artifact schemas	JSON/YAML schemas for snapshots, annotations, dataset decisions, dataset locks, policy metadata, evaluation summaries, deployment tickets, and next-collection briefs.
Prompts	Frozen representative prompt templates for VSA, post-rollout packet review, Data Health Agent interpretation, framework discovery, deployment governance, and master summaries.
Sample records	A minimal lifecycle trace from rollout capture through review, dataset admission, dataset lock, policy metadata, evaluation summary, and deployment recommendation.
Frontend and packet examples	Review-screen screenshots, evidence packet excerpts, operator override records, and the corresponding artifact JSON for the same rollouts.
Benchmark evidence	Consensus labels, sampled keyframes, evidence packets, and packet-review outputs for reproducing review metrics without redistributing full robot videos.
Tests and scripts	Unit tests for contracts and agents, plus scripts for semantic-review scoring, effort aggregation, and closed-loop case tables.

Table 5: Contents of the open-source RoboLineage release.

Online VSA prompt contract. The online VSA prompt binds a short image window to the task contract, current rollout memory, and auxiliary action signals. The prompt asks the VLM to choose one configured phase and return a compact JSON assessment.

Example artifact	Representative content	Downstream use
rollout-042/raw_manifest.json	camera stream, robot-state stream, operator stop event, frame hashes	source of truth for review and training adapters
rollout-042/snapshot.jsonl	five VSA windows: approach, grasp, lift, place, final observation	semantic anchors for packetized post-review
`rollout-042/review_packet_terminal.json`	terminal frames, VSA phase anchors, stop reason, confidence, cited frame IDs	concrete evidence shown in the review screen and consumed by post-rollout review
annotation.final.json	outcome = failure, phase = grasp, evidence = gripper-center offset	input to dataset admission and failure analysis
dataset_decision.json	decision = failure-pool candidate, accepted_for_training=false	prevents bad-data leakage while preserving failure evidence
dataset.health.json	OOD grasp-location gap, undercovered start bins	target for next recollection
next_collection_brief.json	collect successful grasps in underrepresented table regions	operator-facing action for the next rollout batch
policy.meta.json	parent policy, dataset lock, adapter hash, checkpoint hash	policy ancestry for evaluation and deployment tickets

Table 6: A lifecycle trace included in the open-source release. The example follows one rollout-level failure as it becomes a failure-pool artifact, Data Health Agent gap, next-collection action, and policy-ancestry record.

Role. Embodied visual assessment for robot task execution monitoring.
Inputs. Task description; allowed phases; phase definitions and visual signatures; known failure signals; recent rollout memory; event type; anchor frame; timestamps; selected keyframes; optional gripper, end-effector, stillness, and motion summaries.
Rules. Use visual evidence for object state, contact, placement, and failure risk. Use action signals to refine timing and disambiguate contact, release, retry, or stall. Select phases only from the task contract, and keep the sequence temporally coherent with previous snapshots.
JSON fields. `phase`, `progress`, `risk_level`, `imminent_failure`, `needs_review`, and `confidence`.

Post-rollout review prompt contract. Post-rollout review is scoped to evidence interpretation. It reviews selected evidence packets and returns local visual support that the aggregator combines with deterministic timelines and failure rules.

Single-prompt path. Return final success, confidence, final phase, terminal evidence, phase corrections, retry/failure evidence, label quality, training usability, and concise reasoning.
Packet path. For terminal, post-terminal, or temporal packets, return packet terminal state, terminal intactness, post-terminal status, confidence, local final phase, evidence frames, retry/failure evidence, and reasoning.
Rules. Treat retry as an event. Leave dataset admission to the governance stage. Advisory `training_usability` is later separated from `accepted_for_training`. If visual evidence is missing, report insufficient evidence.

Robot onboarding prompt contract. The onboarding prompt turns a newly attached robot profile into a compact operator-facing explanation of the available streams, frames, capabilities, and warnings. It does not invent bindings; it explains and checks the profile that will become the lifecycle entry point.

Inputs. Normalized robot profile, operator note, deterministic warnings, and deterministic assumptions.
Output. Profile summary, binding explanation, warnings, assumptions, and recommended validation checks.
Rule. Use only the ROS topics, streams, frames, and capabilities in the supplied profile. Profile activation and signal validation remain runtime operations.

Data Governance Agent and Data Health Agent prompt contract. These prompts operate after semantic review has produced evidence. Data Governance records rollout-level routing, while Data Health summarizes the dataset-level gaps that matter for training readiness and recollection.

Inputs. Selected rollouts, post-review artifacts, dataset history, task phases, deterministic coverage report, and failure-type counts.
Output. Dataset health summary, coverage notes, risk notes, recommended collection object, and confidence.
Rule. Explain missing phase coverage, imbalance, and targeted failure collection needs using only supplied artifacts. The deterministic report remains the auditable source for training readiness.

Framework Discovery Agent and Training Monitor Agent prompt contracts. Framework discovery translates a training repository and the user's normal commands into a lifecycle adapter plan. Training monitoring then explains logged training state without changing the training code or policy checkpoint.

Framework Discovery Agent inputs. Repository tree, user-provided dataset/train/eval commands, command context, inferred target data contract, deep-inspection signals, and selected code snippets.
Framework Discovery Agent output. Framework type, target dataset expectation, adapter plan, training/evaluation entry points, output locations, log patterns, assumptions, and warnings.
Training Monitor Agent output. Diagnosis, likely causes, recommended action, operator brief, and confidence derived from deterministic log metrics and a bounded log excerpt. The monitor explains the deterministic status artifact.

Policy Evaluation Agent, Deployment Governance Agent, and Master Agent prompt contracts. These prompts close an iteration. Evaluation produces policy-level evidence, Deployment Governance turns that evidence into an auditable recommendation, and the Master Agent maintains the cross-iteration summary used for the next cycle.

Policy Evaluation Agent. Reuse the post-rollout evidence path, then write policy-evaluation and collection-recommendation artifacts.
Deployment Governance Agent. Read deterministic evaluation summaries, deployment decisions, collection recommendations, and next-collection briefs; return diagnostic risk notes, an operator brief, confidence, and an interpretation of the lifecycle transition.
Master Agent. Read compact lifecycle artifacts and return a global summary, operator brief, risk interpretation, suggested next action, and memory updates. The output is reviewable lifecycle state.

A.4 Collection Workflow Boundary

The evaluated implementation focuses on governed rollout capture. Raw streams are recorded as source evidence, event-triggered VSA snapshots provide online semantic anchors, and the Post-Rollout Review Agent writes the authoritative annotation. The task config also records collection policy, so future correction and deployment-time collection modes can reuse the same lineage contract. We leave those richer modes to future work.

B Detailed Agent Descriptions

B.1 Robot Onboarding Agent

Role. The Robot Onboarding Agent normalizes robot-specific facts into a common profile. It is the robot-side analogue of the Framework Discovery Agent: local interface assumptions become a portable lifecycle contract.

Inputs and output. The agent reads a YAML profile draft serialized from the operator's UI task/topic inputs, together with deployment notes, calibration metadata, and stream descriptions. It completes and validates bindings and emits a reusable robot profile, validation report, warnings, and optional binding explanation used by VSA, post-review, dataset conversion, evaluation, deployment governance, and the frontend.

Profile-guided binding procedure. The onboarding pass normalizes the supplied profile, checks required fields, and validates semantic bindings for camera, state, action, gripper, contact, health, and optional operator-event channels. For ROS-based systems, the profile records topic names,

message types, transport, namespace, domain id, and field mappings for runtime binding. The agent explains binding risks and recommends checks; the runtime remains profile-guided.

Operator correction. The agent records assumptions and warnings, exposes suspicious or incomplete bindings to the frontend, and waits for profile activation and signal validation before downstream collection. Corrected bindings are written back into the robot profile, so later rollouts are interpreted against the contract actually used in the session.

Profile contents. The profile records embodiment type, controllable joints or end-effector interfaces, camera streams, state topics, action topics, gripper or contact signals, coordinate frames, safety limits, stream synchronization status, and warnings.

Lifecycle behavior. A data-collection operator can approve or correct bindings in the UI. Downstream modules consume normalized fields, with robot-specific topic names and file layouts resolved by the profile. Topic bindings are stored as data in the onboarding artifact: when a robot exposes a different camera name or state channel, the artifact changes and the rest of the lifecycle keeps the same interface.

Failure traceability. Onboarding is part of lineage. If a later policy fails because a stream was missing, mis-bound, or low confidence, the team can trace the failure to the profile used by that lifecycle run. Multi-robot portability comes from making robot differences explicit and versioned.

B.2 Task Config Agent

Role. The Task Config Agent converts task intent into operational lifecycle rules. It defines success, risk, failure, and training usefulness for a specific policy iteration.

Contract contents. The task artifact records phases, terminal conditions, risk events, snapshot priorities, dataset admission criteria, required sensors, collection policy, default review tolerance, admissible recovery patterns, and labels used by dataset health.

Generation context. The task configuration is generated from the operator's task description, current camera view, and robot/task context. The image matters: a visible drawer handle, an occluded bin, or a deformable object can change phase definitions, visual hints, and failure signals.

Artifact consumers. The same task contract is read by multiple downstream agents. VSA uses phases and snapshot priorities to decide where to look online. The Post-Rollout Review Agent uses terminal conditions, risk events, and admissible recovery patterns to choose packet types and aggregation rules. The Data Governance Agent uses admission criteria to route rollouts into training, review, rejection, or failure pools. The Data Health Agent uses the task labels to report coverage and next-collection gaps.

Task-specific evidence. A placement task should over-sample release and final-settle events, while a drawer task should emphasize contact onset, handle engagement, and motion after contact. A manipulation task may define phases such as approach, grasp, transfer, place, release, and final observation.

Phase vocabulary. The phase list is treated as a closed vocabulary for VSA, post-review, and evaluation. Retry, correction, and intervention are represented as events or failure/recovery evidence. This keeps phase statistics comparable across rollout review, dataset health, and policy evaluation.

Training semantics. The Task Config Agent also specifies admission behavior. For example, recovered failures can enter a separate failure pool while staying out of the primary imitation set by default. The same rules are read by the VSA scheduler, post-rollout packet builder, Data Governance Agent, and Master Agent.

Why explicit task rules matter. The same visual trace can mean different things across tasks. A brief contact may be success evidence for drawer opening, a risk signal for cup placement, or irrelevant for navigation. The Task Config Agent is where semantic perception becomes task-grounded governance.

B.3 Online Visual Snapshot Agent

The VSA has four internal submodules:

1. **Window Scheduler.** Converts event streams into bounded visual windows, coalesces redundant events, and protects critical windows.
2. **Prompt Builder and Parser.** Constructs task-aware multimodal prompts and parses the response into phase, progress, risk, evidence, and confidence fields.
3. **Phase State Machine.** Enforces monotonic phase progression by default, permits evidence-supported jumps, and handles retry after terminal failure.
4. **Temporal Stabilizer.** Smooths jitter across adjacent windows and marks conflicts as review-worthy.

Inputs. The VSA reads the robot profile, task config, rollout metadata, recent robot events, raw image buffers, operator events, and previous stable phase state. It can run before the full rollout finishes, creating sparse semantic anchors while data collection is happening.

Outputs. Each snapshot artifact contains the window identifier, anchor time, event reason, selected frame references, predicted phase, progress summary, risk flags, visible evidence, confidence, review flag, parser status, and links back to the rollout and task contract.

Runtime priority and permissions. Online VSA is prioritized during rollout collection. Offline review, evaluation, and master summaries do not preempt the online worker. When a rollout stops, raw capture closes first, then already queued VSA windows drain before the rollout enters post-review. VSA only writes understanding artifacts: phase, risk, uncertainty, and evidence references.

Lifecycle behavior. The VLM supplies visual perception. The agent maps that perception into phase state, risk flags, and review-worthy uncertainty. This decomposition makes VSA useful for downstream review while keeping final dataset decisions in the offline governance path.

Failure handling. If the VLM times out, emits invalid JSON, or omits required fields, raw capture continues. The agent writes a warning snapshot or review marker for the missing semantics. Online perception remains helpful, and raw capture remains protected.

Details. Appendix D gives the event-windowing, contract-validation, phase-stabilization, and final-observation details.

B.4 Post-Rollout Review Agent

Role. The Post-Rollout Review Agent starts from an evidence index built over the raw rollout. It combines online snapshots, terminal evidence, operator context, and raw keyframes into a final annotation.

Packetization. The agent packages terminal evidence, post-terminal context, temporal context, operator notes, and VSA snapshots into separate review packets. Packetization lets the system weight different evidence sources and isolate failures.

Evidence index. The evidence index records raw video intervals, keyframes, VSA snapshots, operator notes, stop reasons, synchronization warnings, missing streams, and robot-log snippets. Packets are constructed from this index, giving each downstream decision concrete evidence references.

Annotation schema. The final annotation contains a human-facing explanation and machine-facing fields for dataset admission, dataset health, and future collection planning. It records outcome, confidence, phase timeline, failure type, recovery status, risk flags, key evidence references, packet-level warnings, and an L1 label.

Fallback behavior. The review agent sits downstream of both raw evidence and VSA snapshots. Snapshots provide sparse semantic anchors. When an online VLM call is unavailable or uncertain, the final annotation can still rely on raw keyframes, operator notes, and terminal evidence.

Aggregation behavior. The aggregator treats final-state evidence as decisive for task success unless post-terminal evidence contradicts it. Temporal packets explain how the rollout reached the final state and where retries or failures occurred. Missing or failed packets lower confidence and may route the rollout to human review, but they do not erase the evidence produced by other packets.

VLM authority boundary. Offline VLM calls are allowed to provide terminal evidence, phase correction, retry or failure evidence, label-quality hints, and training-usability hints. Dataset admission is written later by governance rules. If a model response contains an old or free-form dataset decision field, the post-review pipeline treats it as advisory evidence and lets the Data Governance Agent write the formal admission artifact.

Details. Appendix E details the evidence index, packet types, aggregation rules, failure analysis, and admission semantics.

B.5 Data Governance Agent

Role. The Data Governance Agent applies structured admission rules to the review output. It is the lifecycle boundary between semantic interpretation and dataset membership.

Inputs. The agent reads the final annotation, task admission policy, failure analysis, operator notes, risk flags, confidence fields, and existing dataset state.

Admission classes. Typical admission classes are:

- `clean_success`: clean success suitable for primary imitation training.
- `accepted_with_labels`: final success with useful recovered failure, retry, or phase labels.
- `successful_but_ambiguous`: trainable evidence that still deserves human inspection because labels or visual confidence are uncertain.
- `failure_pool_candidate`: useful failure evidence for diagnosis, recovery training, or targeted recollection, but excluded from the primary imitation set by default.
- `empty_or_insufficient`: unsafe, corrupted, off-task, or insufficiently evidenced rollout.

Separated fields. Semantic review and admission are kept as separate artifacts. The agent writes separate fields for training eligibility, review priority, failure-pool routing, exclusion reason, and operator override history. This separation lets a rollout be useful for training while still requiring human inspection, or useful for diagnosis while excluded from imitation training.

Decision versus training eligibility. The human-facing `decision` bucket can be `accepted`, `needs_review`, `retry_recommended`, or `rejected`. Training selection uses the separate `accepted_for_training` field, so a rollout can remain review-worthy while still being eligible for training, and a human-approved failure can remain outside the primary imitation set.

Rule order. The implementation treats safety and data integrity checks as hard exclusions, then applies task-specific admission rules, then assigns review priority and failure-pool routing. This order prevents a high-confidence visual success from overriding corrupted streams, unsafe behavior, or missing terminal evidence.

Override semantics. Operator overrides are stored as lineage fields alongside the automated decision. The artifact records who changed the decision, which field changed, what evidence was cited, and whether the override affects training eligibility, documentation review, or failure-pool membership.

Downstream use. The separated fields give the Data Health Agent enough structure to distinguish missing coverage from poor execution, unsafe failure, ambiguous evidence, and label uncertainty.

Lifecycle effect. The output is a dataset decision artifact. It links the review to accepted rollout manifests, rejected or quarantined examples, failure pools, and later dataset locks. Together, these links make data admission governable and traceable across later training runs.

B.6 Data Health Agent

Role. The Data Health Agent turns rollout-level admission decisions into dataset-level diagnosis. It assesses whether the current dataset is balanced, well evidenced, and useful for the next training run.

Inputs. The agent reads accepted and rejected rollout artifacts, final annotations, phase timelines, failure labels, confidence fields, missing-stream warnings, operator interventions, and dataset-lock candidates.

Diagnostics. The health artifact reports phase coverage and the distribution of successes and failures. It summarizes confidence and missing modalities, then checks sample quality for duplication or corruption. Intervention rates and imbalance help explain how representative the collection is. Useful failures remain visible in the failure pool. These diagnostics distinguish a shortage of successful demonstrations from a gap in a particular phase or weak sensor evidence in an otherwise large dataset.

Computation. Health summaries use review and admission artifacts, including episodes outside the primary training set. A small collection may lack an entire task phase. A larger one may still be skewed or depend heavily on operator interventions. The same records expose weak evidence and low-quality streams that simple sample counts would miss.

Warnings. Repeated failures from one scene or an excess of easy successes can flag a coverage problem. Missing terminal evidence or low confidence in a phase can make its labels unreliable. Frequent operator intervention raises a separate warning about the collection process. The agent also flags modality mismatch with the target framework and possible train/eval leakage.

Lifecycle output. The Data Health Agent writes a training-readiness summary and a next-collection recommendation. The Training Lifecycle runner reads this output before adapter conversion, and the Master Agent uses it when proposing what the operator should collect next. Coverage counts, phase histograms, failure distributions, and distribution gaps are computed deterministically; the Data Health Agent and Master Agent turn those evidence fields into structured next-collection briefs, which are still written as artifacts.

Why it is separate from admission. Admission decides whether individual rollouts may enter a dataset or failure pool. The Data Health Agent decides whether the resulting collection is useful as a training distribution. Keeping these agents separate prevents clean individual examples from being mistaken for a complete dataset.

B.7 Framework Discovery Agent and Dataset Adapter Agent

Framework Discovery Agent. The Framework Discovery Agent reads a training repository as a lab would use it: through configuration files, dataset loaders, command-line entry points, expected data layouts, and the commands already trusted by the team. The user's dataset, training, and evaluation commands are treated as authoritative hints, so the host repository can keep its own API.

Discovery output. The agent writes a target data contract, adapter-registry summary, candidate command profile, output expectations, discovery event log, and integration manifest. Optional code inspection helps infer loaders, checkpoint paths, TensorBoard or log sources, and evaluation-result files.

Conservative inference. When multiple loaders, commands, or checkpoint locations are plausible, the agent records alternatives and asks for confirmation through the integration manifest. Explicit user commands outrank speculative repository interpretation; the system adapts to the lab's working practice while preserving its training conventions.

Dataset Adapter Agent. The Dataset Adapter Agent materializes data for the target framework by creating manifests, symlinks, metadata views, or converted files. Raw robot data remains read-only, and every converted output points back to accepted rollout identifiers, admission artifacts, and hashes.

Adapter report. The adapter report records input rollouts, source hashes, output paths, conversion strategy, dropped or remapped fields, camera/action naming, modality availability, and warnings. A later policy can therefore be traced to the governed dataset and to the exact data view presented to the learner.

Lineage behavior. The discovery result contains candidate train and evaluation commands, dataset root assumptions, checkpoint output patterns, metric-log sources, adapter confidence, and warnings for missing or ambiguous conventions. The adapter records whether it used a direct manifest, a user-provided conversion command, or a generated framework-specific view. The adapted data can be regenerated from the dataset lock and adapter profile, keeping its relation to review evidence explicit.

Framework coverage. The same contract pattern supports ACT-style behavior cloning, diffusion-policy training, VLA fine-tuning, and LeRobot-compatible workflows. The governance layer records what the external learner expects while leaving the learner intact.

B.8 Training Monitor Agent and Version Governance Agent

Training lifecycle. The Training Lifecycle runner builds a train manifest from post-review admission artifacts, invokes the Data Health Agent, creates a dataset lock, adapts the selected rollouts, launches the framework command, and records the resulting training status.

Dataset lock. The dataset lock freezes accepted rollout identifiers, task and robot profiles, adapter profile, artifact hashes, and the command context used to create the training view. It is written before training begins, giving each policy candidate a fixed dataset state.

Monitor output. The Training Monitor Agent records process state, command line, working directory, environment hints, stdout and stderr summaries, parsed scalar metrics, checkpoint candidates, evaluation outputs, warnings, and terminal error categories using the profile discovered for that repository.

Failure handling. If training fails, the monitor still writes a status artifact with command, environment hints, logs, error category, and partial outputs. Failed runs remain part of lineage because they often explain why a candidate policy was not produced or why a dataset needed adaptation.

Version governance. The Version Governance Agent creates immutable dataset locks and policy metadata. The selected checkpoint is bound to its parent policy, dataset lock, framework profile, adapter version, command hash, code revision, and observed metrics.

Policy metadata. Policy metadata records checkpoint hash, parent policy, compatibility constraints, rollback target, training run identifier, dataset lock, framework profile, metrics, and evaluation links. The reference implementation marks policy metadata immutable after creation, protecting the identity of a policy version once evaluation begins.

Why it matters. The lock-before-train discipline makes external learners auditable. If a policy later behaves unexpectedly, the team can reconstruct the rollouts, reviews, adapter, command, metrics, and checkpoint that produced it, even when the learner itself is an unmodified repository.

B.9 Policy Evaluation Agent and Deployment Governance Agent

Evaluation. The Policy Evaluation Agent uses the same semantic evidence pipeline as post-rollout review, with evaluation-specific criteria and parent-policy comparison. It records per-trial success, failure phase, safety issue, regression against the parent policy, evidence links, and weak task phases.

Evaluation context. Evaluation trials are linked to the candidate policy, parent policy, dataset lock, task contract, and robot profile. This context separates broad policy regression from failures that occur in task phases already known to be under-covered in the training data.

Deployment governance. The Deployment Governance Agent reads evaluation artifacts together with policy metadata, dataset health, coverage summaries, rollback candidates, policy hash, dataset lock hash, and approval ticket. It then writes the lifecycle route: deploy, hold, rollback, or collect more.

Executor boundary. The deployment artifact is a governance recommendation and audit record. The reference implementation records release decisions while leaving policy-server switching, rollback commands, and safety interlocks to the lab's approved executor. Deployment and rollback therefore remain operator-confirmed actions with their own approval trail.

Decision fields. The deployment ticket records recommendation, confidence, supporting evidence, blocking risks, rollback target, required approvals, coverage caveats, and next-collection hints. Deploy, hold, rollback, and collect-more decisions all share the same evidence-linked ticket format.

Ticket semantics. The deployment ticket contains a recommendation plus reasons and lineage evidence. A team can distinguish "hold because coverage is weak" from "hold because the candidate regresses on a phase the parent already solved."

B.10 Master Agent and Lifecycle Summary Artifacts

Inputs. The Master Agent provides iteration-level memory. Setup and task artifacts establish the current context. Dataset decisions and health reports describe data readiness, while framework discovery and training status describe learner progress. Evaluation and deployment artifacts show what

happened to the policy candidate. The agent reads these summaries alongside its previous memory and emits a lifecycle summary at manual or automatic checkpoints. Artifact hashes link each input to its source record and dataset lock. Stage names and task phases locate the current work. Routing counts and unresolved review queues expose data blockers; evaluation regressions and deployment-ticket status expose policy blockers. Previous memory supplies the history behind these issues. Free-form notes remain evidence strings attached to typed fields.

Low-token contract. Every major stage exposes a compact summary for the Master Agent. The agent uses these summaries by default, which keeps cross-iteration synthesis cheap, responsive, and focused on the next collection decision.

Health boundary. The Master Agent summarizes lifecycle state, risks, and next-collection options. The Health view remains the operational diagnostic surface for data sources, ROS topic freshness, recorder status, VSA workers, post-review workers, and training services. The Master Agent consumes health summaries and leaves low-level service diagnosis to the Health view.

Memory update. The memory artifact records persistent lessons across iterations: recurring failure modes, stable task assumptions, known weak phases, effective collection strategies, and framework-specific integration notes. These summaries are kept compact, allowing future iterations to reuse the lesson without rereading every raw video or log.

Outputs. It writes four artifacts: `master.state.json`, `master.review.json`, `master.memory.jsonl`, and a typed `next_collection_brief`. The brief contains a route (`collect`, `review`, `hold`, `train`, `rollback`), target phase, collection focus, evidence references, confidence, and blocking risks.

Synthesis procedure. The Master Agent follows the same procedure at each checkpoint. It first collects candidate issues from the Data Health Agent, review packets, evaluation summaries, deployment tickets, framework status, and previous memory. It then deduplicates issues by task phase and evidence source, ranks them by current task risks and lifecycle blockers, and asks the language model to explain only the top conflicts that cannot be resolved by fixed rules alone. The final JSON must cite the source artifacts that support the recommendation. If a required field is missing, if the top candidates conflict without enough evidence, or if the confidence falls below the task threshold, the action route becomes `review` or `hold`.

Next-collection planning. Suggested actions are intentionally grounded in existing lineage artifacts. The agent may recommend collecting more data for a weak phase, inspecting ambiguous reviews, rerunning framework discovery, training from a refreshed dataset lock, holding deployment, or rolling back to a parent policy. The summary view records the recommendation and evidence for the operator.

Advisory role. The Master Agent keeps humans oriented and proposes targeted collection or inspection. It answers iteration-level questions: what changed since the last policy, which failure mode dominates, whether the current blocker is data, training integration, evaluation coverage, or deployment risk, and what a data-collection operator should collect or inspect next.

B.11 Design Rationale Across Modules

Design principles in detail. The five principles in Section 3.2 serve as operational constraints for the system. *Traceability* lets a team recover why a checkpoint exists by following its evidence back to the robot and task. *Semantic coverage* retains the observations needed to explain a slip or an unstable terminal state. It also preserves recovered failures that a scalar success metric would hide. *Agent autonomy with explicit state* lets agents make semantic and integration judgments through inspectable artifacts. *Portability* places local interface assumptions in profiles and contracts, with

adapters providing the learner's data view. *Operator efficiency* reduces routine video review and preparation while routing ambiguous evidence to humans.

Why agent governance? Agent governance records context-dependent judgments as executable and inspectable lifecycle operations. Fixed rules are appropriate for thresholds, hashes, locks, and schema checks. Semantic questions need richer context: whether a failure is trainable, whether a repository expects a particular data contract, or whether a policy regression reflects a dataset gap. RoboLineage places agents at these semantic transitions, then records their outputs as typed artifacts with provenance, uncertainty, and bounded downstream effects.

Why lifecycle and lineage? Robot policy improvement unfolds as a sequence of dependent transformations. A rollout is reviewed; review produces admission; admission changes a dataset version; the dataset version produces a training run; the training run produces policy metadata; evaluation and deployment decisions use that metadata. Lineage records these relations in one artifact chain. The resulting record turns the question "why does this policy exist?" into a concrete trace.

Why Robot Onboarding Agent? Multi-robot adaptation requires an explicit account of what each robot observes and controls, which topics or files correspond to camera, state, action, gripper, contact, and health, and which warnings should follow from missing streams. The Robot Onboarding Agent localizes these differences in a profile, giving downstream review, training, and evaluation modules the same contract across robot types.

Why Task Config Agent? The same physical trace can mean different things under different tasks. A contact event can be success for a drawer task, risk for a placement task, and irrelevant for a navigation task. The Task Config Agent makes success criteria, phases, risk events, snapshot priorities, and admission rules explicit. Agents then interpret the video through the task context.

Why packetized VSA/post-review? A single VLM answer over a video rarely carries enough structure for policy iteration. A policy team needs to locate the failed segment and determine whether the terminal state was stable. That evidence must support training eligibility and the dataset route, then remain linked to the later checkpoint and next collection brief. Packetized review attaches frame references and confidence to these judgments, so later stages can inspect and reuse them.

Why Dataset Admission and Data Governance Agent? Success, trainability, review priority, and high-value failure are different concepts. A successful but ambiguous rollout may be useful for training while still requiring documentation review; a failed rollout may be excluded from imitation learning but valuable for failure taxonomy and targeted recollection. The Data Governance Agent records training eligibility, review priority, and failure-pool membership in separate fields.

Why Data Health Agent? Correct individual rollouts do not imply a ready dataset. A collection of clean episodes can still over-represent easy scenes or miss a critical phase. The Data Health Agent checks these distribution gaps alongside evidence quality and intervention patterns. Its diagnosis helps the operator choose what to collect next.

Why Framework Discovery Agent? Real robot-learning labs already have training repositories. Requiring every team to rewrite its training stack around a new system would defeat lightweight adoption. The Framework Discovery Agent reads the existing repository, commands, dataset loaders, configuration files, and output conventions to infer a target data contract and integration manifest. This lets the governance layer adapt to ACT, diffusion-policy, VLA fine-tuning, or other learners without making those learners native to RoboLineage.

Why Dataset Adapter Agent? Conversion is where lineage is often lost. A rollout may become an HDF5 file, manifest row, symlink tree, or framework-specific data view, and the connection to

review evidence can disappear. The Dataset Adapter Agent materializes the target format while preserving rollout identifiers, admission artifacts, hashes, and dataset-lock ancestry.

Why Training Lifecycle, Training Monitor Agent, and Version Governance Agent? Training is often the least inspectable step: a command runs, a checkpoint appears, and the relation to data decisions is reconstructed later by memory. The training lifecycle makes this external process part of lineage. The monitor preserves the run's logs and results, including failures that explain why no usable checkpoint appeared. Version artifacts identify the frozen data and adapter used by the learner. They also retain the training command and parent policy, so the new checkpoint has a traceable identity.

Why Policy Evaluation Agent? Success rate alone hides the structure of policy quality. A candidate may have the same average success rate as its parent while introducing a safety regression, failing in a new phase, or succeeding only on scenes already covered by the dataset. The Policy Evaluation Agent records trial evidence, weak phases, failure types, regressions, and links back to the training lineage.

Why Deployment Governance Agent? Deployment decisions depend on the failure profile as well as success rate. A high success rate may be unacceptable if failures are unsafe or concentrated in critical scenes; a lower success rate may be useful if it fixes an important failure mode and requests specific recollection. The Deployment Governance Agent turns evaluation evidence, coverage, regression, rollback target, and policy provenance into an inspectable deploy/hold/rollback/collect-more recommendation.

Why the Master Agent? Local artifacts explain local decisions, but policy iteration also needs a cross-iteration memory. The Master Agent reads compact lineage artifacts and summarizes the current stage, blocking risks, dominant failure modes, and next collection target. Its lifecycle summary makes the accumulated effect of review, training, evaluation, and deployment legible to the operator.

Why a frontend? A command-line lineage tracker may be auditable but still miss the rhythm of collection sessions. The frontend lets operators control rollout capture and inspect the evidence prepared by agents. Once uncertain cases are resolved and dataset decisions approved, the operator can launch training and follow the policy through deployment review. Agents maintain the records behind these actions, while humans operate the robot and confirm consequential decisions.

Why lightweight artifacts? The system is intended to be adopted by robot labs without becoming a monolithic robotics platform. JSON, YAML, JSONL, hashes, and schema validation are easy to inspect, version, copy with datasets, and consume from existing scripts. This keeps RoboLineage as a portable governance layer around existing robot-learning stacks.

Design summary. Robot policy iteration needs semantic judgment and durable state in the same loop. RoboLineage uses agents to produce task-grounded interpretations and lineage artifacts to carry auditable lifecycle state across iterations.

C Artifact Schemas and Lineage Graph

C.1 Lineage Graph Semantics

The lineage graph is a typed, versioned provenance record over lifecycle artifacts. It is stored through ordinary files and hashes; the implementation does not require a specialized graph database. The record makes policy iteration auditable by showing which artifacts were read, which evidence supported a decision, and which lifecycle transition produced a dataset, policy, evaluation, or deployment ticket.

The graph is append-oriented. Each review, dataset decision, dataset lock, training run, and deployment ticket is added as a node with provenance edges to its dependencies. Corrections, operator overrides, and reruns appear as new artifacts that supersede or annotate earlier artifacts. Rollback and audit stay possible with lightweight storage.

C.2 Node Types

The core node types are:

- `robot.profile`: supplied and validated robot interface, stream bindings, calibration hints, safety limits, and warnings;
- `task.config`: phases, success criteria, risks, snapshot priorities, and admission rules;
- `collection.context`: operator, teleoperation interface, scripted routine, policy-assisted protocol, and intervention policy used for collection;
- `policy.metadata`: parent policy, checkpoint hash, dataset lock, framework profile, metrics, rollback target, and compatibility constraints;
- `rollout.record`: raw data paths, hashes, operator events, stop reason, synchronization status, collection context, and optional parent policy;
- `snapshot.vsa`: event-windowed semantic observation linked to selected frames and rollout events;
- `annotation.final`: post-rollout outcome, phase timeline, failure analysis, confidence, risk flags, and L1 label;
- `dataset.decision`: training eligibility, review priority, failure-pool routing, exclusion reason, and override history;
- `dataset.manifest` and `dataset.lock`: accepted rollout set, artifact hashes, adapter context, and frozen training view;
- `framework.profile`: discovered training repository contract, commands, expected data layout, metrics source, and outputs;
- `training.status`: process state, command, logs, metrics, warnings, checkpoints, and terminal errors;
- `eval.summary`: per-trial outcomes, regressions, weak phases, safety issues, and evidence links;
- `deployment.ticket`: deploy, hold, rollback, or collect-more recommendation with supporting evidence and approvals;
- `master.review`: current lifecycle state, persistent memory, blocking risks, and next-collection recommendation.

C.3 Edge Vocabulary

The graph uses typed provenance edges. A rollout `uses` a robot profile, task config, and collection context; it may also `references` a parent policy when collection is policy-assisted. A snapshot `observes` a rollout window and `references` selected frames. A final annotation `reviews` a rollout and `references` snapshots, packets, and keyframes. A dataset decision `admits`, `rejects`, `quarantines`, or `routes_to_failure_pool` an annotation. A dataset manifest `contains` accepted rollout identifiers. A dataset lock `freezes` a manifest, hashes, task config, robot profile, and adapter context. A training status `runs_on` a dataset lock and `uses` a framework profile. A policy metadata artifact `was_trained_from` a training run and `descends_from` a parent policy. An evaluation summary `evaluates` a policy. A deployment ticket `decides` over an evaluation summary and policy metadata. A master summary `summarizes` compact lineage state and `recommends` a next lifecycle action.

C.4 Cross-Iteration Example

In a typical iteration, a collection context, robot profile, and task config produce rollout nodes. Online VSA snapshots `observe` event-conditioned windows inside each rollout. The Post-Rollout Review Agent `reviews` the rollout using raw keyframes and snapshots, producing an annotation. The Data Governance Agent `admits` or `rejects` that annotation and may route it to a failure pool. The Data Health Agent `summarizes` the accepted and rejected decisions. A dataset lock `freezes` the accepted set before training. A training run `uses` the lock and framework profile, and policy metadata `was_trained_from` the resulting checkpoint. Evaluation `evaluates` the candidate policy, the Deployment Governance Agent `decides` whether to deploy, hold, rollback, or collect more, and the Master Agent `recommends` the next collection target. The next iteration inherits policy metadata and master memory from the same lineage chain.

C.5 Artifact Identity and Versioning

Each artifact has a stable identifier, type, creation time, producer, schema version, content hash, and dependency list. The dependency list is the producing agent's local view of the lineage graph. When an artifact is superseded, the new artifact records the superseded identifier and the reason, such as operator override, updated task config, rerun review, new adapter, or failed training retry. Earlier decisions can be corrected without erasing the audit trail.

C.6 Minimal Rollout Schema

A rollout record contains:

- rollout identifier, session identifier, robot identifier, task identifier, collection context identifier, and optional parent policy identifier;
- raw data paths and hashes;
- time interval, collection context, operator events, and stop reason;
- stream availability and synchronization status;
- links to snapshots, final annotation, and dataset decision.

In the reference implementation, raw robot data is the source of truth and is recorded through a raw manifest. For ROS-based systems this manifest points to the rosbag2 directory, recorded topics, recorder status, domain identifier, and raw-data hashes. Semantic artifacts such as VSA snapshots and post-review annotations reference the raw record.

C.7 Minimal Dataset Decision Schema

A dataset decision record contains:

- decision bucket, training eligibility, review priority, and label quality;
- failure-pool routing, exclusion reason, and retry recommendation;
- source annotation identifier, task admission policy, and confidence fields;
- operator override history and links to evidence used in the override;
- downstream links to dataset manifest, dataset health, and dataset lock.

C.8 Minimal Dataset Lock Schema

A dataset lock record contains:

- lock identifier, dataset manifest identifier, and selected rollout identifiers;
- source artifact hashes for rollout records, annotations, dataset decisions, robot profile, and task config;

- adapter profile, target training contract, and conversion report;
- creation command, producer, schema version, and compatibility notes.

C.9 Minimal Policy Schema

A policy record contains:

- policy identifier, parent policy, training run, framework profile, and checkpoint hash;
- dataset lock hash and accepted rollout count;
- training metrics and configuration hash;
- evaluation summaries and deployment tickets;
- rollback target and compatibility constraints.

These schemas are intentionally small. They support lightweight adoption first, with richer schemas layered on top as labs need them.

D Online Visual Snapshot Agent Details

D.1 Event-Driven Windowing

Uniform frame sampling spends review capacity on uninformative frames. VSA schedules windows around robot, policy, and operator events. The implementation uses events including `sequence_start`, `gripper_close`, `gripper_open`, `gripper_burst`, `contact_transition`, `still_start`, `motion_resume`, `periodic_sample`, `heartbeat`, and `final_observation`. A gripper burst suggests grasp or release; a contact transition suggests a potential success, failure, or risk point; a phase marker suggests a task boundary; and a rollout stop suggests either terminal success or operator intervention. Robot evidence is sparse and structured, and decisive information usually appears near an action boundary, contact change, stop event, or final-state check.

The scheduler coalesces nearby events into one window when they describe the same moment, but preserves protected events such as sequence start and final observation. Periodic and heartbeat windows provide coverage when no high-information event occurs, while high-information events take priority when the online stream is dense. Each scheduled window records its anchor frame, selected keyframes, covered events, and reason codes, giving later review the reason the agent looked at that moment.

D.2 Prompting and Contract Validation

For each scheduled window, the VSA constructs a task-aware prompt from the robot profile, task config, recent events, selected keyframes, phase hints, and failure signals. The VLM is asked for phase, progress, visible evidence, risk level, imminent-failure hints, confidence, and a review flag. The response is validated against the snapshot contract before it becomes lineage evidence. Invalid JSON, timeout, or missing fields do not stop raw capture. They produce a conservative warning snapshot or a `needs_review` marker. A fluent model answer becomes governance evidence only after it satisfies the fields that downstream review, admission, and training agents know how to consume.

D.3 Raw Capture Decoupling

Raw capture is independent of semantic assessment. The raw log or rosbag is the source of truth for training adapters and audit. VSA snapshots are sparse interpretations attached to raw timestamps and frame identifiers. This decoupling is important operationally: if the model is slow, unavailable, or uncertain, the rollout is still captured, and the lifecycle records that the semantic evidence is incomplete. Semantic automation improves the lifecycle while raw evidence remains protected from model latency or model failure.

D.4 Phase Stabilization

The phase state machine keeps online semantics stable. Let p_t be the raw VLM phase prediction at window t and $\hat{p}_t$ be the stabilized phase. By default, $\hat{p}_t$ cannot move backward relative to $\hat{p}_{t-1}$. Sparse online evidence still needs to support real progress, so forward jumps are allowed when the window has transition evidence, high visual confidence, repeated phase observations, boundary events, or terminal evidence. Terminal skips are treated more carefully: a jump to the terminal phase requires stronger evidence unless risk is already high.

Rollback or retry requires explicit counter-evidence, such as a high-risk window, imminent-failure flag, progress regression, gripper burst, contact-transition event, or repeated visual evidence that the object or robot returned to an earlier phase. This distinction is critical after terminal actions: a real failure after release should be captured, while a single ambiguous frame should not rewrite a successful lifecycle.

The temporal stabilizer complements the phase state machine. It smooths phase and progress over adjacent windows, but treats risk asymmetrically: risk can increase immediately, while decreasing risk requires consecutive confirming windows. Boundary events such as gripper open, gripper close, motion resume, or still start reset relevant history so old votes do not block real phase changes. When the model is low-confidence or internally inconsistent, the stabilizer marks the window as `needs_review`. The stabilizer gives downstream dataset decisions lifecycle-consistent phase evidence despite jitter in online VLM predictions.

D.5 Final Observation Handling

Many robot tasks fail after the final action command. An object may slide after release, a drawer may rebound, or a placed item may remain unstable. RoboLineage records final observation as a dedicated snapshot. If a release event occurs, the VSA waits for a settle interval. If the rollout ends without a clean release, the rollout-stop event schedules a final window. Post-rollout review then weights final evidence heavily. In the implementation these are two separate paths: a release-settle path for nominal terminal actions and a rollout-stop path for interrupted or ambiguous episodes. Both paths produce lineage evidence for the final task state, which prevents a common failure mode where the policy stops moving before the most important success evidence is recorded.

E Post-Rollout Review Agent Details

E.1 Evidence Index

The Post-Rollout Review Agent starts from an evidence bundle built over the raw video. The bundle contains the task config, online VSA snapshots, window records, selected keyframes, raw-frame references, operator notes, logs, and final-observation coverage. The evidence index records frame spans, online phase sequence, risk and progress summaries, visual disagreements, and source paths used by review. Every final label can be traced back to the windows, frames, and events that supported it.

E.2 Evidence Packet Types

The review pipeline uses four packet types:

- **Terminal focus:** final observation, terminal keyframes, task success criteria, and relevant VSA terminal claims.
- **Post-terminal context:** frames after terminal action or stop, used to catch unstable final states.
- **Temporal context:** selected windows around phase changes, risks, retries, and interventions.
- **Operator and log context:** stop reason, operator note, robot errors, missing streams, and synchronization warnings.

Final success, post-terminal stability, failure localization, retries, operator context, and training usefulness depend on different evidence. Packetization also provides failure isolation: if one packet fails, the remaining packets can still support a lower-confidence review, and the missing evidence is recorded explicitly.

E.3 Aggregation Rules

The aggregator treats terminal focus as decisive unless contradicted by post-terminal evidence. Temporal context can identify failure phase and recovery, but it cannot by itself declare task success if final state evidence is missing. Missing packets lower confidence and may route to human review. The aggregator also records reason codes so dataset decisions remain inspectable. These rules keep success decisions grounded in terminal evidence while preserving the history needed to diagnose failures.

Packet failures are isolated. If one VLM call times out or returns invalid output, the remaining packets still contribute evidence. The failed packet is recorded with its purpose, image frames, error type, and error message; the rollout is marked for review when the missing packet could affect the decision. Admission remains pending when a failed packet leaves essential evidence missing.

E.4 Failure Analysis and Dataset Admission

Failure Analysis reads the final phase timeline, online risk evidence, retry events, and packet review outputs. It produces candidate failure segments, failure type, severity, recovery status, high-risk frames, and final-failure labels. The Data Governance Agent then separates rollout usefulness from documentation uncertainty. Clean successes enter the primary training set. Successful but ambiguous rollouts can be `accepted_for_training` while still routed to review. Recovered failures are preserved for recovery training and failure taxonomy. Empty, unsafe, corrupted, or insufficient episodes are excluded from training. This separation is central to the system: `needs_review` is a human-attention signal, not the same thing as rejection.

E.5 L1 Annotation

The L1 annotation is a lightweight label embedded in the final annotation. It contains outcome, task phase coverage, primary failure label, keyframes, and training usefulness. Keeping L1 labels in the same final annotation avoids a second manual labeling pass and ensures that dataset health, training selection, and the review UI reference the same source record.

F Representative VSA and Post-Review Evidence Plates

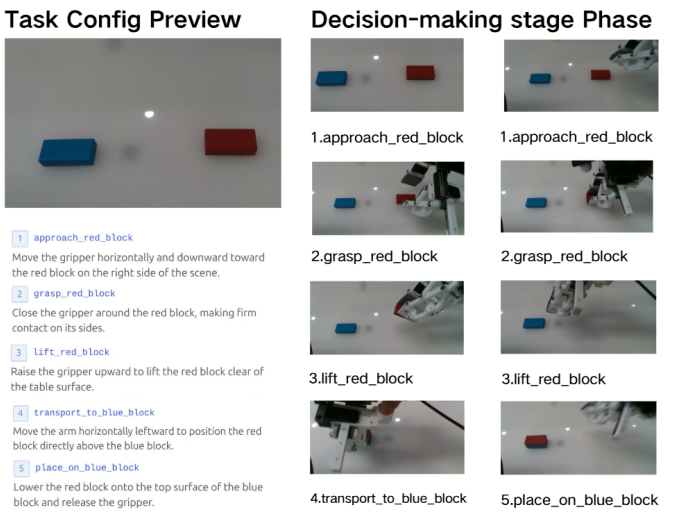
Figures 6–8 show six representative VSA/post-review plates from the benchmark. Each plate combines the task configuration preview, online VSA phase anchors, selected rollout frames, and the post-review decision. Together they span success routing, review routing, deformable and thin-object manipulation, reorientation, insertion, and long-horizon multi-object manipulation.

G Training Integration Details

G.1 Discovery Signals

The Framework Discovery Agent starts from command examples and README fragments. It examines dataset loaders and configuration keys to infer the expected data view. File names and imports provide additional clues, and previous framework profiles supply reusable context. When multiple entry points remain plausible, the agent proposes candidates for confirmation.

User-provided commands are authoritative hints because they encode the lab's actual training practice. Repository structure and code snippets then explain what those commands expect: dataset



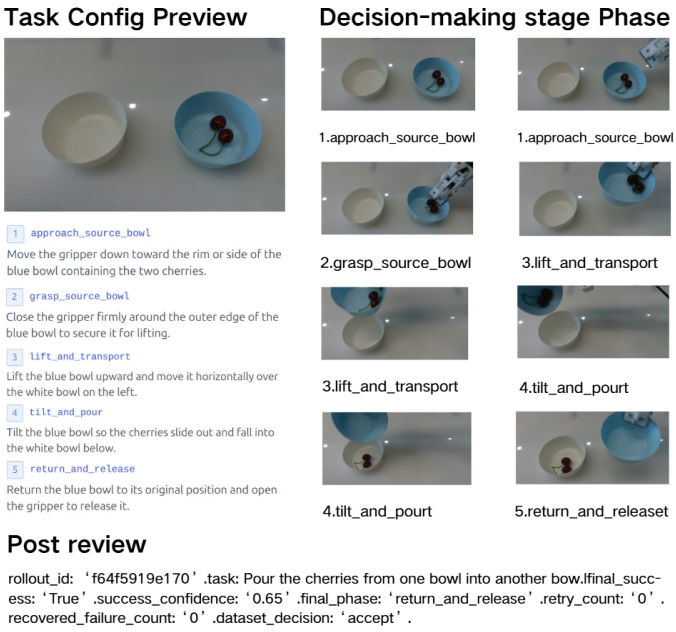
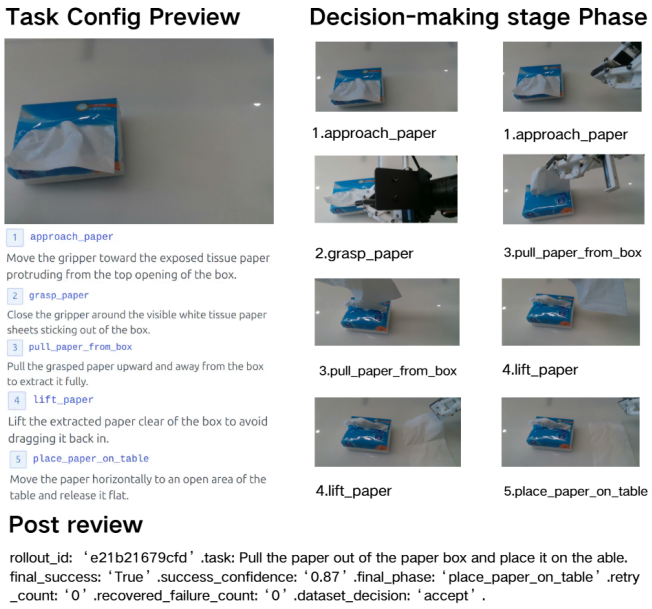


Figure 6: Representative VSA/post-review plates for standard pick-place and container-transfer tasks. The top plate shows an accepted block placement; the bottom plate shows a lower-confidence but accepted pouring rollout.



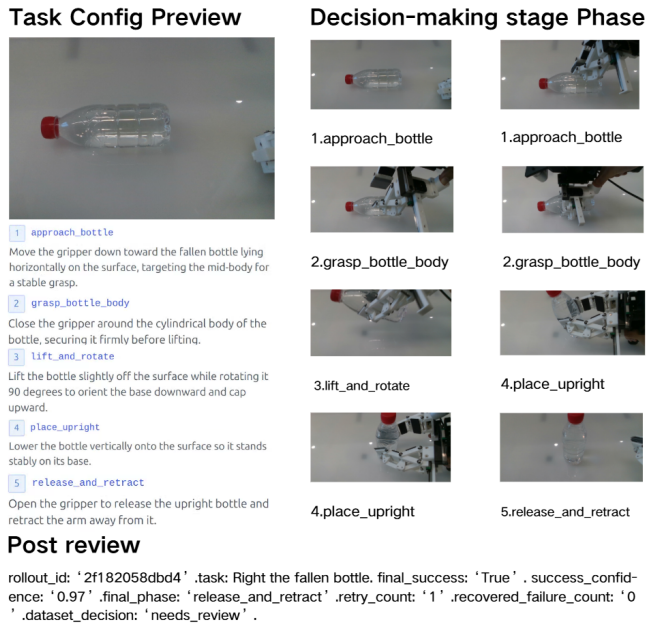
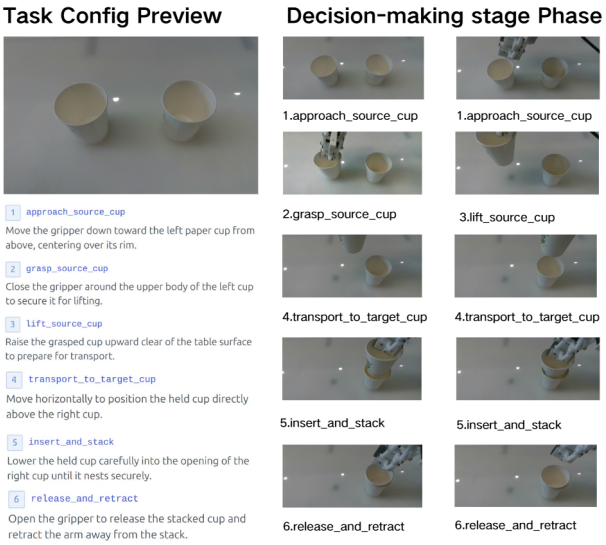


Figure 7: Representative VSA/post-review plates for thin-object extraction and object reorientation. These plates illustrate phase grounding beyond rigid pick-place and show how post-review records uncertainty or acceptance as a governed artifact.



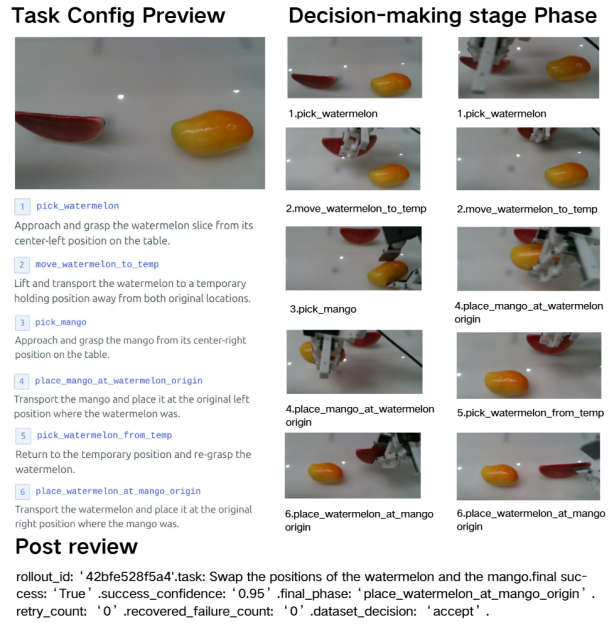


Figure 8: Representative VSA/post-review plates for precision insertion and long-horizon multi-object manipulation. The bottom plate preserves all six phases of the object-swap sequence, including intermediate object states.

loaders reveal file formats, configuration files reveal camera and action keys, and output conventions reveal where metrics and checkpoints will appear.

The discovery artifact records the scanned file summary, inferred framework type, user command intake, target data contract, selected adapter candidate, output locations, warnings, and an event log. The integration manifest then summarizes what later lifecycle stages need: training command, staging directories, checkpoint glob, train-log path, evaluation result path, metrics source, and dataset adapter strategy.

In our implementation, the same discovery interface covers ACT-style behavior cloning, diffusion-policy training, VLA fine-tuning, and LeRobot-compatible workflows. The discovered profile is descriptive: it records the repository's existing expectations and leaves the learner's API intact.

G.2 Target Data Contract

The target data contract bridges governed robot rollouts and an external learner. It records the expected dataset kind, input path template, camera names, action/state assumptions, and adapter confidence. This contract lets RoboLineage preserve one internal lineage graph while adapting to heterogeneous training frameworks. Ambiguous contracts carry warnings and assumptions for operator confirmation.

The contract is intentionally separate from the adapter implementation. The contract states what the learner expects; the adapter materializes accepted rollouts into that expectation. A team can replace an adapter or refine a converter while preserving the lineage meaning of the dataset lock.

G.3 Adapter Strategies

The adapter chooses among three strategies:

- **Direct manifest:** the training framework reads the RoboLineage manifest directly.
- **User command:** a lab-provided command consumes the manifest and performs conversion.
- **Generated converter:** the adapter creates a framework-specific data view while preserving links to source rollouts.

All three strategies produce a dataset lock before training begins.

They also produce an adaptation report with input rollout identifiers, output paths, file hashes, dropped or remapped fields, and warnings. This report documents how a reviewed rollout becomes a training sample for the external learner.

Raw robot data, review artifacts, and dataset locks remain authoritative. Converted files are reproducible projections for a particular training framework.

G.4 Training Lifecycle Outputs

Each training run has a workspace with its selected rollout manifest and dataset health report. Adaptation status and framework results remain alongside the training status. Policy metadata identifies the resulting checkpoint, and a deployment recommendation records what should happen to it next. The lifecycle context links the framework profile and dataset lock to the supporting post-review summary. It then follows the run through evaluation and deployment review. These links let an operator recover the data view and command that produced a checkpoint, together with the recorded metrics and review evidence.

The training status records where and when the command ran. Process state and stdout/stderr summaries show execution progress; parsed metrics and checkpoint candidates show training results. Warning codes and failure categories preserve unsuccessful runs. Policy metadata fixes the selected checkpoint hash and parent policy. It also identifies the dataset lock and framework profile, with the



Figure 9: Robot platforms used in the evaluation. From left to right: ARX tabletop arm, Realman humanoid platform, and GALBOT G1 humanoid platform.

adapter version and code revision needed to reconstruct the training view. Evaluation links connect that policy to later trials.

G.5 Policy Metadata Immutability

Policy metadata is written after training and made immutable at the file level in the reference implementation. The version check validates hashes whenever a policy is evaluated or considered for deployment.

H Frontend Workflow

The frontend turns lineage into a daily working interface. It exposes the lifecycle through operator views for Robot Onboarding Agent, Rollout Control, Review and Lifecycle, Training, Evaluation, Deployment Governance Agent, Master Agent, and Data Health Agent. The operator can:

- onboard a robot and inspect validated streams;
- define task phases, success criteria, risk events, and collection policy;
- run rollouts and watch online snapshots arrive;
- review final annotations with evidence packets and keyframes;
- approve, override, or comment on dataset decisions;
- inspect dataset health before training;
- launch framework adapters and monitor training;
- compare evaluation summaries across policies;
- inspect deployment tickets and rollback targets;
- read Master Agent summaries and next-collection plans.

Lifecycle governance helps only when it fits the rhythm of collection sessions. An auditable command-line tracker is useful for records; the frontend makes the same lifecycle actionable during repeated rollout and review cycles.

The frontend separates judgment from bookkeeping. Humans operate the robot, inspect uncertain evidence, and approve consequential decisions; agents prepare evidence, maintain artifact links, surface warnings, and propose the next lifecycle action. This division of work is the practical mechanism behind the operator-facing workflow evaluated in the main text.

I Experimental Protocol Details

I.1 Annotation Protocol

Expert annotators label rollouts using a common rubric. Each rollout receives: task outcome, primary failure phase, secondary failure labels, safety or risk flags, recovery status, and admission decision. Annotators see raw video and robot logs but not RoboLineage's automated decision during ground-truth labeling. Neither annotator authored the RoboLineage prompts, baseline definitions, or scoring scripts. Final labels are fixed before model evaluation. Two experts independently label the 500-rollout benchmark to estimate label consistency. Before consensus adjudication, they agree on 97.0% of outcomes, 91.0% of primary failure phases, and 98.0% of admission decisions; exact agreement on the combined outcome/phase/admission tuple is 90.0% with Cohen's $\kappa = 0.86$. The benchmark labels used for scoring are consensus labels produced under the rubric. Outcome accuracy scores final success/failure, failure-phase accuracy scores the primary failure phase, and admission accuracy scores exact agreement with expert consensus over normalized data-use routes across all 500 rollouts before operator overrides, under fixed admission rules. Bad-data leakage is a stricter false-accept metric over the 194 expert-rejected rollouts. Table 1 reports point estimates on this fixed 500-rollout benchmark.

Task contracts are frozen before benchmark scoring. For each task, the contract is written from the natural-language goal, robot profile, available streams, and two smoke rollouts used only to check phase names and success criteria. These smoke rollouts are excluded from all reported metrics. After freezing the contract, we do not edit prompts, phase definitions, admission rules, or thresholds based on benchmark labels. Six late-added task variants were scored after the prompt and schema set had already been frozen; they use the same contract-generation procedure and serve as a small check against tuning to the original task roster.

We stress-test task-contract brittleness with a 100-rollout replay. Paraphrased task goals had little effect in pilot replays. Underspecified or wrong success criteria reduced review quality and increased review routing. Contract conflicts are recorded for inspection before primary-training admission.

I.2 Baseline Definitions

All semantic-review baselines use the same frozen Claude Sonnet 4.6 model configuration and are scored against the same expert labels. The distinction is what evidence is available before the final review artifact is written. Table 7 summarizes the controlled differences.

Method	Evidence available
Direct video-to-VLM	Uniformly sampled frames or video summary; no VSA anchors
VLM + schema only	Same frames as direct VLM, with schema validation and malformed-output retries
VSA-only (no post-review)	Online VSA snapshots, final VSA memory, and task contract; no asynchronous packet review
Post-review only (no VSA)	Raw keyframes and final observation; no online snapshot artifacts
No packetized review	VSA snapshots and raw frames concatenated into a single review prompt
No final observation / terminal focus	VSA and packet evidence except final settle/release window
Full RoboLineage	VSA snapshots, packet evidence, final observation, and task contract

Table 7: Controlled baseline definitions for Table 1. Baselines share the same model backend, rollout pool, task contracts, and scoring rubric; they differ in which evidence the VSA/post-review pipeline can use.

The deterministic rule tree was written from the task-independent governance rubric, schema fields, task contracts, and the two smoke rollouts used to check each contract. It was finalized before benchmark scoring, did not use consensus labels from the 500-rollout benchmark, and was not revised after inspecting RoboLineage errors; authoring it took about 3.5 hours.

Diagnostic	Count	Rate	Meaning
Primary training manifest	296	59.2%	Rollouts admitted into the training view after review and Data Governance Agent
Routed to human review	74	14.8%	Ambiguous, low-confidence, or contract-conflicting cases held for inspection
Failure pool	76	15.2%	Useful failures or recoverable traces preserved outside the primary training manifest
Rejected / excluded	54	10.8%	Unsafe, wrong-task, corrupt, or irrecoverable rollouts excluded from training
Operator override	18	3.6%	Human changes after inspecting routed evidence; stored as lineage fields
False accept into primary manifest	4	2.1% of rejected	Expert-rejected rollouts that still entered primary training
False reject / unnecessary exclusion	14	4.6% of accepted	Expert-accepted rollouts excluded or routed away from primary training

Table 8: Routing and override diagnostics for the Full RoboLineage row in Table 1. The first four rows partition the 500-rollout benchmark before operator overrides; error rows use the same decisions. Overrides record later human changes within routed cases. False accepts use the 194 expert-rejected rollouts as denominator, and false rejects use the 306 expert-accepted rollouts as denominator.

Error type	Representative case	Lineage containment
False accept	A cube stack appeared stable in the terminal camera view, but the object slid outside the success tolerance after release	Post-terminal evidence was missing; the rollout was later linked to a terminal-settle rule update and marked as a reviewed exception
False reject	A drawer-opening rollout stopped early because the operator released the handle after reaching the threshold; the visual packet looked like an incomplete pull	Operator override restored the rollout to the accepted set while preserving the original rejection reason and evidence links
Contract mismatch	A long-horizon sorting rollout used a phase vocabulary from a pick-place-reset variant, making the intermediate success cue ambiguous	Data Governance Agent withheld admission and routed the rollout to review; the task contract was superseded by a new versioned artifact
Model perception error	VSA confused visually similar red/blue blocks during approach, assigning the wrong target object to an online anchor	Packetized post-review corrected the final annotation from terminal and temporal evidence before admission
Missing evidence	A Realman drawer rollout had usable motion logs but a dropped terminal frame	The rollout stayed outside the primary manifest and remained available as failure-pool evidence for diagnosis

Table 9: Representative failure and containment cases for the Full RoboLineage row in Table 1. The records preserve the error, cited evidence, routing decision, and subsequent correction.

I.3 Policy Quality Protocol

The human-comparison study uses eight tasks that are distinct from the two closed-loop improvement case studies. For each task, we record a fixed 100-rollout candidate pool while RoboLineage's online VSA runs during collection. The human expert workflow and the RoboLineage workflow then operate on the same raw candidate pool and independently select 60 accepted demonstrations for training; rejected and uncertain candidates remain outside the primary training set. Both workflows use the same robot, task distribution, training family, training command, and held-out 60-trial evaluation protocol. The human expert workflow uses manual rollout review, manual dataset admission, spreadsheet or DVC-like versioning, and hand-prepared training inputs. The RoboLineage workflow uses a data-collection operator working through agent-generated evidence and framework adapters. Figure 4 therefore evaluates dataset admission and workflow preparation from a shared candidate pool under matched training budgets.

I.4 Effort Study Protocol

For the effort study, operators process 12 matched 30-rollout pre-training cycles, four per robot family, under manual and RoboLineage-assisted conditions. Two trained data-collection operators run both workflows; condition order is alternated by robot family, and the first practice cycle for each workflow is discarded before timing. The manual workflow uses the lab's maintained scripts, folder templates, review sheets, and version notes. We count active human time before the training command begins: onboarding, rollout collection, post-review/data admission, and training frame-

work integration. Physical rollout collection is counted equally in both workflows. Manual sessions use the lab's normal video, spreadsheet, and data conversion workflow. Operators inspect complete rollout videos, mark phases and transitions, diagnose outcome and failure, record rationale, and assemble the training manifest. Manual onboarding inspects robot configuration and topics and edits recorder bindings. Manual framework preparation maps camera, state, action, and timing fields, materializes files, edits configuration, and checks training batches. RoboLineage sessions use the frontend, evidence packets, dataset decisions, and automated adapter preparation. Human inspection of uncertain rollouts, overrides, and admission approvals are counted in post-review time. We count visible waiting or inspection time when it blocks the active user, but not GPU training time, because both workflows call the same training stack once the dataset is ready. The reported 2-minute framework-integration cost is a per-iteration cost after a framework profile and adapter have been created; one-time adapter authoring, robot bring-up, and hardware debugging are not included in this routine-cycle timing. Online VSA overlaps raw collection, and asynchronous post-review overlaps later rollouts. The measured 9.4-minute review stage is the residual blocking time for unfinished or routed cases and manifest confirmation. Onboarding includes UI task/topic entry followed by agent-generated and validated reusable profile bindings. The validated profile is saved as YAML for reuse.

Item	Counted?	Scope
Rollout collection, review, admission, adapter preparation	Yes	Repeated every cycle
Robot profile / task contract validation	Yes, when used in-cycle	Reusable after validation
Hardware bring-up, drivers, workspace construction	No	Lab-specific setup
New adapter authoring or framework debugging	No	One-time integration
GPU training time	No	Same command after dataset prep

Table 10: Scope of the routine-cycle effort measurement.

Workflow	Onboarding	Collection	Post-review	Framework prep.	Total
Human expert workflow	10.1 ± 2.4	40.5 ± 4.8	59.2 ± 7.1	20.4 ± 3.0	130.2 ± 10.7
RoboLineage workflow	2.3 ± 0.8	40.6 ± 4.6	9.4 ± 2.1	2.0 ± 0.7	54.3 ± 5.9

Table 11: Active pre-training effort in minutes over 12 matched 30-rollout sessions. Values are mean $\pm$ standard deviation. Collection time is similar because the robot executes the same number of demonstrations; the savings come from review, admission bookkeeping, and framework preparation.

I.5 Storage Footprint

We account for storage over the same 500-rollout benchmark. Raw videos, robot streams, and recording metadata average 72.6 MiB per rollout. Structured lineage artifacts average 68 KiB per rollout, or 0.09% of raw storage. The VSA keyframe cache averages 16.3 MiB per rollout, or 22.5%; it is a regenerable evidence cache. Artifact generation is append-oriented, and summaries read indexed structured artifacts. These measurements quantify the evaluated workload; sustained latency and review backlog at larger scales remain future measurements.

I.6 Model Backend Sensitivity and Retry Protocol

All model-facing agents validate JSON outputs against their artifact schemas. Invalid or unparsable responses are retried up to two times with the same frozen prompt and model configuration. Online VSA timeouts, API errors, or parse failures write conservative low-confidence snapshot artifacts and mark the evidence for review; they do not block raw capture. Offline packet failures are isolated to the failed packet, while successful packets still aggregate into the final annotation. If missing evidence could affect admission, Data Governance routes the rollout to review or excludes it from the primary training manifest.

The main model route uses deterministic decoding where supported (temperature zero and fixed prompt templates), fixed image sampling rules, and logged request identifiers for replay. If a recurring failure cluster receives contradictory or low-confidence semantic labels across rollouts, the Data

Health Agent opens a calibration warning and the Master Agent can recommend more inspection or targeted recollection, but the cluster is not promoted into a clean dataset decision until review resolves the contradiction. The warning and its resolution remain linked to subsequent dataset decisions.

To separate the governance architecture from a single model backend, we replay a stratified 200-rollout subset with Gemini 3.1 Pro Preview and qwen3-vl-plus routes using the same task configs, packet prompts, schemas, and expert labels. These routes are not tuned for the benchmark. Table 12 shows that semantic accuracy depends on model quality, while packetized evidence, schema validation, and conservative admission still route uncertain cases away from the primary training manifest. For the full pipeline, weaker backends produce more review routes under the same governance rules, increasing the inspection workload alongside the drop in semantic accuracy.

For the local open-weight replay, we use Qwen3-VL-32B-Instruct through vLLM on two RTX 4090 GPUs modified to 48 GB each. The 200 rollout identifiers, expert labels, prompts, task contracts, schemas, thresholds, and scoring are unchanged; we do not retune the route. This local model is distinct from the hosted qwen3-vl-plus route. It yields 165/200 correct outcomes, 146/200 correct phases, 167/200 matching admission routes, 6/100 false accepts, and 53/200 review routes.

Route on 200-rollout replay	Outcome ↑	Phase ↑	Admission ↑	Leakage ↓	Review route
Full RoboLineage, Claude Sonnet 4.6	95.5	87.5	96.0	2.0	13.5
Full RoboLineage, Gemini 3.1 Pro Preview	94.5	86.5	95.0	3.0	14.5
Full RoboLineage, qwen3-vl-plus	85.5	77.0	86.5	5.0	25.0
Direct video-to-VLM, Gemini 3.1 Pro Preview	81.0	61.5	82.5	9.0	7.0
Direct video-to-VLM, qwen3-vl-plus	72.0	52.0	75.0	14.0	7.5
Full RoboLineage, local Qwen3-VL-32B	82.5	73.0	83.5	6.0	26.5

Table 12: Model-backend sensitivity replay on a stratified 200-rollout subset. All values are percentages. Leakage is measured over 100 expert-rejected rollouts in the subset. Gemini tracks the main backend closely. qwen3-vl-plus lowers semantic accuracy and increases review routing, but the governed VSA/post-review pipeline still keeps false admission below direct video review with the same backend. The local open-weight route uses Qwen3-VL-32B-Instruct.

I.7 Closed-Loop Recollection Protocol

For each closed-loop task, we run three independent runs of three recollection rounds. Before recollection, the initial policy π_0 is evaluated on 60 trials in each run, for 180 trials per task. The reported initial success rates are means over these three runs. Each round uses 30 accepted demonstrations for training and 30 diagnostic evaluation trials for failure analysis. The random baseline collects additional rollouts using the same task distribution as the initial dataset, so it measures the benefit of more data without failure-targeted recollection. The human-expert condition asks an experienced robot-learning practitioner to inspect the same diagnostic rollouts and propose the next collection focus. The RoboLineage-guided condition uses Master Agent summaries, Data Health Agent artifacts, and Deployment Governance Agent tickets to choose target phases, failure modes, and scene variations. No lineage memory uses the same current-round VSA snapshots, post-review packets, schemas, admission artifacts, and session-local coverage summaries as RoboLineage, but forgets earlier rounds and policy ancestry. In practice, this removes the persistent Data Health Agent state, parent-policy outcomes, and prior next-collection briefs that connect a new failure to earlier data gaps and policy changes. After the third round, all final policies are evaluated on an independent 60-trial test set not used for recollection planning.

J Additional Results and Case Studies

J.1 Task Roster and Qualitative Review Coverage

Table 13 lists the semantic-review benchmark at the task family level. The benchmark intentionally mixes clean successes, recoverable failures, wrong-object failures, phase mismatch, clutter, and long-horizon partial completions so that reliable review cannot be obtained by checking only the final image.

Family	Tasks / rollouts	Representative tasks	Notable review cases
Blocks: rearrange / stack / obstacle / clutter	6 / 120	pick up the red block and place it on the blue block; swap red/blue blocks; stack blocks on the blue block; place the blue block on the red block; move a cube around an obstacle; pick/place blocks under clutter	wrong-object grasp, OOD placement, obstacle placement, clutter-induced object confusion
Containers and insertion	5 / 100	pour cherries from one bowl into another; place a ball into a cup; place red/blue cubes into box or basket; place plush toys into left/right baskets; put all four blocks into a basket	partial completion, missing container, multi-object omission, unstable pour/transfer
Contact-rich articulation	4 / 80	pull a drawer and close it; pull a drawer, retrieve the red cube, and close it; open a box and pick up the cup inside	phase lag, partial handle engagement, task-config mismatch
Deformable / thin objects	3 / 60	pull toilet paper out and place it on the table; close an open book; stack two paper cups	visually subtle terminal success, failure to extract, incomplete close
Orientation and tools	4 / 80	turn a cup upside down; reset a fallen bottle; place knife and fork on a plate; hang scissors on a rack	high-risk orientation errors, one-object-only completion, drop events
Long-horizon multi-object	3 / 60	swap the positions of the watermelon and the mango; sort panda/elephant/tiger/lion plush toys into baskets; move toys in cluttered scenes	task over-decomposition, phase drift, multiple recoverable failures

Table 13: Task coverage for the semantic-review benchmark. Table 1 reports aggregate metrics and leakage over these families; qualitative examples below illustrate common VSA/post-review interactions. The 500 rollouts were collected across 11 lab days, five workspace layouts, and 43 physical object instances.

Task family	Rollouts	Outcome	Phase	Admission	Leakage
Blocks: rearrange / stack / obstacle / clutter	120	97.5	90.0	97.5	2.4
Containers and insertion	100	97.0	91.0	99.0	0.0
Contact-rich articulation	80	95.0	86.3	95.0	2.8
Deformable / thin objects	60	93.3	83.3	93.3	3.6
Orientation and tools	80	96.3	88.8	96.3	0.0
Long-horizon multi-object	60	95.0	85.0	95.0	4.2
All	500	96.0	88.0	96.4	2.1

Table 14: Per-family semantic-review diagnostics for the Full RoboLineage row in Table 1. Values are percentages. Leakage is the fraction of expert-rejected rollouts in each family admitted to primary training.

Robot / policy family	Rollouts	Outcome	Phase	Admission	Leakage
ARX / ACT-style action chunking	200	96.5	88.5	96.5	1.8
Realman / Diffusion Policy	180	95.0	86.7	95.6	2.6
GALBOT G1 / VLA/LeRobot-compatible workflow	120	96.7	89.2	97.5	2.0

Table 15: Per-robot and per-policy-family semantic-review diagnostics for Full RoboLineage. The split matches the 25-task benchmark: 10 ARX tasks, 9 Realman tasks, and 6 GALBOT G1 tasks. Values are percentages; leakage uses expert-rejected rollouts within each robot group as its denominator.

Qualitative review patterns. We observed four recurring interactions between VSA and post-rollout review. First, online VSA sometimes lagged or confused visually similar objects, such as

red/blue block swaps, while post-review corrected the final outcome from terminal and temporal packets. Second, when VSA detected high-risk events during bottle reset, ball-in-cup, scissors, or drawer failures, post-review usually routed the rollout to review while preserving it as trainable or failure-pool evidence when appropriate. Third, task-config mismatch, such as a drawer rollout evaluated under a bottle-reset phase vocabulary, surfaced as a lineage inconsistency requiring inspection. Fourth, long-horizon and cluttered tasks produced the hardest VSA errors, but packetized review still preserved partial evidence and conservative routing.

J.2 Closed-Loop Recollection Case Study

Case-study plate format. Figure 10 is an appendix plate for the stacking closed-loop case. It uses three columns for the dominant failure stages that emerge across successive policy iterations, and three rows for consecutive frames from one representative failure. The text underneath each column is the corresponding lineage artifact summary: observed evidence, next-collection recommendation, and the observed result after retraining. Each frame reference corresponds to a keyframe strip in the Appendix and release package, where the same evidence packet can be inspected together with its artifact hashes.

The plate illustrates the mechanism behind the closed-loop results: lineage artifacts make the current dominant failure explicit, turn it into a concrete collection target, and link the observed changes after retraining to the corresponding dataset update.

K Expanded Limitations and Future Work

Evaluated scope. RoboLineage governs the data lifecycle around the lab's existing learner and collection interface. Its decisions depend on the supplied task configuration and the quality of robot streams. Evaluation criteria and safety thresholds also come from the lab's configuration. Our experiments cover routine policy-iteration cycles carried through this interface with agent-generated evidence and governed artifacts.

Semantic uncertainty. RoboLineage can flag uncertainty and preserve the evidence behind a decision. Its semantic interpretation can still be wrong, especially when the decisive event is invisible to cameras. Force-dominant assembly, for example, may need force or tactile measurements to establish success. Other criteria may depend on sound or an external measurement. Our evaluation emphasizes tabletop and visually grounded manipulation. Broader mobile manipulation and deformable-object tasks need evaluation with evidence suited to their failure modes. Task-specific sensor validation will also be needed where visual review is insufficient.

Model backend. The main evaluation uses one frozen Claude Sonnet 4.6 configuration. Appendix Table 12 reports hosted Gemini/Qwen and local open-weight Qwen sensitivity replays; broader model tuning and prompt-perturbation studies remain future work.

Governance beyond one lab. The current implementation is designed for a lab or small fleet. Sharing across labs would require stable rollout identifiers and checks that another lab can interpret the records. A federated version would also need to preserve lineage when the underlying data stay private. Researchers could then share a dataset with the evidence behind its admission decisions and the history of policies trained and evaluated on it.

Richer correction and deployment-time collection modes. The evaluated workflow covers ordinary rollout capture and post-hoc review. One extension is raw collection for generalization studies, without online semantic review. Another is reference correction: a policy proposes actions for inspection without driving the robot. Deployment monitoring could record when an operator takes over and how the execution is corrected, preserving those traces in lineage. Policy-foresight or AR trajectory displays could help the operator inspect model intent while remaining separate from

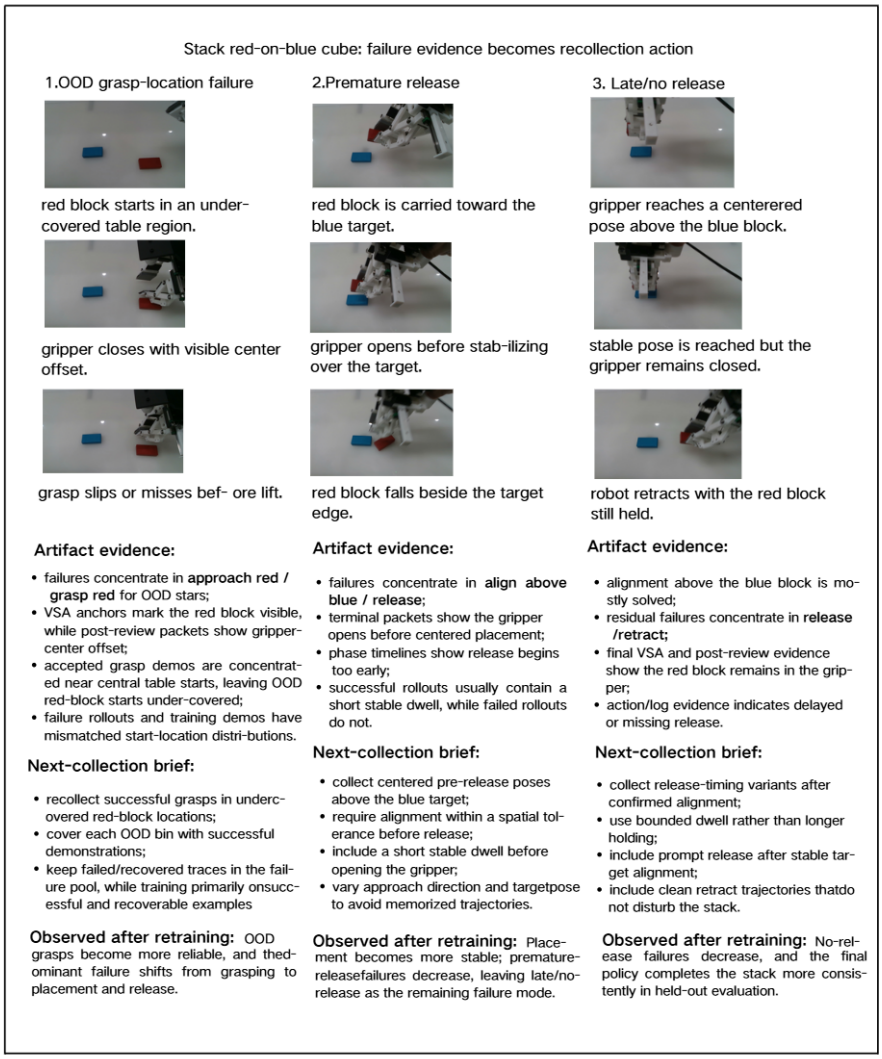


Figure 10: Stack red-on-blue closed-loop case. The plate follows three dominant failure stages through evidence artifacts, next-collection briefs, and observed changes after retraining.

governance agents. The artifact model can represent these modes, but their interfaces and safety procedures still need validation.

Lifelong robot memory. RoboLineage currently summarizes each policy iteration and links it to previous policies. A longer-lived memory could retain recurring failures and the data that helped resolve them. It could also track sensor reliability and revisit release decisions after deployment. These lessons should remain linked to inspectable evidence and versioned records across iterations.

Learning governance decisions. Our dataset and deployment transition rules are explicit by design. Over time, systems may learn better priors for which ambiguous rollouts are worth human review, which failure pools improve robustness, or which evaluation regressions predict deployment risk. The challenge is to learn these priors without removing accountability from the final decision.

Benchmarks for data ecosystems. Robot learning benchmarks usually score policies. The workflow that produced a policy also deserves evaluation. A benchmark could measure how reliably episodes are reviewed and how quickly they become training-ready data. It could test whether failure diagnosis leads to useful recollection. Complete lineage would let evaluators trace those decisions and check whether the training run can be reproduced.

Community interface standards. The field would benefit from simple standards for robot rollout manifests, semantic review artifacts, dataset admission records, and policy metadata. RoboLineage is one concrete proposal, but the deeper goal is interoperability. A policy trained in one framework should be able to carry its data lineage into another evaluation or deployment environment.