ROBOCOMPILER: AGENTIC EXPERIENCE COMPILATION FOR ROBOT POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Successful demonstrations provide correct behavior, but their value for post-training depends on what a robot policy can already do. Long, reliable motions can dominate replay, leaving brief, difficult interactions with little supervision. We study how an agent can use the policy's execution experience to organize further learning. We present ROBOCOMPILER, a framework for *agentic experience compilation*. A multimodal agent compares policy successes and failures with expert demonstrations, relates them through observable task phases, and maps diagnosed difficulties to relevant expert behavior. It then writes a supervision program that concentrates training around these interactions while preserving the surrounding skill. The program determines both how much exposure each phase receives and which observations provide that supervision. Across ten RoboTwin 2.0 tasks, ROBOCOMPILER improves $\pi_{0.5}$ mean success by 16.3 percentage points over full-demonstration training under the same learner-update budget. Experiments with LingBot-VLA and three physical robot tasks support its use across learners and in real manipulation. Component studies show the value of policy evidence and phase allocation, with a further, task-dependent benefit from local selection. These findings demonstrate a practical route from an agent's interpretation of robot experience to improved policy learning.

1 INTRODUCTION

Training a robot requires decisions about what experience to collect, how to interpret its behavior, and what it should learn next. Agents increasingly participate in these decisions through reward design, curricula, data organization, and coordination of collection and learning (Ma et al., 2024; Liang et al., 2025; Ahn et al., 2024; Luo et al., 2026). Advances in multimodal reasoning, long-context processing, and tool use make it practical to inspect heterogeneous robot records (Comanici et al., 2025; OpenAI, 2025). This creates an opportunity to connect an agent's understanding of execution to concrete choices in policy training.

For pretrained robot policies, a central choice is how to use successful demonstrations for adaptation. A policy may reach the workspace, acquire an object, and transport it near the goal, yet fail at a short interaction that determines success. Expert demonstrations contain the needed behavior together with the rest of the skill. Their usefulness depends on the learner: the same execution may warrant different training emphasis for a policy that struggles to acquire an object and one that loses it during placement.

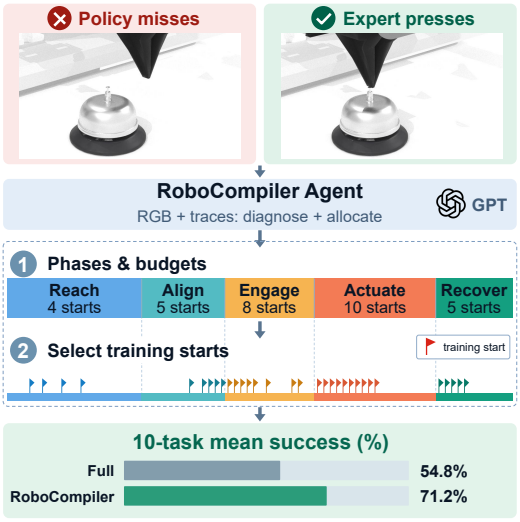

Figure 1: **Turning execution evidence into supervision.** The agent uses policy–expert differences to assign phase budgets and select training starts. Bell illustrates the saved five-phase allocation; bars report ten-task $\pi_{0.5}$ means under equal training updates.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Uniform replay assigns exposure largely according to duration: a long approach contributes many examples and a brief alignment contributes few. Temporal selection can improve robot learning by changing this distribution (Wen et al., 2021; Yu et al., 2026). Motion and gripper events identify informative moments in demonstrations. Policy executions add evidence about which interactions the learner handles reliably and where it departs from successful behavior. We study how to use this evidence to allocate expert supervision within a task.

The useful connection extends beyond an episode's success label. In the Bell example in Figure 1, reaching the target does not ensure a successful press. Identifying what to practice requires relating images, robot traces, and task instructions across executions that progress at different speeds or stop at different stages. This comparison can direct supervision toward contact while preserving the approach and completion that make it useful.

ROBOCOMPILER uses a multimodal agent to make these connections. It grounds observable task phases, compares policy successes and failures with experts, and identifies expert intervals relevant to its diagnosis. It then writes a supervision program specifying how much exposure each phase receives and which local observations provide it. Coverage requirements preserve the complete skill as attention shifts toward difficult transitions. We call this *experience compilation*: translating observations about the learner into an executable organization of expert supervision.

Structure gives these decisions concrete roles: phases relate executions, diagnosis links behavior to expert examples, and allocation turns that interpretation into a training distribution. The agent inspects ambiguous events with tools and expresses its decisions as code. Once checked, the program supplies a fixed training subset with the original expert targets. The learner keeps its training objective and ordinary inference procedure, allowing a controlled study of how the agent changes the use of experience.

We evaluate ROBOCOMPILER on ten RoboTwin 2.0 tasks with $\pi_{0.5}$ (Chen et al., 2025b; Physical Intelligence, 2025), five tasks with LingBot-VLA (Wu et al., 2026), and three physical robot tasks. On the ten-task study, it improves mean success over full-demonstration training by 16.3 percentage points. Component controls examine the use of policy evidence, structured authoring, phase coverage, allocation, and local selection. Our contributions are:

- **Experience compilation for robot post-training.** An agent uses policy executions to organize supervision within successful expert demonstrations.
- **A structured supervision program.** Grounded phases and evidence-linked diagnoses guide phase allocations and local selection, concentrating practice while retaining skill coverage.
- **Evaluation across learners and physical tasks.** Comparisons with dense replay, temporal selectors, and component controls establish the procedure's utility and examine its allocation decisions.

2 RELATED WORK

Agent-assisted robot learning. Agents assist robot learning through reward design and curricula (Ma et al., 2024; Liang et al., 2025), simulation task and demonstration generation (Wang et al., 2024; Hua et al., 2024), and autonomous collection with human feedback (Ahn et al., 2024; Wang et al., 2026a). They also coordinate learning: ENPIRE revises training infrastructure and algorithms using physical rollouts, and RoboLineage connects execution, data, training, and evaluation through lifecycle artifacts (Xiao et al., 2026; Luo et al., 2026). In language-model training, CurateEvo and AutoData make curation programmable through executable data-processing algorithms and development feedback (Wang et al., 2026b; Meng et al., 2026). These approaches show how agent reasoning can shape the objectives, conditions, and experience of learning. ROBOCOMPILER brings this role to supervision within successful robot demonstrations. The agent compares policy behavior with experts and writes phase allocations and local selection rules, making its judgments explicit in the training distribution. Compilation finishes before post-training; the learner retains its architecture, objective, expert targets, and ordinary inference procedure.

Learning from policy experience. Policy behavior guides expert queries, intervention, and data acquisition in DAgger and its human-in-the-loop extensions (Ross et al., 2011; Kelly et al., 2019;

Hoque et al., 2022), active multi-task fine-tuning (Bagatella et al., 2024), and RoboPocket (Fang et al., 2026). Execution also informs reuse: Sirius and HELP organize autonomous experience and human recovery data for learning (Liu et al., 2023; Zhai et al., 2026); Demo-SCORE learns from rollout outcomes to filter successful demonstrations at the episode or chunk level, and CUPID estimates demonstrations' effect on policy return (Chen et al., 2025a; Agia et al., 2025). RoboCompiler expresses this dependence through an allocation over the phases of complete successful demonstrations. Its agent relates policy and expert behavior through task phases, examines departures around unsuccessful interactions, and assigns supervision to the relevant expert intervals. Experts supply every training target, including later phases never reached in a failed rollout. This allows diagnosis to concentrate practice around a difficulty while preserving the approach, transition, and completion needed for the whole skill.

Data selection and supervision allocation. Prioritized replay and error-based reweighting use learner signals (Schaul et al., 2016; Liu et al., 2021); data pruning and core-set methods balance example utility and representative coverage (Paul et al., 2021; Sorscher et al., 2022; Sener & Savarese, 2018). Robot-data methods optimize domain mixtures or select demonstrations using information, datamodels, and local gradient compatibility (Hejna et al., 2024; 2025; Dass et al., 2025; Xu et al., 2026). Within trajectories, temporal selection emphasizes changepoints, removes redundant pairs, or retrieves useful sub-trajectories (Wen et al., 2021; Zhang et al., 2026; Memmel et al., 2025); AWE uses waypoints to shorten the decision horizon (Shi et al., 2023). FrameSkip scores action, visual, progress, and gripper cues, combining selected and full-frame training (Yu et al., 2026); PHASER balances sub-skill replay to limit continual-learning interference (Chen et al., 2026). RoboCompiler instead determines phase exposure and local training observations from comparisons between the current policy and experts. The resulting program links diagnosed behavior to a concrete allocation, retains coverage of the skill, and preserves original expert action chunks. Its allocation and placement controls test these two decisions separately (Section 4.3), connecting the authored supervision distribution to policy learning.

3 RoboCompiler

RoboCompiler uses policy executions to decide which parts of successful demonstrations deserve more training exposure (Figure 2). The agent relates policy and expert executions through task phases, diagnoses relevant departures, and writes a program that allocates supervision across phases and selects observations within them. This concentrates practice while preserving the behavior needed to reach and complete the difficult interactions. A shared prompt and inspection tools guide authoring; execution fixes the training subset before post-training.

3.1 Problem formulation

We are given a pretrained policy θ_0, a task description, diagnostic policy rollouts $\mathcal{D}^{\text{diag}} = \{\tau_j^{\text{diag}}\}_{j=1}^{M}$, and successful expert demonstrations $\mathcal{D}^{+} = \{\tau_i^{+}\}_{i=1}^{N}$. Rollouts contain camera observations, robot states, executed actions, and episode outcomes. They inform selection; the experts supply all training targets. The goal is to improve the policy's success on unseen task instances through post-training on selected expert experience.

A *training start* is a time index t at which an expert observation o_{it} and robot state s_{it} condition an action chunk $a_{i,t+1:t+H}$ of horizon H. Demonstration i has L_i legal starts. We retain K distinct starts per demonstration, where $1 \le K \le \min_i L_i$. A supervision program P produces

$$A_i = P(\tau_i^{+}) = \{t_{i1} < \cdots < t_{iK}\} \subseteq \{0, \ldots, L_i - 1\}, \qquad |A_i| = K. \tag{1}$$

Here K controls the number of retained conditioning observations. Every selected start keeps its original expert action chunk. Uniform sampling over the selected tuples gives the post-training objective

$$\mathcal{L}_P(\theta) = \frac{1}{NK} \sum_{i=1}^{N} \sum_{t \in A_i} \ell(\theta; o_{it}, s_{it}, a_{i,t+1:t+H}), \tag{2}$$

with the learner's original loss ℓ. We compare selections under a common update budget. Exact-K methods give each demonstration equal sampling mass; Full samples all legal starts. Appendix E specifies the shared learning and evaluation protocol.

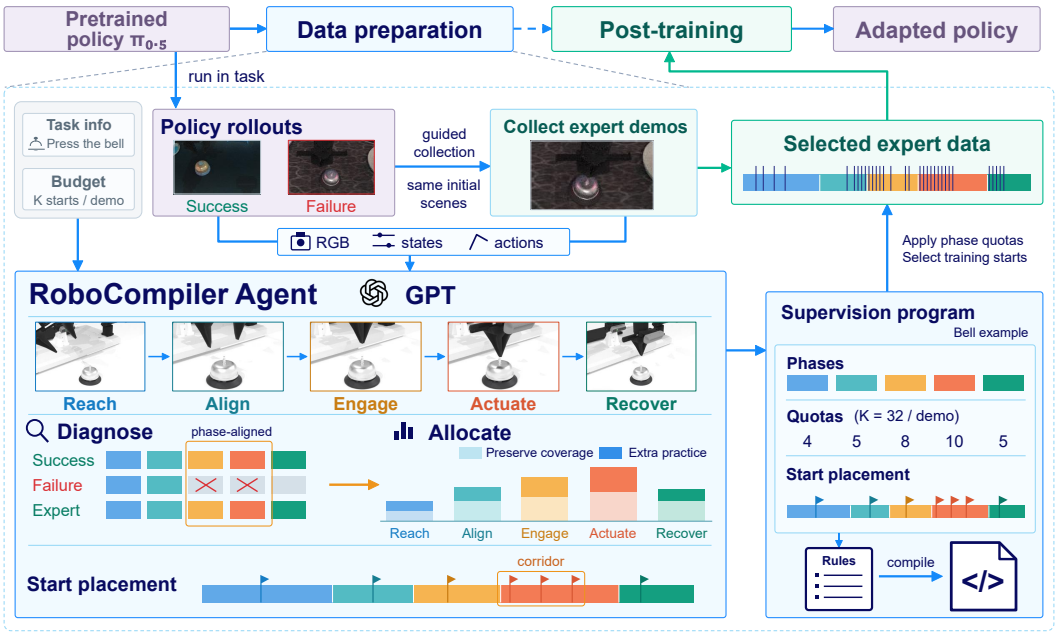

Figure 2: **RoboCompiler workflow.** Policy rollouts and expert demonstrations provide RGB, states, and actions. Together with the task and budget, they guide phase alignment, diagnosis, allocation, and local selection. The resulting program selects expert data for post-training. In the main Exact setting, successful experts are collected from all ten diagnostic initial scenes. The five-phase Bell example illustrates the allocation and placement decisions.

3.2 Grounding Task Progress in Observable Phases

Clock time provides a poor basis for comparing executions that proceed at different speeds or stop at different stages. The agent instead describes progress through observable subgoals, such as acquisition, stable grasp, transport, contact, or release. These are illustrative roles; the supplied trajectories determine the actual structure. Images establish task meaning, while gripper changes and motion suggest candidate boundaries.

The agent inspects contact sheets, selected frames, and numerical windows from robot-state and action traces. It records a phase graph $\mathcal{G} = (V, E)$ whose nodes describe subgoals with entry and exit evidence and whose edges relate task progress. For selection, each expert receives a temporal partition into disjoint phase intervals, grounded separately to accommodate its timing. This gives diagnosis and allocation a common reference: an attempted contact can be related to expert contact even at a different timestamp. Complete experts also expose later stages absent from unsuccessful rollouts, allowing the program to preserve the full skill.

3.3 Connecting Policy Difficulties to Expert Behavior

Policy successes establish what the learner can execute; failures identify behavior for closer inspection. The agent relates rollouts to expert phases through observable events and examines surrounding windows. For each supported difficulty, a diagnosis card records the observed departure, policy evidence, a successful contrast when available, plausible explanations, and expert intervals nominated for additional supervision.

Inspection extends backward from terminal failure. A failed placement may follow an unstable acquisition, and repeated motion near a bell may follow poor alignment. The agent seeks the earliest departure supported by the views and traces, recording alternatives when evidence is ambiguous. Tool use enables closer inspection and numerical comparisons of recorded channels. The resulting diagnosis connects a policy difficulty to correct expert behavior to practice. Policy actions supply evidence throughout; every training target comes from an expert.

3.4 ALLOCATING SUPERVISION ACROSS PHASES

Phase duration determines the exposure supplied by dense replay, but it need not reflect the learner's need for practice. ROBOCOMPILER makes that allocation explicit. For phase v in demonstration i, the program assigns an integer quota k_{iv}, with $\sum_v k_{iv} = K$. Under the uniform sampler in Equation 2, that phase receives k_{iv}/K of the demonstration's training exposure.

The agent specifies minimum coverage for required phases and allocates the remaining starts using the diagnosis and successful contrasts, subject to each phase's capacity. These quotas are authored judgments with recorded evidence. Extra exposure can support an interaction and the preparation on which it depends; minima retain reaching, maintaining, and completing the skill, including stages absent from failed rollouts.

In the saved Bell case in Figure 2, difficulty establishing a verified press directs attention to expert alignment, engagement, and actuation. The five quotas $(4, 5, 8, 10, 5)$ allocate eighteen starts to engagement and actuation, retaining fourteen across reach, alignment, and recovery. Local rules then choose the observations supplying this exposure. This makes the diagnostic judgment concrete before training: the record identifies the difficulty, the relevant expert behavior, and the resulting allocation (Appendix H). Section 4.3 tests quotas separately from local placement.

3.5 SELECTING OBSERVATIONS WITHIN EACH PHASE

A quota sets how often a phase is practiced; placement chooses the observations from which actions are learned. Observations before contact, during contact, and after stabilization can be close in time yet require different predictions. The agent can retain a short window of consecutive or nearby training starts around a supported transition, which we call a *corridor*, or rank candidate starts using the recorded evidence. A grasp corridor can include alignment, closure, and verification, alongside broader phase coverage. Grounded intervals, episode-specific parameters, spacing, capacity handling, rounding, and tie breaking are saved with the rules to produce definite selections.

Each start retains its original future-action chunk. Earlier observations can therefore supervise later-phase actions, and neighboring starts can share targets. Selection changes their conditioning observations and sampling frequency. We distinguish starting-observation selection from future-action coverage (Appendix E.4).

For example, the saved Bell program places its ten actuation starts near the beginning of a nineteen-start phase. Their original 32-step action chunks extend through the press and recovery. The corridor therefore pairs pre-contact observations with successful subsequent actions, concentrating practice on entering the interaction while preserving its expert continuation.

3.6 EXECUTING THE PROGRAM FOR POST-TRAINING

Execution applies the authored program to every expert. Common checks verify source identities, evidence references, interval bounds, quota feasibility, coverage, and exactly K sorted, unique, legal starts. The resulting indices implement the declared allocation and placement rules over the original expert observations.

After these checks, the program, source bindings, and indices are frozen. Training loads the original expert tuples with no further agent decisions, and the adapted policy uses ordinary inference. Compilation thus makes the agent's interpretation explicit and replayable over the supplied demonstrations. Appendices A and B give the tools, completion rules, and shared prompt; Appendix H follows the saved Bell and Scan allocations, with an executable Bell example in the accompanying code.

RoboCompiler procedure
Input: task, policy rollouts, successful experts, and budget K.
1. Inspect trajectories; ground phase intervals and compare policy behavior with experts.
2. Record diagnoses; author coverage, phase quotas, and local placement rules.
3. Execute the program and check all expert selections.
4. Freeze validated indices; post-train the policy with the original expert targets.
Output: saved supervision program, selected expert starts, and adapted policy.

4 EXPERIMENTS

We compare supervision choices, examine their component decisions, and evaluate a second learner and physical robots.

4.1 EXPERIMENTAL SETUP

Tasks and experience. All simulation evaluations use RoboTwin 2.0's randomized setting. Ten tasks span acquisition, transport, placement, actuation, and coupled manipulation. Each repetition supplies ten successful experts with seed-specific initial-position perturbations, shared across methods. RoboCompiler also reads ten Frozen-policy rollouts, including successes and failures. In the main Exact setting, experts complete all ten diagnostic initial scenes, including those solved by Frozen. Compressed methods share a task-level K retaining around one third of legal starts, inspired by FrameSkip (Yu et al., 2026) (Appendix E).

Each repetition randomizes object positions in twenty held-out test templates. Methods share realized scenes, instructions, and inference action-noise seeds within each learner, task, and repetition. Ten validation scenes are separate from diagnostic and expert sources; tests are separate from all three. The official checker determines success.

Learner and repetitions. The main learner is RoboTwin-pretrained $\pi_{0.5}$ (Motus Robotics, 2025). Conditions share its objective, batch size 32, and 2,500 updates with 83 warmup updates. We save EMA checkpoints every 500 updates, select the earliest with maximum validation success, and evaluate it on the repetition's test scenes (Appendix E.1). Trained simulation conditions use three separately trained runs; RoboCompiler recompiles with the shared prompt in each. Agent conditions use `gpt-5.6-sol` at `xhigh`. Main tables report mean success; appendices give sample SD across training, scene perturbations, and applicable compilation. Frozen evaluates unchanged weights on the three test realizations. Overall scores weight tasks equally.

Comparators. **Frozen** evaluates the starting policy; **Full** samples all legal expert starts. **Uniform** spreads K starts throughout each demonstration, **Motion** prioritizes action variation, and **Handcrafted (HC)** selects event windows. **FrameSkip-style (FS)** adapts FrameSkip's action, visual, and temporal importance cues to exact-K selection (Yu et al., 2026). This holds counts, episode weighting, and learner settings fixed across compressed selectors (Appendix E.5.1). **RoboCompiler** compiles its selection from policy and expert experience. All selected starts retain their original expert action chunks.

4.2 MAIN RESULTS

Table 1: **Success across ten tasks.** Mean success (%) across three repetitions; methods share each repetition's N20 roster. Sample SD: Appendix C. Bold: highest mean. Mean: equal task weights. HC: Handcrafted. †FS is our exact-K FrameSkip-style adaptation, sharing start counts, episode weighting, and the learner recipe (Appendix E.5.1).

Task (K)	Frozen	Full	Uniform	Motion	HC	FS†	RoboCompiler
Cabinet (88)	43.3	56.7	66.7	76.7	65.0	68.3	**80.0**
Scan Object (60)	18.3	36.7	38.3	48.3	46.7	30.0	**58.3**
Alarm (32)	55.0	63.3	60.0	46.7	55.0	25.0	**80.0**
Bell (32)	48.3	78.3	76.7	80.0	65.0	58.3	**95.0**
Stamp Seal (48)	26.7	28.3	30.0	36.7	40.0	45.0	**46.7**
Place Phone (44)	41.7	60.0	58.3	58.3	28.3	55.0	**71.7**
Rotate QR Code (56)	35.0	63.3	55.0	55.0	53.3	45.0	**65.0**
Place Fan (52)	50.0	66.7	76.7	71.7	63.3	50.0	**78.3**
Diverse Bottles (44)	31.7	48.3	45.0	60.0	33.3	51.7	**73.3**
Object on Scale (48)	45.0	46.7	38.3	53.3	**65.0**	46.7	63.3
Mean	39.5	54.8	54.5	58.7	51.5	47.5	**71.2**

ROBOCOMPILER achieves 71.2% mean success: 16.3 points above Full and 12.5 above Motion, the strongest fixed selector overall. It exceeds Full and Frozen on every task and has the highest mean

on nine. Gains are substantial on Bell, Bottles, and Scan; Stamp, QR Code, and Fan are close, while HC leads on Scale. Across thirty matched task–run pairs, it wins 28 against Full and 27 against Motion (Appendix C).

The strongest fixed selector varies: Motion leads on Scan and Bell, Uniform on Fan, FS on Stamp, and HC on Scale. Policy rollouts can identify difficulties beyond expert salience, including contact and alignment with little motion.

Full gives longer demonstrations more sampling mass; exact-K selectors share episode weights and start counts. The ablations examine how RoboCompiler arrives at its allocation.

4.3 Ablations of the compilation procedure

Table 2 examines five tasks from the ten-task suite, using the main three-run protocol and reference results. Appendix F gives statistics and intervention definitions.

Table 2: **Compilation ablations.** Mean success (%), three runs. Tasks have equal weight. Sample SD and definitions: Appendix F.

Condition K	Bell 32	Scan 60	Phone 44	Alarm 32	Bottles 44	Mean
RoboCompiler	**95.0**	**58.3**	**71.7**	**80.0**	**73.3**	**75.7**
Direct agent	85.0	48.3	61.7	65.0	58.3	63.7
No diagnostics	85.0	46.7	58.3	66.7	60.0	63.3
No coverage	85.0	45.0	58.3	73.3	55.0	63.3
Direct, no diag.	80.0	43.3	61.7	65.0	56.7	61.3
Uniform in phases	90.0	53.3	**71.7**	75.0	68.3	71.7
Duration quotas	83.3	41.7	60.0	66.7	53.3	61.0

Policy evidence and structure. Without policy diagnostics, the agent constructs phases and allocations from expert demonstrations alone, resulting in lower success. Policy evidence helps more when organized into phases and allocations than in direct selection. This 2×2 comparison supports structured authoring of supervision from execution evidence. Both procedures share the model, reasoning effort, evidence, and tools. Direct selection returns indices and a rationale; RoboCompiler requires phases, diagnosis, and allocation code.

Coverage, allocation, and placement. Removing coverage lowers mean success on all five tasks, most on Bottles, followed jointly by Scan and Phone. Holding phases and uniform placement fixed, compiled quotas yield 71.7% success versus 61.0% for duration-proportional quotas. Phase duration is an insufficient guide to training emphasis. Local placement adds a further 4.0 points on average, improving four task means while matching Phone. Quotas govern phase exposure; local rules choose the conditioning observations that supply it.

4.4 Sensitivity to the supervision budget

Figure 3 varies retained starts under the same update budget. Each task performs better over an intermediate range than at the smallest budget or Full. Phone benefits from expanding small budgets; Bell sustains high success across several settings. This agrees with FrameSkip's retention study (Yu et al., 2026).
The pattern is consistent with balancing context and concentration: small budgets restrict coverage around interactions; larger budgets spread exposure across more observations. Broad favorable ranges leave latitude in choosing a budget; phase quotas determine its allocation. When an action chunk spans a transition, a small subset may retain the future action targets yet omit observations from which the learner needs to predict them. This connects budget sensitivity to the observation-level coverage examined next.

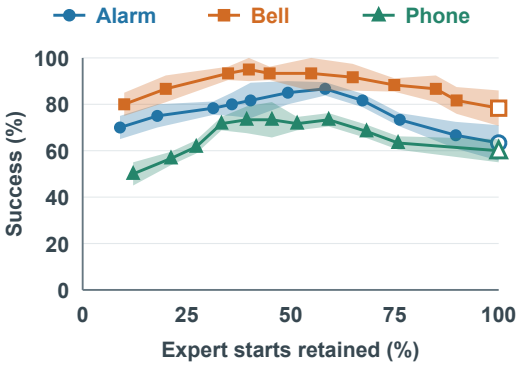

Figure 3: **Supervision budget.** Mean ± SD, three runs. Full (open markers) also changes episode weighting (Appendix D).

4.5 How the program changes supervision

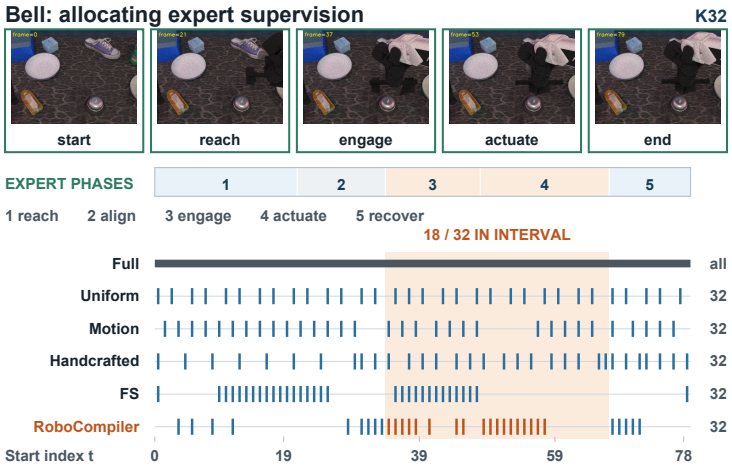

Figure 4: **Bell supervision.** Full and five K32 selections on the same expert. RoboCompiler places 18 starts in engagement and actuation (shaded), retaining 14 across reach, alignment, and recovery.

Bell illustrates how phase budgets and local placement shape supervision (Figure 4). Motion gives actuation 4.9 of its 32 starts on average; RoboCompiler gives it ten. HC also gives this phase 8.9 starts, but spreads them across the press, while the compiled corridor gathers observations leading into it. FS chooses none within actuation despite high future-action coverage. Appendix H examines the saved indices and matched Bell behavior.

A separate saved-run cohort provides a behavioral contrast on the same scene: Frozen hovers near the bell; Full and Motion attempt a downward motion but time out; RoboCompiler completes the interaction. Appendix H.1 relates this observation to the full matched cohort, including both resolved failures and retained successes.

Scan shows these allocation choices over a longer chain. Motion selects no start in its terminal phase, yet earlier chunks cover almost all original action indices. FS omits pregrasp starts; RoboCompiler retains all seven roles and places short corridors across transitions. Similar action-target coverage can therefore coexist with different conditioning observations (Appendices H.2 and E.4).

The fixed selectors rank moments using features of successful demonstrations. RoboCompiler additionally compares these demonstrations with policy executions to determine phase exposure and local training starts, while retaining preparation and completion. The quota and placement ablations test these decisions across tasks; the saved Bell and Scan programs show what changed in the training inputs.

4.6 Source-setting comparison

Exact pairs experts with diagnostic initial scenes; Random collects them independently. Full and RoboCompiler share each condition's experts and each repetition's main test roster. Exact reuses main runs; Random trains separately under the same protocol.

Matching scenes raises both means. RoboCompiler also improves over Full with independent experts: phases relate executions across configurations of the same task (Appendix C).

Table 3: **Expert sources.** Mean success (%), three runs. Frozen and Exact reuse Table 1. Sample SD: Appendix C.

Source	Program	Alarm	Bell	Phone	Mean
—	Frozen	55.0	48.3	41.7	48.3
Random	Full	55.0	75.0	51.7	60.6
	RoboCompiler	71.7	93.3	70.0	78.3
Exact	Full	63.3	78.3	60.0	67.2
	RoboCompiler	80.0	95.0	71.7	82.2

4.7 Evaluation with LingBot-VLA

LingBot-VLA-4B (Wu et al., 2026) shares the expert data, budgets, Exact sources, and three-run protocol, supplying its own diagnostics. Test rosters differ between learners; methods within each learner share scenes and noise. LingBot retains its action-expert recipe and 50-step chunks (Appendix E.2).

RoboCompiler reaches 88.0%, gaining 8.7 points over Full and 7.7 over FS-style. It leads on four tasks and ties Full on Phone. Gains are largest on Alarm and Bell; Full already performs strongly on Scan and Phone. Both learners benefit despite different starting abilities and training recipes.

Table 4: **LingBot-VLA.** Mean success (%), three runs. RC: RoboCompiler; Bott.: Diverse Bottles. Dispersion: Appendix C.1.

Method	Bell	Scan	Phone	Alarm	Bott.	Mean
Frozen	45.0	76.7	81.7	35.0	78.3	63.3
Full	66.7	91.7	**96.7**	58.3	83.3	79.3
Uniform	61.7	91.7	95.0	53.3	85.0	77.3
Motion	68.3	91.7	93.3	55.0	83.3	78.3
HC	66.7	83.3	95.0	60.0	81.7	77.3
FS	66.7	93.3	95.0	60.0	86.7	80.3
RC	**88.3**	**95.0**	**96.7**	**71.7**	**88.3**	**88.0**

Compilation cost. Independent measurements on Bell, Scan, and Phone compare the authoring procedures under shared model settings and evidence access (Table 5; input, cache, and tools in Appendix G). Compilation runs once per training run; its fixed selection supplies all updates, with no compiler calls at deployment.

Table 5: **Compilation usage.** Three independent task measurements per procedure. Output includes reasoning.

Procedure	Output (k)	Min.
RoboCompiler	33.1–40.4	27.5–32.5
Direct	15.6–20.1	15.0–21.7

4.8 Real-robot evaluation

For each ARX task, we adapt OpenPI $\pi_{0.5}$ on fifty demonstrations for 15,000 updates to obtain Frozen. Its rollouts in ten new configurations are paired with successful expert demonstrations from the same layouts. Compilation follows the simulation procedure.

All methods use the same Frozen policy, ten new expert demonstrations, and 6,000 updates; selectors retain around one third of starts. Twenty test configurations are separate from adaptation and diagnosis (Appendix C.2). RoboCompiler leads on all three tasks in this single-seed physical study.

Table 6: **Real-robot results on ARX.** Success (%), one training seed and twenty unseen trials per condition. RC: RoboCompiler.

Method	Stack	Drawer	Button	Mean
Frozen	30.0	50.0	40.0	40.0
Full	65.0	80.0	80.0	75.0
Uniform	65.0	75.0	70.0	70.0
Motion	70.0	80.0	75.0	75.0
HC	75.0	85.0	80.0	80.0
FS	75.0	85.0	80.0	80.0
RC	**85.0**	**90.0**	**90.0**	**88.3**

5 Discussion and Conclusion

RoboCompiler turns an agent's interpretation of robot experience into explicit decisions about policy supervision. Grounded phases connect policy behavior to successful experts; allocations and local rules determine what the learner practices while preserving the surrounding skill. Experiments across two learners and physical tasks demonstrate the utility of this procedure. The component controls support policy evidence, coverage, and phase allocation, with a task-dependent benefit from local placement. A saved program carries these decisions into ordinary post-training, providing a concrete interface through which agents can participate in policy improvement.

The current study considers single-task adaptation from small expert portfolios, with one training seed per physical condition. The controls evaluate the observed authoring procedures; finer attribution of diagnosis and reasoning effort remains open. Learning the supervision budget, scaling to larger task collections, and compiling again as the policy improves are natural extensions. As competence changes, an interaction that needs concentrated practice may later require only maintenance coverage. New rollouts could guide the same expert portfolio toward the learner's remaining difficulties. Appendix J discusses these limits and the broader research opportunities enabled by experience compilation.

REFERENCES

Christopher Agia, Rohan Sinha, Jingyun Yang, Rika Antonova, Marco Pavone, Haruki Nishimura, Masha Itkina, and Jeannette Bohg. CUPID: Curating data your robot loves with influence functions. In *Proceedings of The 9th Conference on Robot Learning*, volume 305 of *Proceedings of Machine Learning Research*, pp. 2907–2932, 2025. URL https://proceedings.mlr.press/v305/agia25a.html.

Michael Ahn, Debidatta Dwibedi, Chelsea Finn, et al. AutoRT: Embodied foundation models for large scale orchestration of robotic agents. *arXiv preprint arXiv:2401.12963*, 2024.

Marco Bagatella, Jonas Hübotter, Georg Martius, and Andreas Krause. Active fine-tuning of multi-task policies. *arXiv preprint arXiv:2410.05026*, 2024.

Annie S. Chen, Alec M. Lessing, Yuejiang Liu, and Chelsea Finn. Curating demonstrations using online experience. In *Proceedings of Robotics: Science and Systems*, 2025a. URL https://www.roboticsproceedings.org/rss21/p071.pdf.

Tianxing Chen, Zanxin Chen, Baijun Chen, et al. RoboTwin 2.0: A scalable data generator and benchmark with strong domain randomization for robust bimanual robotic manipulation. *arXiv preprint arXiv:2506.18088*, 2025b.

Ziyang Chen, Shaoguang Wang, Weiyu Guo, Qianyi Cai, He Zhang, Pengteng Li, Yiren Zhao, and Yandong Guo. PHASER: Phase-aware and semantic experience replay for vision-language-action models. *arXiv preprint arXiv:2606.03598*, 2026. URL https://arxiv.org/abs/2606.03598.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. URL https://arxiv.org/abs/2507.06261.

Shivin Dass, Alaa Khaddaj, Logan Engstrom, Aleksander Madry, Andrew Ilyas, and Roberto Martín-Martín. DataMIL: Selecting data for robot imitation learning with datamodels. *arXiv preprint arXiv:2505.09603*, 2025. URL https://arxiv.org/abs/2505.09603.

Junjie Fang, Wendi Chen, Han Xue, et al. RoboPocket: Improve robot policies instantly with your phone. *arXiv preprint arXiv:2603.05504*, 2026.

Joey Hejna, Chethan Bhateja, Yichen Jiang, Karl Pertsch, and Dorsa Sadigh. Re-Mix: Optimizing data mixtures for large scale imitation learning. *arXiv preprint arXiv:2408.14037*, 2024. URL https://arxiv.org/abs/2408.14037.

Joey Hejna, Suvir Mirchandani, Ashwin Balakrishna, Annie Xie, Ayzaan Wahid, Jonathan Tompson, Pannag Sanketi, Dhruv Shah, Coline Devin, and Dorsa Sadigh. Robot data curation with mutual information estimators. *arXiv preprint arXiv:2502.08623*, 2025. URL https://arxiv.org/abs/2502.08623.

Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S. Brown, and Ken Goldberg. ThriftyDAgger: Budget-aware novelty and risk gating for interactive imitation learning. In *Proceedings of the 5th Conference on Robot Learning*, volume 164, pp. 598–608, 2022. URL https://proceedings.mlr.press/v164/hoque22a.html.

Pu Hua, Minghuan Liu, Annabella Macaluso, et al. GenSim2: Scaling robot data generation with multi-modal and reasoning LLMs. *arXiv preprint arXiv:2410.03645*, 2024.

Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J. Kochenderfer. HG-DAgger: Interactive imitation learning with human experts. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019. URL https://arxiv.org/abs/1810.02890.

William Liang, Sam Wang, Hung-Ju Wang, Osbert Bastani, Dinesh Jayaraman, and Yecheng Jason Ma. Environment curriculum generation via large language models. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 433–454, 2025. URL https://proceedings.mlr.press/v270/liang25a.html.

Evan Zheran Liu, Behzad Haghgoo, Annie S. Chen, Aditi Raghunathan, Pang Wei Koh, Shiori Sagawa, Percy Liang, and Chelsea Finn. Just train twice: Improving group robustness without training group information. *arXiv preprint arXiv:2107.09044*, 2021. URL https://arxiv.org/abs/2107.09044.

Huihan Liu, Soroush Nasiriany, Lance Zhang, Zhiyao Bao, and Yuke Zhu. Robot learning on the job: Human-in-the-loop autonomy and learning during deployment. In *Proceedings of Robotics: Science and Systems*, 2023. URL https://arxiv.org/abs/2211.08416.

Qian Luo, Wentao Guo, Zhennan Qin, Nanchun Guo, Yunhan Zhao, Yi Ma, and Yanchao Yang. RoboLineage: Agent-native data lifecycle governance across robot policy iterations. *arXiv preprint arXiv:2606.22142*, 2026. URL https://arxiv.org/abs/2606.22142.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.12931.

Marius Memmel, Jacob Berg, Bingqing Chen, Abhishek Gupta, and Jonathan Francis. STRAP: Robot sub-trajectory retrieval for augmented policy learning. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/a06e129e01e0d2ef853e9ff67b911360-Abstract-Conference.html.

Yan Meng, Dhruv Srikanth, Bingchen Zhao, Zhengyao Jiang, and Yuxiang Wu. AutoData: Agentic search for pre-training data selection. *arXiv preprint arXiv:2609.19754*, 2026. URL https://arxiv.org/abs/2609.19754.

Motus Robotics. pi0.5 RoboTwin 2.0 checkpoint. Hugging Face model card, 2025. URL https://huggingface.co/motus-robotics/pi0.5_robotwin2. Model revision effc6e1a99bd4d0edf9447e8ba313def67649cfe.

OpenAI. Introducing GPT-5 for developers. https://openai.com/index/introducing-gpt-5-for-developers/, 2025.

Mansheej Paul, Surya Ganguli, and Gintare Karolina Dziugaite. Deep learning on a data diet: Finding important examples early in training. In *Advances in Neural Information Processing Systems*, volume 34, 2021. URL https://arxiv.org/abs/2107.07075.

Physical Intelligence. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pp. 627–635, 2011. URL https://proceedings.mlr.press/v15/ross11a.html.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. In *International Conference on Learning Representations*, 2016. URL https://arxiv.org/abs/1511.05952.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1708.00489.

Lucy Xiaoyang Shi, Archit Sharma, Tony Z. Zhao, and Chelsea Finn. Waypoint-based imitation learning for robotic manipulation. *arXiv preprint arXiv:2307.14326*, 2023. URL https://arxiv.org/abs/2307.14326.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. Beyond neural scaling laws: Beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems*, volume 35, 2022. URL https://arxiv.org/abs/2206.14486.

Boyuan Wang, Zhenyuan Zhang, Zhiqin Yang, et al. Zero2Skill: Bootstrapping robot skills through autonomous data collection, training, and deployment. *arXiv preprint arXiv:2607.14047*, 2026a.

Dingzirui Wang, Xuanliang Zhang, Keyan Xu, et al. CurateEvo: Data-curation evolving for agentic post-training. *arXiv preprint arXiv:2607.06140*, 2026b.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. GenSim: Generating robotic simulation tasks via large language models. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=OI3RoHoWAN.

Chuan Wen, Jierui Lin, Jianing Qian, Yang Gao, and Dinesh Jayaraman. Keyframe-focused visual imitation learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 11123–11133, 2021. URL https://proceedings.mlr.press/v139/wen21d.html.

Wei Wu, Fan Lu, Yunnan Wang, Shuai Yang, Shi Liu, et al. A pragmatic VLA foundation model. *arXiv preprint arXiv:2601.18692*, 2026. URL https://arxiv.org/abs/2601.18692.

Wenli Xiao, Jia Xie, Tonghe Zhang, et al. ENPIRE: Agentic robot policy self-improvement in the real world. *arXiv preprint arXiv:2606.19980*, 2026.

Runze Xu, Yuanfan Xu, Cuijie Xu, Shuang Dai, Yining Li, Yu Wang, and Jincheng Yu. RoboDrop: Curating VLA post-training data via local gradient compatibility. *arXiv preprint arXiv:2609.10021*, 2026. URL https://arxiv.org/abs/2609.10021.

Bin Yu, Shijie Lian, Xiaopeng Lin, Zhaolong Shen, Yuliang Wei, Changti Wu, Hang Yuan, Haishan Liu, Bailing Wang, Cong Huang, and Kai Chen. FrameSkip: Learning from fewer but more informative frames in VLA training. *arXiv preprint arXiv:2605.13757*, 2026. URL https://arxiv.org/abs/2605.13757.

Shaopeng Zhai, Qi Zhang, Tianyi Zhang, Haoran Zhang, Fuxian Huang, Zhanhui Lin, Zijun Xu, and Weinan Zhang. HELP: Human-efficient large-scale robot post-training with rollout segmentation. *arXiv preprint arXiv:2607.09776*, 2026. URL https://arxiv.org/abs/2607.09776.

Yu Zhang, Yuqi Xie, Huihan Liu, Rutav Shah, Michael Wan, Linxi Fan, and Yuke Zhu. SCIZOR: A self-supervised approach to data curation for large-scale imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2026. URL https://ut-austin-rpl.github.io/SCIZOR/.

Reproducibility Statement

Appendices A and B describe the compilation protocol and shared prompt. Appendices C–H specify the result aggregation, learner settings, interventions, and saved supervision examples. The accompanying `robocompiler-supplement.zip` contains code, configurations, prompts, executable examples, and complete policy rollout videos for the three physical tasks and matched Bell/Scan comparisons. Its README gives the entry points. Small result files accompany the analysis code so that the reported means, sample SDs, and reuse relationships can be recomputed. The saved Bell example demonstrates execution of a fixed allocation. Appendix G reports the separate instrumented measurements of compilation usage.

Ethics Statement

The experiments use simulated and physical robots for manipulation tasks and do not involve experiments on human subjects.

AI Use Statement

Generative AI assisted with language polishing, literature search, and writing and running experimental code. Agent-authored supervision programs are part of the studied method. Deterministic code checks their executable outputs for compliance with the selection constraints. The authors are responsible for the experiments, analysis, citations, and manuscript.

A Compilation Protocol

All main RoboCompiler runs use the same prompt, `gpt-5.6-sol` model, `xhigh` reasoning effort, inspection tools, and execution checks. Each run independently compiles its program before training. Appendix B gives the shared prompt; usage measurements and recorded examples appear in Appendices G and H.

A.1 Inputs and source identities

A compilation instance receives the task description, ten complete successful expert demonstrations, ten Frozen-policy diagnostic rollouts, and the per-demonstration budget K. The diagnostic packet contains every rollout from its ten-scene roster, with all successes and failures as observed. Each trajectory carries an episode identity, task instruction, length, and source binding. The evaluator supplies an episode-level success/failure label. Camera observations, recorded robot states, and executed actions are the behavioral evidence available to the agent.

In the main Exact source setting, experts execute successfully from all ten diagnostic initial scenes, including scenes where the policy succeeds. Random-source experiments supply independently collected experts through the same interface. Policy rollouts inform diagnosis and allocation; experts supply the training examples.

The compiler receives no simulator object states, planner traces, checker code or thresholds, or validation/test data. Prompt development uses no test scenes, rollouts, or results. Tasks have separate compilation contexts: the shared prompt and tools are reused, and each task supplies its own evidence. Task-specific conversations, diagnoses, and programs are not passed between tasks.

A.2 Preparing and inspecting trajectories

The simulation packet retains synchronized head-camera and two wrist-camera images, recorded state and action channels, and end-effector poses where recorded. Missing channels remain absent. The inspection tools read the supplied files, extract numerical traces with Python, and display camera frames. They give the agent both an overview of task progress and access to detailed windows around an uncertain event.

For an episode with F frames, the overview renderer takes $m = \min(16, F)$ frames at indices $\mathrm{round}(j(F-1)/(m-1))$, for $j = 0, \ldots, m-1$, using index zero when $m = 1$. Each camera has a separate four-column contact sheet with frame labels. Tiles retain their stored image resolution; the Bell record uses 320×240 pixels per tile. The packet also lists the sixteen largest action-change and visual-change transitions, with earlier indices breaking ties. Action change is the Euclidean difference between consecutive 14-dimensional actions. Visual change is the mean absolute pixel difference divided by 255, averaged over the three cameras.

The landmarks suggest inspection windows. The agent can request supplied frames, read state/action intervals, and write analysis code over the allowed channels. Further queries address unclear events; episode identities and frame indices link them to the program's evidence references.

A.3 Authored decisions and executable rules

The agent writes three connected records. The phase graph describes observable subgoals, their entry and exit evidence, and the corresponding intervals in every expert demonstration. Diagnosis cards record policy departures, supporting windows, successful contrasts when available, alternative explanations, and relevant expert intervals. The allocation specifies phase quotas and minima, local placement rules, and the parameters needed to execute them.

Phase boundaries are grounded for the supplied demonstrations. The saved program can include episode-specific intervals and parameters together with shared rules for allocation and placement. This accommodates variation in timing while making the resulting selection reproducible. The number and names of phases follow the task evidence; the examples in the prompt guide attention to observable subgoals.

The allocation code specifies corridor widths or rankings, spacing, capacity handling, integer-budget correction, and deterministic tie breaking. Executing it yields the selected indices for each expert episode. The term *anchor* in saved code and in the prompt denotes the training start defined in Section 3.1; it uses the same observation, robot state, and original expert action chunk. *Lowering* denotes this execution of the saved allocation and placement rules.

Direct agent selection receives the same raw evidence, model, reasoning effort, inspection tools, and final index checks. It can write analysis code and use the tools to construct indices; its required outputs are indices and a rationale, without the structured records. Conditions without diagnostics remove policy trajectories and their derived annotations. Appendix F specifies these interventions and their construction for each repetition.

A.4 Execution and saved outputs

The agent executes its rules over every expert demonstration. Common code checks source identities, evidence references, interval bounds, phase minima, quota feasibility, and exactly K sorted, unique, legal starts. Compilation ends when the diagnosis and allocation cover all supplied experts and every selection passes these checks. The prompt, program, code, indices, and source identities are saved with their hashes; training loads the original expert examples at the fixed indices.

B SHARED ROBOCOMPILER PROMPT

The shared prompt below uses Agent-Core as RoboCompiler's internal name. Each run instantiates it with its task, evidence packet, and budget.

B.1 MECHANISM, DIAGNOSIS, AND ALLOCATION

```
You are the Agent-Core experience-programming agent for the supplied
fixed robot learner. Your program determines which successful-expert
starting observations receive training exposure under budget K.

Read the task description, the complete successful demonstrations
with synchronized camera observations, robot proprioceptive states,
and actions, and the Frozen-policy rollouts with success/failure labels.
Use only the supplied evidence roster and these recorded modalities.
Infer phases and interaction events from the observations. Do not use
simulator object states, planner traces, or checker code and thresholds.

Describe the task's execution mechanism as a phase graph. Ground each
phase in an observable subgoal, entry and exit evidence, and candidate
intervals in every expert episode. Cover the skill from approach
through verified completion, including acquisition, transport,
retention, release, or stabilization when the task requires them.
Use the task evidence to determine phase boundaries.

Align successful and failed learner behavior to this mechanism. For
each supported weakness, identify the earliest observable divergence,
cite a diagnostic window, provide a successful contrast when available,
and state plausible alternative explanations. Map the weakness to
the corresponding interval in a successful expert demonstration.
Explain which task requirement makes that interval worth additional
training exposure.

Allocate exactly K unique legal starts in every demonstration. Retain
evidence-bearing supervision in each required phase. Allocate
additional starts to supported weak transitions with the context
needed to learn them. Focus local placement around these transitions
with short corridors or evidence-based rankings; retain coverage
elsewhere. Explain the quota and displaced coverage. Record
starting-observation coverage separately from future-action coverage.
```

B.2 OUTPUT AND EXECUTION CONTRACT

```
Return the phase graph, evidence-linked diagnosis cards, per-phase
allocations, per-episode intervals and any specified corridors, the
deterministic lowering rules, and the selected expert indices.

Specify phase minima, corridor widths, spacing, capacity handling,
integer-budget correction, and tie breaking. The implementation must
check source identities, evidence references, interval bounds, and
the requested coverage. Lowering must produce sorted, unique,
in-range anchor sets of cardinality K for every episode.
Report an infeasible budget or an unsupported mechanism explicitly.

Keep each selected observation's original expert action chunk.
Learner rollout actions and failures supply diagnosis only; they
cannot become training labels. Preserve the learner, optimizer,
training schedule, action horizon, evaluation protocol,
embodiment, cameras, physics, and declared demonstration budget.

Freeze the program and its source identities before its associated
learner update.
```

C RESULTS AND AGGREGATION

Each trained simulation condition has three runs with validation-selected checkpoints. The twenty held-out test templates are realized with separately randomized object positions in each repetition. Frozen evaluates the same pretrained policy on these three realizations without retraining. Within each repetition, methods share scenes, instructions, and inference action-noise seeds. RoboCompiler uses the shared prompt with fresh compilation and a different training seed per run. The accompanying code preserves all repetition-level success counts.

Table 7: **Success across ten tasks.** Mean ± sample SD (%) across three repetitions; methods share each repetition's N20 roster. Bold: highest mean. Mean: equal task weights. HC: Handcrafted. †FS is our exact-K FrameSkip-style adaptation, sharing start counts, episode weighting, and the learner recipe (Appendix E.5.1).

Task (K)	Frozen	Full	Uniform	Motion	HC	FS†	RoboCompiler
Cabinet (88)	43.3±5.8	56.7±7.6	66.7±7.6	76.7±7.6	65.0±5.0	68.3±5.8	**80.0±5.0**
Scan Object (60)	18.3±5.8	36.7±5.8	38.3±7.6	48.3±7.6	46.7±2.9	30.0±5.0	**58.3±7.6**
Alarm (32)	55.0±5.0	63.3±7.6	60.0±5.0	46.7±7.6	55.0±5.0	25.0±5.0	**80.0±5.0**
Bell (32)	48.3±7.6	78.3±7.6	76.7±2.9	80.0±5.0	65.0±5.0	58.3±2.9	**95.0±5.0**
Stamp Seal (48)	26.7±5.8	28.3±2.9	30.0±5.0	36.7±5.8	40.0±5.0	45.0±5.0	**46.7±2.9**
Place Phone (44)	41.7±2.9	60.0±5.0	58.3±7.6	58.3±2.9	28.3±2.9	55.0±5.0	**71.7±2.9**
Rotate QR Code (56)	35.0±5.0	63.3±2.9	55.0±5.0	55.0±5.0	53.3±7.6	45.0±5.0	**65.0±5.0**
Place Fan (52)	50.0±5.0	66.7±2.9	76.7±2.9	71.7±2.9	63.3±7.6	50.0±5.0	**78.3±2.9**
Diverse Bottles (44)	31.7±2.9	48.3±2.9	45.0±5.0	60.0±5.0	33.3±2.9	51.7±2.9	**73.3±5.8**
Object on Scale (48)	45.0±5.0	46.7±7.6	38.3±2.9	53.3±2.9	**65.0±5.0**	46.7±5.8	63.3±5.8
Mean	39.5	54.8	54.5	58.7	51.5	47.5	**71.2**

For task t, method m, and run r, let c_{tmr} be the success count and $p_{tmr} = 100c_{tmr}/20$. The table reports

$$\bar{p}_{tm} = \frac{1}{3}\sum_{r=1}^{3} p_{tmr}, \qquad s_{tm} = \sqrt{\frac{1}{2}\sum_{r=1}^{3}(p_{tmr} - \bar{p}_{tm})^2}. \tag{3}$$

The overall score is $\frac{1}{10}\sum_t \bar{p}_{tm}$. Each task–method cell contains sixty test executions across three realizations of twenty scene templates. For trained methods, the sample SD s_{tm} summarizes training and evaluation variation, together with fresh compilation for RoboCompiler. For Frozen, it summarizes scene-position and inference-noise variation for the same pretrained weights. The three repetition-level success rates are the units used to compute this dispersion.

Differences within matched repetitions. For conditions a and b on the same task and repetition, the success-rate difference is $5(c_{tar} - c_{tbr})$ percentage points. Table 8 summarizes these paired totals and their direction across repetitions. Against Full, RoboCompiler has 28 higher counts, one tie, and one lower count across the 30 $\pi_{0.5}$ task–run comparisons; against Motion, it has 27 higher counts and three ties. The smaller local-placement effect varies more across runs, with eight higher counts, four ties, and three lower counts. These summaries preserve the shared-roster comparison and describe observed repeatability. The per-condition SD and the run-level differences summarize distinct aspects of variation; neither is an episode-level paired confidence interval.

Source-setting results. Alarm, Bell, and Phone reuse the corresponding main-table test roster per repetition. Exact + Full and Exact + RoboCompiler reuse their three main training runs and results in the same order; Frozen reuses its three main evaluations. Random conditions train separately on randomly sourced experts under the shared learner and checkpoint-selection protocol. Means weight the three tasks equally.

C.1 LINGBOT-VLA RESULTS

LingBot-VLA uses three runs per trained method. Its twenty-scene test panel differs from the $\pi_{0.5}$ panel, and each repetition randomizes object positions within its scene templates. Methods share realized scenes within each repetition. Frozen evaluates the same pretrained LingBot-VLA policy

Table 8: **Differences within matched repetitions.** Each comparison uses the same task and repetition's test roster. The gap averages task mean differences in percentage points. H/E/L counts task–run pairs in which the first condition has a higher, equal, or lower success count. Each task contributes three pairs; these are descriptive run-level comparisons.

Learner	Comparison	Tasks	Gap	H/E/L
$\pi_{0.5}$	RoboCompiler – Full	10	+16.3	28/1/1
$\pi_{0.5}$	RoboCompiler – Motion	10	+12.5	27/3/0
$\pi_{0.5}$	RoboCompiler – No diagnostics	5	+12.3	14/1/0
$\pi_{0.5}$	RoboCompiler – Direct selection	5	+12.0	13/2/0
$\pi_{0.5}$	RoboCompiler – Uniform within phases	5	+4.0	8/4/3
$\pi_{0.5}$	Uniform within phases – Duration quotas	5	+10.7	14/0/1
LingBot	RoboCompiler – Full	5	+8.7	10/3/2
LingBot	RoboCompiler – FrameSkip-style	5	+7.7	9/3/3

Table 9: **Source setting and temporal compilation.** Success (%, mean ± sample SD), using Table 1's three-run evaluation protocol. Frozen and Exact reuse its results. Mean weights the three tasks equally.

Source	Program	Alarm	Bell	Phone	Mean
—	Frozen	55.0±5.0	48.3±7.6	41.7±2.9	48.3
Random	Full	55.0±5.0	75.0±5.0	51.7±2.9	60.6
	RoboCompiler	71.7±2.9	93.3±2.9	70.0±0.0	78.3
Exact	Full	63.3±7.6	78.3±7.6	60.0±5.0	67.2
	RoboCompiler	80.0±5.0	95.0±5.0	71.7±2.9	82.2

on all three realizations, so its SD reflects evaluation variation. Table 10 reports means and sample SDs; Table 4 gives compact means. Overall scores average these five tasks separately from the ten-task $\pi_{0.5}$ study.

Table 10: LingBot-VLA success (%): mean ± sample SD over three trained runs or, for Frozen, three evaluations of the same pretrained policy without retraining. Each repetition uses 20 test scenes shared across selection methods within the learner and task.

Method Budget	Bell K32	Scan K60	Phone K44	Alarm K32	Bottles K44
Frozen	45.0±5.0	76.7±2.9	81.7±2.9	35.0±5.0	78.3±5.8
Full	66.7±7.6	91.7±2.9	96.7±5.8	58.3±7.6	83.3±5.8
Uniform	61.7±10.4	91.7±2.9	95.0±5.0	53.3±10.4	85.0±10.0
Motion	68.3±7.6	91.7±5.8	93.3±7.6	55.0±5.0	83.3±7.6
HC	66.7±5.8	83.3±2.9	95.0±5.0	60.0±10.0	81.7±2.9
FS	66.7±7.6	93.3±2.9	95.0±8.7	60.0±5.0	86.7±5.8
RoboCompiler	88.3±2.9	95.0±5.0	96.7±5.8	71.7±7.6	88.3±7.6

Starting performance and compilation gains. Table 11 reports the same five tasks across learners and their respective rosters. Each gain compares RoboCompiler with Full within one learner and test roster: four improvements and a Phone tie for LingBot-VLA, and five improvements for $\pi_{0.5}$.

Where the additional gains occur. LingBot-VLA gains are largest on Alarm and Bell, where RoboCompiler also exceeds the strongest fixed comparator. Gains are smaller on Scan and Bottles, while Phone ties Full. Full already performs strongly on Scan and Phone; its different means across learners provide context for the size of their compilation gains.

Consistency across learners. RoboCompiler exceeds the strongest fixed selector for each learner: Motion for $\pi_{0.5}$ and FS for LingBot-VLA. Both benefit under their respective action horizons and adaptation recipes. Component controls examine $\pi_{0.5}$ (Section 4.3); LingBot-VLA evaluates the complete procedure.

C.2 Real-robot setup and results

We evaluate Stack Cube, Open Drawer, and Press Button on ARX, using wrist-mounted RealSense RGB observations at 30 Hz together with robot states and actions. For each task, fifty expert demonstrations and 15,000 optimizer updates adapt the official OpenPI $\pi_{0.5}$ weights to obtain Frozen.

Table 11: Starting performance and within-learner compilation gains on five tasks. The learners use different test panels; methods share realized test scenes within each learner, task, and repetition. Frozen is mean success (%) for unchanged pretrained weights on three position-randomized realizations; Δ is RoboCompiler minus Full in percentage points, computed from the three-run means.

	Frozen		Δ(RoboCompiler, Full)	
Task	$\pi_{0.5}$	LingBot-VLA	$\pi_{0.5}$	LingBot-VLA
Bell	48.3	45.0	16.7	21.7
Scan Object	18.3	76.7	21.7	3.3
Phone	41.7	81.7	11.7	0.0
Alarm	55.0	35.0	16.7	13.3
Diverse Bottles	31.7	78.3	25.0	5.0
Mean	39.0	63.3	18.3	8.7

Frozen then executes ten new initial configurations, providing both successful and failed diagnostic rollouts. An expert supplies a successful demonstration from each of these ten configurations, with objects reset to the corresponding initial positions. All ten configurations contribute paired experts, including those solved by Frozen. Figure 5 illustrates the setups.

RoboCompiler follows the simulation procedure: it compares the policy and expert executions, grounds phases, and assigns quotas and local selections subject to coverage. Stack Cube uses $K = 80$ (30.4% of starts), Open Drawer $K = 72$ (31.1%), and Press Button $K = 24$ (32.5%), with each retention rate computed over that task's ten post-training experts. Each method starts from the same task's Frozen policy and post-trains on the same ten new experts for 6,000 optimizer updates. OpenPI's ARX LoRA configuration uses an effective batch of eight (micro-batch one, accumulation eight), a peak learning rate of 2.5×10^{-5}, and 1,000 warmup updates followed by its configured cosine schedule. Post-training starts with a fresh optimizer and the Frozen normalization statistics; EMA is disabled. Twenty further initial configurations, disjoint from both the fifty pretraining demonstrations and the ten diagnostic/expert configurations, provide the final test set. Table 6 reports one training seed per condition, with success measured over these twenty trials. The accompanying videos show complete executions of the RoboCompiler post-trained policies on all three tasks at their original playback speed.



(a) Stack Cube (b) Open Drawer (c) Press Button

Figure 5: **Physical task setups.** ARX manipulation tasks use the same policy–expert compilation procedure as the simulation study. Backgrounds outside the experimental workspace are masked for presentation.

D Budget Sweep and Normalization

The budget sweep reports three runs per task and K, including three runs for each Full endpoint. Alarm K32, Bell K32, Phone K44, and their Full endpoints reuse the corresponding main-table runs. Means and sample SDs use the same aggregation rule as the main results. The horizontal coordinate in Figure 3 is

$$q(K) = 100 \frac{NK}{\sum_{i=1}^{N} L_i}, \tag{4}$$

where L_i counts the legal starts in demonstration i of the reference expert portfolio in Table 12, and $N = 10$. These lengths provide a common horizontal scale for the three repetitions, whose expert sources have separately randomized initial positions. Each compressed program contributes K starts to every demonstration. Full uses every legal start and is plotted at 100%.

Table 12: Reference expert-source lengths and supervision budgets. Each reference portfolio contains ten demonstrations. Lengths count legal starting observations; each listed budget satisfies $K \leq \min_i L_i$ in that portfolio. The totals provide a shared normalization for the budget-sweep curves.

Task	K	Minimum	Median	Maximum	Total
Put Object Cabinet	88	276	291.5	385	3,006
Scan Object	60	173	176	184	1,769
Alarm	32	83	87.5	96	892
Bell	32	78	79	88	801
Stamp Seal	48	141	145	160	1,485
Place Phone	44	124	131	142	1,319
Rotate QR Code	56	154	163	170	1,630
Place Fan	52	135	150.5	164	1,489
Diverse Bottles	44	122	128.5	145	1,295
Object on Scale	48	145	148.5	167	1,528

Episode lengths vary, so the normalization uses the total number of legal starts in the reference portfolio. Full is a dense-replay reference with its original episode weighting; each curve connects the final RoboCompiler budget to Full at 100% for comparison. Each budget averages three N20 runs, and the shaded band spans one sample SD above and below the mean.

Bell maintains 91.7–95.0% mean success over K28–K52, and Phone reaches 71.7–73.3% over K44–K78, compared with Full's 78.3% and 60.0%, respectively. These broad ranges complement Alarm's intermediate-budget peak and agree with FrameSkip's finding that selective temporal supervision can outperform dense replay across several retention ratios (Yu et al., 2026). Changing K changes the complete allocation, including quotas and conditioning observations. The fixed-K quota and placement controls separately examine these decisions. Together, the studies motivate allocating sufficient exposure to interactions while retaining the surrounding skill, with a task-dependent choice of budget.

E Selector and Training Protocol

Shared learning interface. Each compressed method supplies an episode identity and exactly K sorted, unique, legal starting indices for each of ten successful demonstrations. The learner retrieves the original observation, state, and future-action chunk at each selected start. The $\pi_{0.5}$ simulation implementation uses a 32-step action horizon and an effective batch size of 32. Its trained conditions use 2,500 optimizer updates with 83 warmup updates and select one of five saved EMA checkpoints by validation performance. The $\pi_{0.5}$ policy, LoRA parameterization, and optimizer are shared within a task comparison. Full accesses all legal starts. A reduction in distinct starts can leave substantial overlap in their future-action targets.

Action indexing. The notation in Section 3.1 labels the first target following observation t as action $t+1$. Every selector preserves the learner's original observation–action pairing, action horizon, and end-of-episode padding.

Sampling mass. Sampling a fixed exact-K subset gives every demonstration the same total mass, $1/N$, and every selected start mass $1/(NK)$. Dense replay assigns equal mass to each legal start, so longer demonstrations receive more total mass. Comparisons among the exact-K methods share this episode balancing and isolate the selected positions.

E.1 $\pi_{0.5}$ simulation learner configuration

The simulation starting policy is the PyTorch RoboTwin 2.0 $\pi_{0.5}$ checkpoint released by Motus Robotics (Motus Robotics, 2025). That release was trained on clean and randomized demonstrations from fifty RoboTwin tasks. Our simulation experiment studies further adaptation from this checkpoint. The recorded model revision is `effc6e1a99bd`, with weight-file SHA-256 prefix `de8e03a445f3` and normalization-asset prefix `1dce5ceb9a98`. Full identities are retained in the experiment manifests.

All simulated $\pi_{0.5}$ conditions share the learner configuration and validation-based checkpoint selection in Table 13.

Table 13: Shared $\pi_{0.5}$ simulation post-training configuration. LoRA parameters and the listed non-transformer modules are trained jointly.

Component	Setting
Language-stack LoRA	Rank 16, alpha 16; dropout 0
Action-expert LoRA	Rank 32, alpha 32; dropout 0
LoRA targets	Attention Q/K/V/O and MLP gate/up/down projections
Other trainable modules	Vision tower, multimodal projector, action input/output projections, and time MLPs
Frozen parameters	Pretrained language and action-expert transformer weights
Optimizer	AdamW; peak learning rate 2.5×10^{-5}; $\beta=(0.9,0.95)$; $\epsilon=10^{-8}$; weight decay 10^{-10}
Training budget	2,500 optimizer updates for every main and ablation condition
Schedule	83 warmup updates, then cosine decay toward 2.5×10^{-6}
Batch	Microbatch 16, accumulation 2; effective batch 32
Stabilization	Gradient norm clipped at 1; EMA decay 0.99
Memory strategy	Gradient checkpointing enabled
Saved checkpoints	EMA at updates 500, 1,000, 1,500, 2,000, and 2,500
Validation	Ten shared scenes; task success rate; earliest checkpoint on ties
Reported policy	Selected validation checkpoint; independent N20 test
Precision	bfloat16 autocast; float32 trainable adapter and non-transformer parameters
Action representation	32-step chunks; 14 physical action dimensions; model dimension 32
Inputs	Head and two wrist views (224×224); robot state; task prompt

The loader constructs the original observation/action-chunk examples and applies the model transforms before a selector indexes the resulting dataset. The twelve arm coordinates use delta-joint actions, and the two gripper coordinates remain absolute. The checkpoint's quantile normalization statistics are retained. The stored transition convention pairs observation and state at t with the next recorded action and drops the final observation as a new training start.

Across all methods, chunks extending beyond an episode repeat the final action to complete the 32-step target. These repeated positions enter the loss, which averages over all 32 time positions and 32 action dimensions (including eighteen zero-padded dimensions). Repeated actions add no distinct source indices to the coverage calculation below.

Images undergo aspect-preserving resizing and centered padding to 224×224. Training adds a 95% crop and a rotation sampled from $[-5,5]$ degrees to the head view. All three views receive brightness, contrast, and saturation factors sampled from $[0.7,1.3]$, $[0.6,1.4]$, and $[0.5,1.5]$, respectively. Augmentation parameters are sampled once per camera per batch. Inference uses the deterministic image transforms without these augmentations.

E.2 LingBot-VLA learner configuration

The second learner is the RoboTwin-post-trained LingBot-VLA-4B release (Wu et al., 2026). We use the official trainer at code revision `4eb34b7693a0`, the pretrained checkpoint revision `fb71a2c9749c`, and the released RoboTwin normalization statistics. The selector indexes legal starting observations before the official loader constructs their original future-action chunks. Full and all selectors share the same learner configuration within each task. LingBot-VLA and $\pi_{0.5}$ use different twenty-scene test panels, with method comparisons paired within each learner and repetition.

We train LingBot-VLA's action expert and freeze its vision encoder. The corresponding official flags are `train_expert_only` and `freeze_vision_encoder`, both set to `true`. The official `robotwin_load20000h.yaml` configuration supplies the flow-matching objective and constant learning rate 10^{-4}. We retain 50-step action chunks and ten denoising steps. Microbatch size 4 and gradient accumulation 8 give effective batch size 32. Each method uses 2,500 optimizer updates. The formal three-seed comparison follows the same checkpoint schedule and validation rule as the $\pi_{0.5}$ study: checkpoints every 500 updates, maximum success on ten shared validation scenes, earliest checkpoint on ties, and seed-specific object-position randomization of twenty test templates, with realized scenes and inference seeds shared across methods within that repetition.

The compiler uses the same prompt and tools with LingBot-VLA's own Frozen-policy diagnostics, collected in the same source scenes as the expert demonstrations.

E.3 Training execution and resource use

Each $\pi_{0.5}$ simulation learner run uses one NVIDIA GeForce RTX 5090. The recorded model has 450,442,000 trainable parameters and 3,199,788,032 frozen parameters. Trainable tensors are held in float32; forward computation uses bfloat16 autocast. This includes the LoRA parameters and the additional trainable modules listed in Table 13.

Loading the selected experience. The wrapper verifies the selected global indices and constructs a `torch.utils.data.Subset` of the original transformed dataset. A PyTorch data loader shuffles this subset using a generator initialized with the run's training seed. It uses microbatches of sixteen, zero worker subprocesses, and `drop_last=True`. Each pass visits shuffled examples without replacement; after the pass, iteration restarts with a new shuffle. Training continues until the fixed optimizer-update budget is reached.

Optimizer updates. One optimizer update accumulates two microbatches of sixteen examples. For each microbatch, the model computes its flow-matching loss, averages the returned loss tensor, divides by two for accumulation, and backpropagates. After both microbatches, the implementation clips the trainable gradient norm at 1, assigns the current learning rate to AdamW, applies the optimizer update, and updates the EMA. Gradients are then cleared with `set_to_none=True`. The EMA uses decay 0.99 for every trainable tensor. Every 500 updates, the learner saves the EMA parameters as a candidate checkpoint. Validation selects one of these five checkpoints for the reported test evaluation.

Exact learning-rate schedule. Let $s \in \{0, \dots, U-1\}$ index optimizer updates, with $U = 2500$ and warmup parameter $W = 83$. Define $\eta_{\max} = 2.5 \times 10^{-5}$, $\eta_{\text{end}} = 2.5 \times 10^{-6}$, and $\eta_0 = \eta_{\max}/(W+1)$. The implementation sets

$$\eta(s) = \begin{cases} \eta_0 + (\eta_{\max} - \eta_0)s/W, & s \le W, \\ \eta_{\text{end}} + \dfrac{\eta_{\max} - \eta_{\text{end}}}{2}\left[1 + \cos\left(\dfrac{\pi(s-W)}{U-W}\right)\right], & s > W. \end{cases} \tag{5}$$

Thus the first update uses 2.97619×10^{-7}, the 84th update reaches the peak, and the final update uses approximately 2.50001×10^{-6}. The schedule is indexed by optimizer updates, so both accumulated microbatches contribute to the same update.

Table 14: Recorded resource use for a completed Bell run of the 2,500-update $\pi_{0.5}$ recipe. The run uses one RTX 5090 and an effective batch size of 32.

Task	K	Updates	Training time (h)	Peak allocated (GiB)
Bell	32	2,500	5.98	21.27

The measured training duration is 5.98 hours. The training timer starts after model setup and includes the optimizer loop, checkpoint callbacks, and final EMA and hash operations. Memory is the peak tensor allocation reported by `torch.cuda.max_memory_allocated` after resetting the peak counter at the start of the timed region; one GiB is 2^{30} bytes. These measurements describe the learner execution under the stated recipe.

E.4 Starting observations and future-action coverage

Selecting a start changes both the conditioning observation and the frequency with which its future actions enter training. These quantities need separate accounting. For a saved selection S_i, action horizon $H = 32$, and L_i original transitions, define the covered source-action indices as

$$U_i = \bigcup_{t \in S_i} \{t+1, \dots, \min(t+H, L_i)\}. \tag{6}$$

The retained-start fraction is $\sum_i |S_i| / \sum_i L_i$; the unique future-action fraction is $\sum_i |U_i| / \sum_i L_i$. Padding is excluded. An index identifies a recorded action occurrence, even when its value matches

another action. This calculation measures source-index coverage; observation conditioning, action transforms, sampling multiplicity, and loss masking also affect the training signal.

Table 15: Distinct starts and original future-action coverage in the program examples. All values are percentages pooled over ten demonstrations. Every selector in a row retains the same number of starts. Full covers 100% of starts and original actions.

		Retained	Unique original future actions				
Task	K	starts	Uniform	Motion	HC	FS	RoboCompiler
Alarm	32	35.87	99.89	95.85	100.00	91.03	97.09
Bell	32	39.95	100.00	97.63	100.00	99.38	96.13
Phone	44	33.36	99.77	97.19	69.45	99.01	100.00
Scan	60	33.92	99.89	98.08	100.00	87.62	100.00
Stamp	48	32.32	99.60	97.64	74.68	93.40	100.00

Across these programs, retaining 32–40% of starting observations can preserve nearly all source-action indices through overlapping chunks. For example, Motion's Scan program covers 98.08% of original actions despite selecting no starting observation inside the terminal scan phase. The allocation analysis in Section 4.5 therefore concerns direct supervision from each phase's observations.

The calculation checks all selected indices against the legal episode lengths and verifies each union by set enumeration and interval merging.

E.5 SELECTORS

Uniform. The legal starts are partitioned into K progress bins, and a deterministic representative is selected from each bin. This distributes exposure across the episode without task semantics or policy diagnostics.

Motion. For raw 14-dimensional joint-action vectors $a_0, \ldots, a_L$, the selector scores a legal start t by

$$s_t = \|a_{t+1} - a_t\|_2 + \frac{0.1}{14} \sum_{d=1}^{14} \mathrm{Var}_{u=t+1,\ldots,\min(t+3,L)}(a_{u,d}), \tag{7}$$

using population variance and unscaled float64 action coordinates, including both grippers. Its lookahead is three actions, independently of the learner's target horizon. Starts are ranked by descending score, with earlier indices breaking ties. For all tasks except Cabinet, the selector greedily accepts t only if $|t - u| > 1$ for every previously accepted start u. It stops after K acceptances, returns the indices in temporal order, and rejects a budget it cannot fill. The dedicated Cabinet implementation takes the top K scores without temporal suppression. These selectors read only successful-expert action traces.

Handcrafted. The temporal priors use demonstration end-effector and gripper traces. Alarm/Bell and Scan choose the arm with greater XYZ path length, with the right arm breaking ties, and reject a stationary trace. For Alarm/Bell, p is the first point reaching at least 50% of the total path followed by a low-motion slice of up to five transitions; the speed threshold is the larger of 10^{-4} meters and 5% of peak displacement. Let h be the earliest maximum-height observation through p, and e the earliest minimum-height observation from p through $L-1$. The selector requires a height drop of at least 1 cm and a post-p crossing of 20% of that drop. It samples $8/16/8$ starts from $[0, p-1]$, $[p, e]$, and $[e+1, L-1]$. These intervals are geometric press proxies.

Scan uses the first cumulative-path crossings at 60% and 90%, with boundaries clipped to $p \in [8, L-24]$ and $e \in [p+15, L-9]$. It samples $15/30/15$ starts from the three regions at K60 and rejects a region too short for its quota. In both implementations, a closed interval $[a, b]$ with quota q is sampled at $a + \lfloor j(b-a)/(q-1) \rfloor$ for $j = 0, \ldots, q-1$.

The dedicated Cabinet rule detects one downward 0.5 crossing in each gripper signal. With ordered crossings $f < s$ and $m = s + \lfloor (L-s)/2 \rfloor$, it samples $K/4$ starts from each of $[0, f-1]$, $[f, s-1]$, $[s, m-1]$, and $[m, L-1]$. It rejects an insufficient region.

The six task profiles below construct four event-centered windows, as specified in Table 16. Here $Q(r)$ denotes the first transition whose cumulative active-arm XYZ displacement reaches fraction

r of its total, capped at $L-1$. These profiles compute cumulative displacement without a leading zero; the Scan quantiles above include that zero. Gripper events are absolute changes in coordinates 6 and 13 at least as large as the greater of 10^{-8} and 25% of the peak change; a peak at most 10^{-8} produces no events. Candidates within two indices compete by change magnitude, with earlier events breaking ties. Opening and closing changes can both count. G is the first event, falling back to $Q(.22)$; R is the last of at least two events, falling back to $Q(.92)$.

Table 16: Handcrafted event centers for six tasks, supplementing the dedicated Cabinet, Alarm/Bell, and Scan rules above.

Task	Four centers	Initial starts per center
Stamp Seal	G, $Q(.45)$, $Q(.68)$, subsequent minimum height	12
Place Phone	G, $Q(.55)$, $Q(.82)$, R	11
Rotate QR Code	G, $Q(.42)$, $Q(.74)$, R	14
Place Fan	G, $Q(.42)$, $Q(.74)$, R	13
Diverse Bottles	First left/right gripper events, shared-path $Q(.70)$, latest release proxy	11
Object on Scale	G, $Q(.42)$, $Q(.76)$, R	12

Each center contributes the $K/4$ closest legal starts, with earlier indices breaking distance ties. The union removes duplicates. If fewer than K remain, the selector adds a K-point integer-uniform grid, followed by all legal starts if necessary. It ranks the resulting candidates by distance to their nearest center and returns the first K in temporal order. Thus overlapping windows can change the final allocation per center.

For Diverse Bottles, both arms must move and the shorter XYZ path must be at least 10% of the longer. Each arm's first gripper event falls back to its own $Q(.20)$; shared quantiles use the sum of arm displacements. The last center uses the latest final event from arms with at least two events, falling back to shared $Q(.92)$. Stamp's height minimum is the earliest minimum from $Q(.68)$ onward, capped at $L-1$. These centers are deterministic task priors, with no Frozen-policy diagnosis or learned event detector.

E.5.1 FrameSkip-style selector

FrameSkip-style (FS) adapts FrameSkip's importance cues (Yu et al., 2026) to exactly K fixed starts per demonstration under the shared learner recipe. Action-variation importance (AVI), visual–action coherence (VAC), temporal-position importance (TPI), and event preservation determine the selected subset. This interface holds start counts, episode weighting, and training sampling constant across selectors. The weights $(0.6, 0.2, 0.2)$ follow FrameSkip's published defaults. Its annotation-free Gaussian prior supplies temporal importance from the recorded trajectory, without borrowing RoboCompiler's phase labels. The following fixed scoring and selection rules define the reported adaptation.

Inputs and visual features. Let $b_0, \ldots, b_{L-1} \in \mathbb{R}^{14}$ be the recorded joint-action rows after dropping the terminal raw frame. The selector uses float64 values in their original coordinates. It samples sixteen head-camera observations at indices $j_m = \lfloor m(L-1)/15 \rfloor, m = 0, \ldots, 15$. Frozen DINOv2-small produces a 384-dimensional CLS feature f_m for each image. RGB images are resized bilinearly so the shorter edge is 224 pixels, with the other dimension rounded, then center-cropped to 224×224. Pixel values are divided by 255 and normalized with mean $(0.485, 0.456, 0.406)$ and standard deviation $(0.229, 0.224, 0.225)$. The encoder runs in evaluation mode with gradients disabled.

Action variation. For $t = 1, \ldots, L-1$, the score is

$$A_t = \|b_t - b_{t-1}\|_2 + \frac{0.1}{14} \sum_{d=1}^{14} \mathrm{Var}_{u=t,\ldots,\min(t+2,L-1)}\, b_{u,d}, \qquad A_0 = A_1. \tag{8}$$

Variance uses the population denominator; a one-row window has zero variance. The three-row lookahead is clipped at the episode boundary. This indexing follows the implementation, which prepends the first computed score to the adjacent-difference sequence.

Visual–action coherence. At the sparse observations, define

$$z_m = \frac{\|f_m - f_{m-1}\|_2}{\|b_{j_m} - b_{j_{m-1}}\|_2 + 10^{-8}}, \qquad m = 1, \dots, 15,$$
$$v_m = \min(z_m, Q_{0.95}(z_1, \dots, z_{15})), \quad v_0 = v_1. \tag{9}$$

Quantiles use linear interpolation between adjacent order statistics, as in NumPy's default percentile calculation. The dense VAC score V_t is the piecewise-linear interpolation of (x_m, v_m) at integer t, where $x_m = m(L-1)/15$. The interpolation grid uses these real-valued positions; image extraction and sampled-action lookup use the integer indices j_m.

Temporal prior and normalization. We use FrameSkip's annotation-free Gaussian progress prior,

$$T_t = \exp\left[-\frac{(t/(L-1) - 0.5)^2}{0.2}\right]. \tag{10}$$

For each component vector y, let $\Delta_y = \max_t y_t - \min_t y_t$ and set

$$\mathcal{N}(y)_t = \begin{cases} 1, & \Delta_y < 10^{-6}, \\ (y_t - \min_u y_u)/\Delta_y, & \text{otherwise.} \end{cases}$$
$$I_t = \max\{0,\ 0.6\mathcal{N}(A)_t + 0.2\mathcal{N}(V)_t + 0.2\mathcal{N}(T)_t\}. \tag{11}$$

All normalization is within one demonstration. The temporal prior uses normalized frame position and requires no additional stage annotations.

Preserved events and exact-budget selection. The initial selected set contains both endpoints, every index $t \geq 1$ where either gripper changes by more than 0.1, and every index whose adjacent action difference is at or above the episode's 90th percentile. Grippers are coordinates 6 and 13 under zero-based indexing. The action-change event at t uses $\|b_t - b_{t-1}\|_2$, and its percentile is computed over $t = 1, \dots, L-1$.

The remaining positions are ordered by descending (I_t, t), so later indices break exact score ties. The selector adds previously unselected positions until the set contains K elements, then returns the indices in increasing temporal order. It applies no additional temporal suppression. It rejects nonfinite inputs, an invalid sparse-feature inventory, $L < 16$, $K \notin \{1, \dots, L\}$, or a preserved-event set larger than K. These checks keep the output unique, legal, and exactly budgeted.

Integration with the shared learner. The resulting K-element set supplies all training starts throughout the run. The common data loader shuffles this subset and retrieves each start's original observation and future-action chunk. Under this interface, the exact-K selectors differ in the selected positions while sharing the supervision budget, episode weighting, and sampling rule. Learning-rate warmup, optimizer updates, and validation-based checkpoint selection follow the shared learner recipe.

E.6 Compilation and evaluation in simulation

Expert source settings. The `exact` source condition reuses the diagnostic scene seeds and task settings for successful expert collection. Each repetition applies its own initial-position perturbations to the ten source configurations; its diagnostic rollouts and successful experts are paired in those realized scenes. The recorded instruction, scene manifest, and realized scene identity bind each recollected episode to its source setting. The expert executes the complete task, and the task checker verifies success. The `random` condition collects ten successful demonstrations from independently sampled task settings. Full and the selectors share the resulting expert portfolio within each repetition and source condition.

Supervision budgets. The compiler receives a task-level integer K as an input, fixed across selectors and repetitions. Inspired by FrameSkip's finding that selective temporal supervision can improve adaptation (Yu et al., 2026), we target retention around one third as a common operating scale. Table 12 gives the exact supplied budgets and demonstration lengths. The ratio is $NK/\sum_i L_i$, so

equal per-demonstration budgets need not yield identical task-level fractions. Alarm and Bell use K32, Cabinet K88, Phone K44, and Scan K60. All ten tasks satisfy $K \leq \min_i L_i$; Cabinet has at least 276 legal starts per demonstration for K88. Execution also checks phase-level quota feasibility. The separate sensitivity study examines a range of budgets around these operating values.

Independent compilation and training. Every RoboCompiler run uses the shared prompt in Appendix B and the inspection procedure in Appendix A. The task specification, Frozen-policy diagnostic trajectories, successful demonstrations, and K instantiate its inputs. The agent independently generates the phase graph and allocation program, and the compiler checks the expert-start sets used for training. Each of the three runs uses a different training seed and follows the common training and checkpoint-selection procedure.

Checkpoint selection and test evaluation. We construct separate scene rosters for compilation and training, validation, and testing. The ten validation scenes are disjoint from all diagnostic and expert-source scenes. Testing uses twenty held-out scene templates, disjoint from validation, diagnostics, and expert sources. Object positions are randomized separately for each repetition, yielding sixty realized test layouts across the three runs. The Exact condition shares realized scenes only between policy diagnosis and expert collection. Validation rollouts select checkpoints, and test rollouts supply the reported success rates.

Each trained simulation run saves EMA checkpoints at updates 500, 1,000, 1,500, 2,000, and 2,500. We select the earliest checkpoint with maximum success on ten validation scenes, sharing scenes and inference seeds across methods. Its repetition-specific N20 evaluation uses RoboTwin's `demo_randomized` configuration. Within each learner, task, and repetition, methods share the same realized test scenes, instructions, and inference action-noise seeds. The evaluator checks scene identities and records one success bit per episode, with failures contributing zero. Main comparisons, ablations, source settings, the budget sweep, and LingBot-VLA use three training repetitions with seed-specific scene perturbations. Every sweep point, including Full, follows this checkpoint-selection and evaluation procedure. Frozen evaluates unchanged pretrained weights on the three N20 realizations. Appendix C gives the aggregation rule; the accompanying code retains all counts.

For $\pi_{0.5}$, the evaluation wrapper executes the returned 32-action chunk before querying the policy again, subject to task completion or the episode limit. Cabinet allows 700 simulator steps, Scan 500, and each of the other eight tasks 400. Inference uses ten Euler flow-integration steps from standard-normal action noise and returns the first fourteen action dimensions.

F Ablation Definitions and Analysis

The component study evaluates six interventions on five tasks randomly selected from the ten-task suite: Bell, Scan, Phone, Alarm, and Bottles. Conditions share experts, K, learner settings, training budget, and validation/test protocol. Agent conditions use `gpt-5.6-sol` at `xhigh`. Each condition has three runs with seed-specific object-position randomization of the twenty test templates; methods share realized scenes and inference seeds within a repetition. RoboCompiler reuses the main runs. The accompanying code retains all success triplets.

Table 17: **Ablations of the compilation procedure.** Success (%, mean ± sample SD), using Table 1's three-run protocol and shared demonstrations, K, and updates. Mean weights tasks equally; bold marks column maxima. Appendix F defines the interventions; the accompanying code contains the run-level counts.

Condition	Bell	Scan	Phone	Alarm	Bottles	Mean
K	32	60	44	32	44	
RoboCompiler	**95.0±5.0**	**58.3±7.6**	**71.7±2.9**	**80.0±5.0**	**73.3±5.8**	**75.7**
Direct agent selection	85.0±5.0	48.3±2.9	61.7±7.6	65.0±5.0	58.3±7.6	63.7
No policy diagnostics	85.0±5.0	46.7±2.9	58.3±2.9	66.7±7.6	60.0±5.0	63.3
No coverage constraints	85.0±5.0	45.0±5.0	58.3±7.6	73.3±2.9	55.0±5.0	63.3
Direct, no diagnostics	80.0±5.0	43.3±2.9	61.7±2.9	65.0±5.0	56.7±5.8	61.3
Uniform within phases	90.0±5.0	53.3±2.9	**71.7±2.9**	75.0±5.0	68.3±5.8	71.7
Duration quotas	83.3±2.9	41.7±2.9	60.0±5.0	66.7±7.6	53.3±7.6	61.0

Table 18: Operational definitions of the six ablations.

Intervention
Direct Agent Selection. Give the agent the same raw task, expert, and policy evidence, model, reasoning effort, and inspection tools. Request K legal indices directly. The output does not require a phase graph, diagnosis cards, or phase quotas.
Without Policy Diagnostics. Remove policy rollouts and all diagnosis-derived annotations. Recompile the phase structure and allocation from the task and the same successful expert demonstrations.
Without Coverage Constraints. Retain the phase structure and diagnostic evidence. Remove phase minima and coverage repair, then reallocate the same total K. A phase may receive zero starts; omission is not forced.
Direct Selection, No Diagnostics. Request direct selection using the task and expert demonstrations alone. Together with RoboCompiler and the two corresponding single interventions, this forms the structured-authoring comparison in Table 20.
Uniform Placement Within Phases. Preserve the full program's phase intervals and integer quotas. Replace local rankings and mandatory corridors with deterministic within-phase uniform placement.
Duration-Proportional Phase Budgets. Preserve the phase partition. Allocate the total budget in proportion to phase duration, using largest-remainder rounding and the same coverage and capacity handling. Apply the uniform within-phase placement rule.

Removing diagnostics recompiles both skill structure and allocation from the same experts. The placement intervention instead preserves phase intervals and quotas, changing only within-phase selection. On Phone, it matches the full program's success count in every repetition and contributes no observed placement gain.

F.1 Construction of the three repetitions

Let P_r be the full RoboCompiler program independently compiled in repetition $r \in \{1, 2, 3\}$. Table 19 shows how each condition obtains its training subset from the corresponding inputs or program. Every condition trains its own learner and uses the shared validation rule and repetition-specific N20 test roster.

Deterministic placement and apportionment. The supplementary reference implementation specifies both transformations. For a phase interval $[a, b)$ of length $d = b - a$ and quota $q \geq 2$, uniform placement selects

$$a + \left\lfloor \frac{j(d-1)}{q-1} \right\rfloor, \qquad j = 0, \ldots, q-1. \tag{12}$$

Table 19: Program construction in repetition r. Each resulting condition has its own learner run. Uniform within phases and duration-based quotas transform the corresponding RoboCompiler program without an additional agent call.

Condition	Starting material	Compilation or transformation
RoboCompiler	Task, experts, diagnostics	Independently compile P_r with the shared prompt.
Direct selection	Task, experts, diagnostics	Run direct agent selection anew; return indices without requiring the structured program.
No policy diagnostics	Task and experts	Recompile phases and allocation with policy diagnostics removed.
No coverage constraints	Phases and diagnosis of P_r	Recompute quotas with the agent after removing phase minima and coverage repair; retain local placement rules.
Direct selection without diagnostics	Task and experts	Run direct selection anew with policy diagnostics removed.
Uniform within phases	Phases and quotas of P_r	Replace local placement with deterministic uniform sampling inside each phase.
Duration-based quotas	Phases of P_r	Replace quotas with duration-proportional allocation, retaining coverage and capacity handling; use the same uniform placement rule.

This includes the first and last legal starts in the phase. A quota of one selects $a + \lfloor (d-1)/2 \rfloor$; a zero quota selects no starts. For duration-based quotas, let d_p and m_p be phase p's capacity and the coverage minimum inherited from P_r. Find $\lambda \geq 0$ such that

$$x_p = \min\{d_p, \max\{m_p, \lambda d_p\}\}, \qquad \sum_p x_p = K. \tag{13}$$

Set $q_p = \lfloor x_p \rfloor$ and give the remaining units to phases in descending order of $x_p - q_p$, breaking ties by temporal order. Phases at capacity receive no further unit. Without active bounds, $x_p = K d_p / \sum_j d_j$. The duration-based condition then applies Eq. 12. The code uses exact fractions for apportionment and rejects inputs that violate $0 \leq m_p \leq d_p$ or $\sum_p m_p \leq K \leq \sum_p d_p$.

F.2 Condition-specific instructions

The following instruction changes apply to the shared prompt in Appendix B. Inspection tools, index validation, and the training protocol remain shared.

Direct selection.
Choose exactly K unique legal starts in every expert demonstration using the task, experts, and policy rollouts. Return the indices and a selection rationale. You may reason about task phases; a phase graph, diagnosis cards, and an explicit quota program are optional.

No policy diagnostics.
Use the task description and successful expert demonstrations to construct phases, coverage, quotas, and placement rules. Policy rollouts, their outcomes, and diagnosis-derived annotations are absent from the input packet.

No coverage constraints.
Keep the supplied phase intervals, diagnostic evidence, and local placement rules. Reassign integer quotas totaling K without phase minima or coverage repair. A phase may receive zero starts. Respect each phase's capacity and the common legal-index checks.

Direct selection without diagnostics.
Apply the direct-selection deliverable using the task and successful experts only. Policy rollouts and their derived annotations are absent, as in the no-policy-diagnostics condition.

F.3 Structured authoring and policy evidence

Direct selection can inspect trajectories and reason about phases; it must deliver indices and a rationale. RoboCompiler additionally requires grounded phases, diagnosis cards, and allocation rules. Table 20 compares the complete authoring procedures with and without policy evidence.

Table 20: Structured authoring and policy evidence. Entries are five-task mean success rates (%). Row differences compare the two authoring procedures under each evidence condition.

Evidence	Direct selection	Structured compilation	Difference
Task and experts	61.3	63.3	+2.0
Task, experts, and policy diagnostics	63.7	75.7	+12.0

Diagnostics improve both authoring procedures, with a larger gain for structured compilation. This interaction is consistent with structure helping organize policy evidence; the comparison includes consequent changes in inspection, reasoning, and allocation. Effects vary by task: without diagnostics, direct selection has a higher mean on Phone.

Coverage removal has its largest effect on Diverse Bottles (18.3 points), followed jointly by Scan and Phone (13.3 each); Bell and Alarm decrease by 10.0 and 6.7 points. The quota comparison fixes phases and uniform placement, varying starts per phase. Restoring local placement increases the Bell, Scan, Alarm, and Bottles means by 5.0 points each and leaves Phone unchanged. The Phone tie concerns test success counts; its local selection rules still change under the intervention. Figure 6 shows these task-dependent effects.

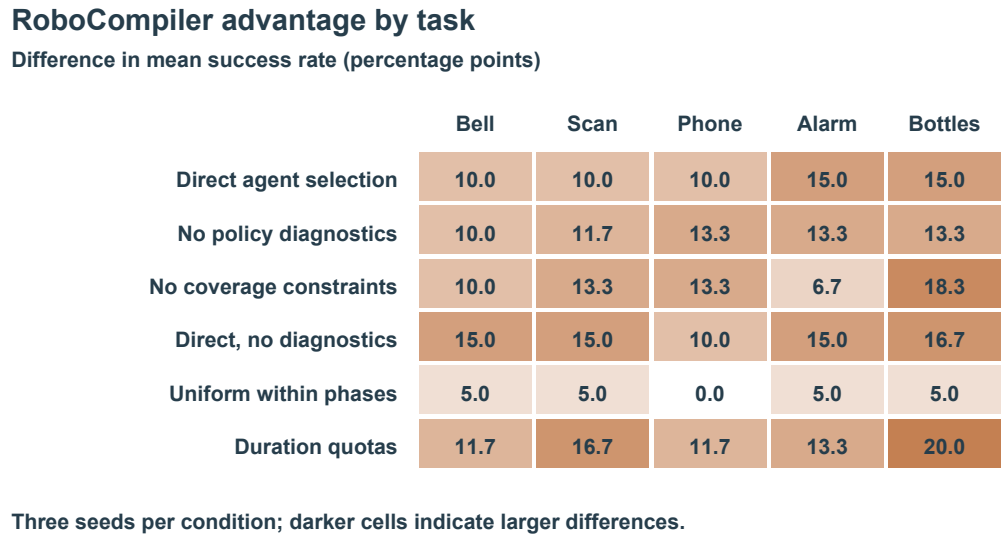

Figure 6: **Task-level effects of the controls.** Each cell subtracts the condition's three-seed mean from RoboCompiler's mean. The full method has a higher observed mean in 29 of the 30 comparisons and ties uniform within-phase placement on Phone. These are differences of reported means.

G COMPILATION USAGE

We measure the complete compilation procedure on Bell K32, Scan K60, and Phone K44. Each task supplies ten successful experts and ten Frozen-policy rollouts from matching scenes. RoboCompiler and direct selection receive identical source packets and inspection tools in separate, fresh contexts. The runtime uses `gpt-5.6-sol`, `xhigh`, and Codex CLI version `0.155.0-alpha.2.6`. Each condition has one measured run per task; the resulting programs are separate from the three-run performance experiments.

Table 21: **Measured compilation usage.** One fresh run per task and condition with the shared prompt and `gpt-5.6-sol`, `xhigh`. Input includes cached tokens; output includes reasoning tokens. Time covers the complete interactive session. These runs measure compilation, separately from the performance experiments.

Task	Condition	Input (M)	Cached (M)	Output (k)	Min.	Responses
Bell K32	RoboCompiler	3.498	3.181	33.9	32.5	31
Bell K32	Direct	1.710	1.615	15.6	15.0	23
Scan K60	RoboCompiler	3.074	2.783	33.1	27.5	31
Scan K60	Direct	2.463	2.331	20.1	21.7	24
Phone K44	RoboCompiler	4.035	3.872	40.4	28.3	36
Phone K44	Direct	2.216	2.106	16.6	19.5	30

Table 22: **Inspection and validation in the same runs.** Tool interactions count top-level calls, which may group several actions. Images count displayed image objects, including contact sheets and repeat views. Detail and numeric requests count calls to the supplied inspection helper.

Task	Condition	Tool interactions	Images	Detail	Numeric	Repairs
Bell	RoboCompiler	30	64	15	3	0
Bell	Direct	22	38	19	31	0
Scan	RoboCompiler	30	46	20	42	0
Scan	Direct	23	58	18	4	0
Phone	RoboCompiler	35	36	20	56	0
Phone	Direct	29	44	16	16	0

Accounting. Table 21 reports session token counters checked against per-response records. Input includes repeated context and cache hits; cached tokens are a subset of input, and reasoning tokens a subset of output. Response counts exclude transport attempts without usage records. Wall time includes tool execution and retries from process launch to completion, excluding prior source preparation and transfer. Source preparation, expert collection, and scene resets are outside this compilation timing. The runtime exposes no per-run monetary charge.

Inspection and completion. After the overview, the agent chooses further inspection windows, writes its diagnosis and selection, and freezes the artifact when all ten episodes pass validation. All six runs finish within the two-hour timeout and pass their first common-validator invocation without repair or human edits. Table 22 records interaction counts. Independent replay reproduces each RoboCompiler output byte for byte and rechecks all source hashes.

RoboCompiler takes 27.5–32.5 minutes versus 15.0–21.7 for direct selection. Its required outputs include phases, diagnosis cards, allocation rules, and replayable code; direct selection returns indices and a rationale. The traces measure the work each procedure actually performs under shared model settings and evidence access. Table 2 compares these complete authoring procedures. Compilation is performed once per run, and its frozen output supplies all subsequent learner updates.

H COMPILED SUPERVISION EXAMPLES

The saved Bell and Scan programs show how allocation changes the training distribution within complete expert demonstrations. Bell requires a short contact sequence within a relatively direct approach; Scan requires several coordinated transitions before the terminal interaction. The analysis follows the saved phase intervals and selected starts across all ten experts, then relates the Bell allocation to recorded policy behavior in matched scenes.

Within a demonstration, a phase containing k of the K selected starts receives sampling mass k/K (Equation 2). A larger quota therefore increases the phase's expected exposure under the shared uniform sampler. Because the total remains K, assigning additional starts to one phase reduces the capacity available elsewhere. Coverage constraints preserve a minimum allocation across the declared skill stages.

The quota intervention in Table 2 changes these budgets while holding the phase partition and uniform within-phase placement rule fixed. The selected observations change with the quotas, but both conditions use the same rule to distribute a phase's starts. This comparison examines allocation across phases. The separate placement intervention preserves the quotas and changes which observations realize them.

Neighboring starts can share future actions while conditioning them on different observations, such as before and after gripper closure. Corridors retain a sequence of such conditions around an interaction. Starts in other phases preserve supervision for approaching and completing it. Consequently, broad future-action coverage alone does not describe the observation distribution seen during training. Appendix E.4 quantifies this overlap for the saved selections.

H.1 BELL: SUPPORTING ENGAGEMENT AND ACTUATION

The Bell policy approaches the target but sometimes fails to produce a verified press. Comparing the failed and successful windows in Figure 7 yields the candidate explanations in Table 23, which map to expert alignment, engagement, and actuation intervals.

Table 23: An evidence-linked Bell diagnosis card. Episode positions index the ten-rollout diagnostic portfolio. The card connects an observed behavior to expert training roles.

Field	Recorded content
Observed behavior	The approach brings the bell into the active wrist view, but the episode does not register a press. Failed episodes 0, 3, and 7 continue to the 400-action limit after the initial approach; the saved diagnosis notes no visibly distinct second attempt.
Failure references	Episodes 0, 3, and 7; frames 24–96.
Successful contrasts	Episodes 1 and 8; frames 16–64. Episode 1 registers success after 64 actions.
Candidate explanations	Contact-normal error, wrist-view occlusion, or insufficient downward displacement.
Expert training roles	Fine alignment, engagement, and state actuation.
Coverage consideration	Retain supervision for direct successful approaches alongside the near-contact intervals.

The agent maps the cited alignment-to-press difficulty to the expert's engagement and actuation phases. The illustrated episode receives quotas $(4, 5, 8, 10, 5)$: eighteen of the K32 starts support these interaction phases, while fourteen support reach, alignment, and recovery. The program therefore expresses a learning priority while retaining observations along the rest of the skill. Table 24 records the phase intervals and exact selected starts for expert episode 0.

Table 24: Literal Bell K32 allocation for the expert episode in Figure 4. Intervals are half-open; ranges in the selected-start column include both endpoints.

Phase	Interval	Quota	Selected starts
Coarse reach	$[0, 21)$	4	3, 5, 8, 11
Fine alignment	$[21, 34)$	5	28, 30–33
Engagement	$[34, 48)$	8	34–38, 40, 44, 45
State actuation	$[48, 67)$	10	48–57
Recovery	$[67, 79)$	5	67–71

Observed interaction states. In this expert episode, the gripper is closed at frame 53 and its end effector descends 21.6 mm by frame 61. These images and robot states ground the actuation interval. The selected corridor 48–57 supplies observations before the later descent, with original action chunks extending through it. The diagnostic policy examples use different scenes and active arms; their observed behavior motivates the allocation. The accompanying code replays these saved selections.

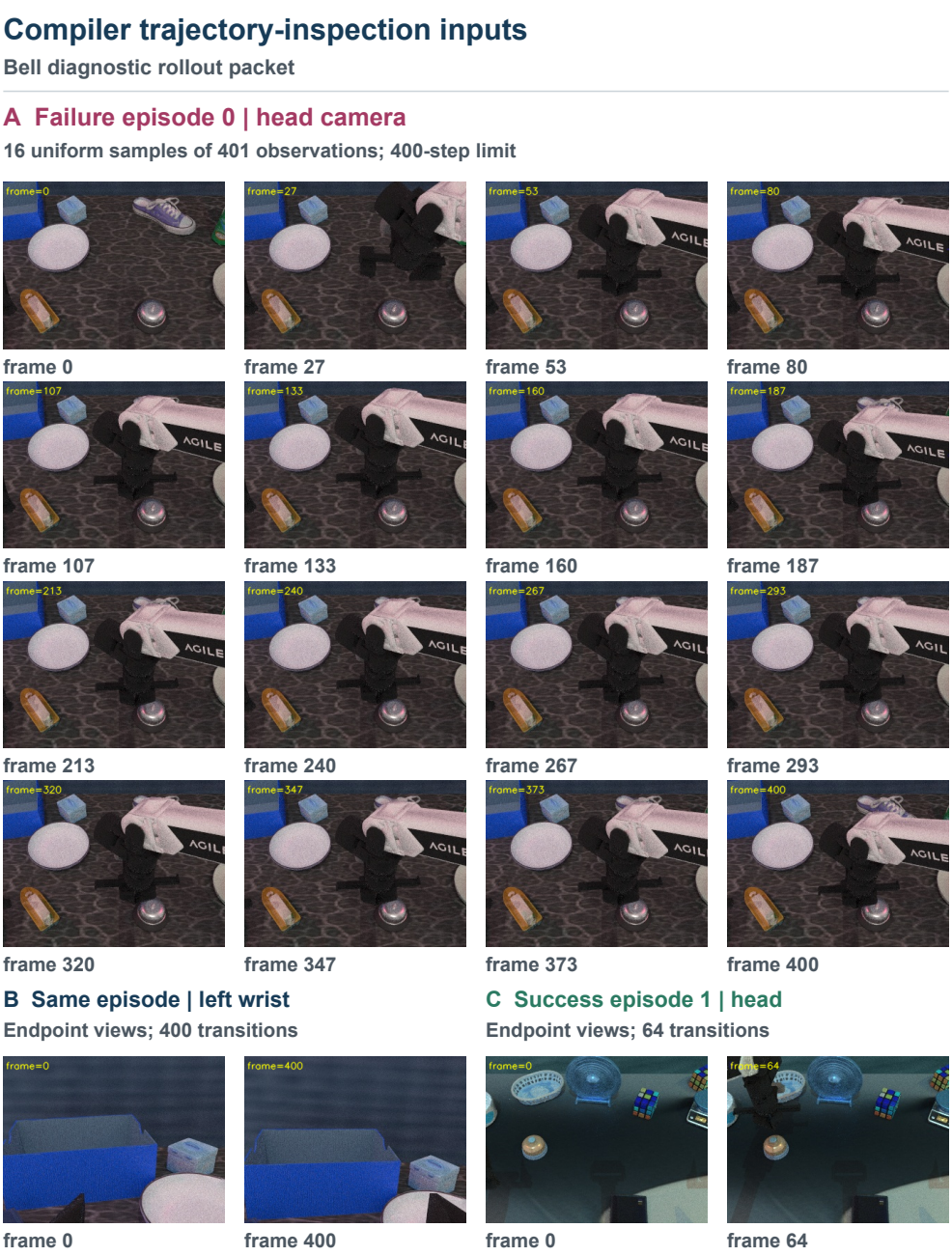

Figure 7: **Bell observations used for diagnosis.** (A) Failed diagnostic episode 0: sixteen uniformly sampled observations from a 400-transition rollout. (B) Endpoint observations from its left wrist camera. (C) Endpoint observations from successful episode 1, which completes in 64 transitions. The two episodes use different scenes and instructions. Frame indices identify observations in the source trajectories.

How the Bell selectors distribute their starts. We assign every selected start to the five intervals identified in the saved RoboCompiler program for the same expert demonstration. These intervals are an analysis reference: Uniform, Motion, HC, and FS do not receive them as input. Table 25 averages the resulting counts over all ten experts. Every compressed selector uses exactly 32 starts per expert.

Full and Uniform: broad temporal coverage. Full trains from every legal observation, so the length of each phase determines its share of starts. Uniform compresses that distribution into 32

Table 25: Bell starting observations by phase, averaged over ten demonstrations. The final column counts demonstrations with at least one start in every phase.

Selector	Reach	Align	Engage	Actuate	Recover	All five
Uniform	9.0	5.4	5.6	7.7	4.3	10/10
Motion	9.9	5.2	7.0	4.9	5.0	10/10
HC	5.5	5.0	5.9	8.9	6.7	10/10
FS	12.8	6.0	12.2	0.0	1.0	0/10
RoboCompiler	4.0	5.0	8.0	10.0	5.0	10/10

regularly spaced starts. It reaches every phase in all ten experts, but gives 9.0 starts on average to coarse reach, versus 5.6 to engagement and 7.7 to actuation. This is a useful complete-trajectory reference: the issue in Bell is not whether the selector ever encounters the press, but how often it trains from the observations just before and during the interaction.

Motion: salient movement is not the same as needed practice. Motion scores changes and local variation in expert joint actions, then suppresses nearby selections. It also covers all five phases, with 7.0 starts in engagement and 4.9 in actuation on average. In the displayed expert it places only five starts in the actuation interval, at 56, 58, 60, 62, and 64; the compiled program places ten consecutive starts at 48–57. Motion records the later movement, while the compiled corridor repeatedly presents the state leading into the press. Motion's 97.63% unique future-action coverage (Table 15) shows that most expert commands remain training targets. The difference lies in which observations condition those commands, not in a wholesale loss of action labels.

Handcrafted: phase counts do not determine local placement. HC uses the arm's geometric path and height to construct broad event windows. It gives actuation 8.9 starts on average, close to RoboCompiler's ten, and covers every phase. In the displayed expert HC spans actuation at 48, 51, 53, 55, 58, 60, 62, 65, and 66, whereas RoboCompiler concentrates at 48–57. The expert's gripper is closed at 53 and the end effector has descended 21.6 mm by 61. The compiled corridor thus includes several closely spaced conditioning states before the later descent; HC also trains on later states in the press and recovery. Similar counts in this interval leave the rest of the selection undetermined. Applying the uniform-within-phase rule to the ten saved allocations yields an average overlap with HC of 14.9 out of 32 starts (range 13–17). This geometric comparison describes the saved portfolio; Table 2 separately tests local placement while holding the compiled phases and quotas fixed.

FrameSkip-style: dense early events can leave a phase without starts. The exact-K FS adaptation ranks observations using action variation, visual–action coherence, a temporal prior, and event preservation. In this Bell portfolio it allocates 12.8 starts to reach and 12.2 to engagement on average, but none to actuation in any expert. In the displayed demonstration, 26 of its 32 starts fall in those two phases; only one is in recovery. Its overlapping action chunks still cover 99.38% of original action indices (Table 15). The absence concerns a training input located within actuation, even though later actions may appear in chunks beginning earlier. This exact-K adaptation puts the selector on the same start budget and learner recipe as the other compressed methods.

RoboCompiler: diagnosed emphasis with complete phase support. The Bell program assigns $(4, 5, 8, 10, 5)$ in every expert: coarse reach receives four starts, while engagement and actuation together receive eighteen. Its local rules then place contiguous starts around alignment, engagement, and the approach to the press. The allocation gives the learner more practice under near-contact observations without abandoning approach or recovery. Compared with Uniform, the program changes both the amount of exposure and its exact visual–proprioceptive context. Compared with HC, the press quota is similar but its local corridor differs. In the formal three-run Bell evaluation, RoboCompiler reaches 95.0% mean success, versus 80.0% for Motion, the strongest fixed selector on this task. The saved indices show the training contrast; the quota and placement controls test its components across tasks.

Matched Bell behavior after training. A separate recorded comparison evaluates Frozen, Full, Motion, and RoboCompiler on the same twenty randomized Bell scenes. The RoboCompiler checkpoint was trained from the saved Bell selection analyzed above. Scene identity, instruction, and realized environment state match across methods. Each trained method here contributes one check-

point at 2,500 updates. This case study is distinct from the formal three-run result, which selects among five checkpoints on validation scenes before testing. Table 26 partitions outcomes according to whether the Frozen policy originally succeeded. It separates learning a new successful behavior on a difficult scene from retaining a behavior the starting policy could already execute.

Table 26: Matched Bell behavior on twenty scenes in the recorded fixed-checkpoint case study. The middle column counts successes among ten scenes failed by Frozen; the right column counts retained successes among ten scenes solved by Frozen.

Method	Success/20	Frozen failure → success	Frozen success retained
Frozen	10	0/10	10/10
Full	15	7/10	8/10
Motion	12	7/10	5/10
RoboCompiler	19	9/10	10/10

The compiled policy succeeds on nine of the ten scenes Frozen fails and retains all ten of its successes. Full also resolves seven initial failures but loses two original successes; Motion resolves seven and loses five. The four scenes where RoboCompiler succeeds and Full fails divide evenly: seeds 62900007 and 62900019 were Frozen failures, while 62900010 and 62900014 were Frozen successes. The common failure on seed 62900001 remains in the analysis. This complete-scene accounting supports both recovery and preservation, rather than attributing every advantage to repair of the initially diagnosed failure.

On seed 62900007, all four policies approach the bell. Frozen hovers with small corrections; Full and Motion each show a downward attempt, but time out after 400 actions. RoboCompiler reaches the interaction and the task checker records success at action 64. The images locate the outcome difference near the bell interaction; they do not reveal contact force or establish which geometric error prevented the other attempts from registering. Seed 62900019 offers a second contrast: Frozen and Full time out, while Motion and RoboCompiler succeed. Accordingly, the paired outcomes and observed trajectories are consistent with the compiled program's greater near-contact practice, without identifying one omitted start or one physical failure mechanism as the unique cause.

H.2 SCAN: A COMPLETE CHAIN WITH SHORT TRANSITION CORRIDORS

Scan requires both objects to be acquired and brought into a usable relative configuration. Successful acquisition must be followed by clearance, stable target placement, scanner prealignment, and a final adjustment of the scanning geometry. These roles explain why the saved K60 program distinguishes seven phases. A long movement can connect two roles; a short, visually subtle adjustment can determine whether the next role is reached. Table 27 summarizes the program across ten expert demonstrations. Figure 8 uses the same phase intervals to inspect every selector's output; these reference intervals are not supplied to the fixed selectors.

Table 27: Scan K60 allocation over ten expert demonstrations. The mean counts sum to the exact per-demonstration budget.

Phase	Mean selected starts
Coarse approach	8.2
Final pregrasp alignment	5.3
Close, acquire, and verify	9.4
Lift and clear	8.7
Position and hold target	14.6
Prealign scanner	10.8
Terminal scan alignment	3.0
Total	60.0

Full and Uniform: coverage follows duration. Full retains every legal start, giving a phase sampling mass proportional to its number of observations. Uniform approximates this temporal distribution with sixty starts per demonstration. In the illustrated portfolio, it selects 12.3 starts during coarse approach and 15.2 during target positioning and holding, but only 1.2 during terminal scanning. It covers all seven phases in every demonstration. Thus, complete phase coverage is already achievable without diagnosis; the remaining decision is how much practice each phase receives. A

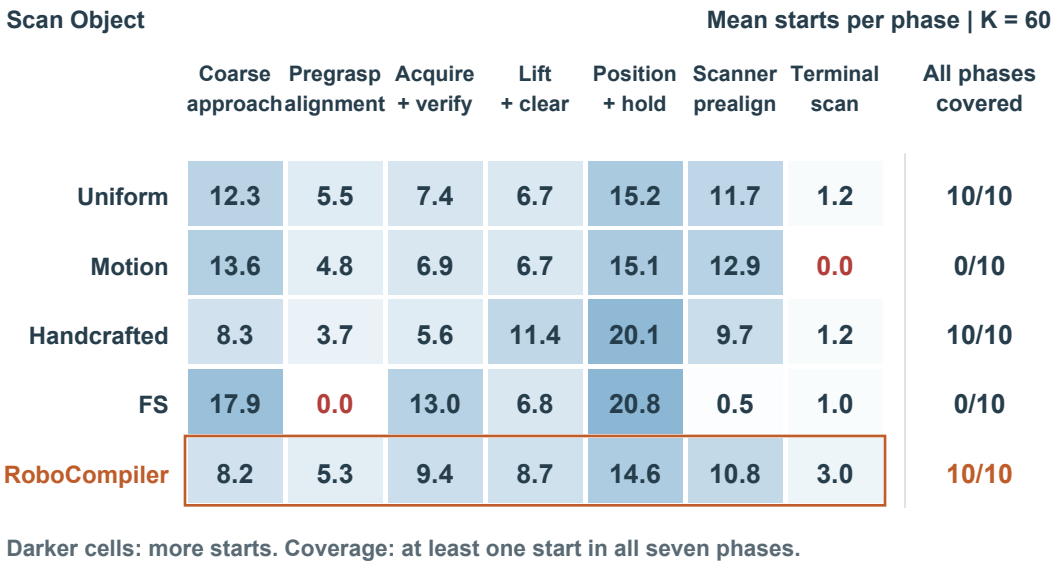

Figure 8: **Where Scan supervision goes.** Each cell gives the mean number of selected starts in one phase over ten expert demonstrations, with K60 fixed for every method. The final column counts episodes with at least one selected start in all seven phases. These values describe the compiled training distribution.

brief terminal correction receives little mass under a duration-based allocation even when reaching the correct relative geometry is difficult for the learner.

Motion: action variation leaves a gap at completion. Motion scores adjacent changes and short-window variance in expert joint actions. Its temporal suppression spreads high-scoring selections apart. This captures active transitions throughout much of Scan: the mean counts for approach, acquisition, lift, and prealignment are 13.6, 6.9, 6.7, and 12.9. Nevertheless, it selects no start in the terminal phase in any of the ten demonstrations. The score measures variation in a successful expert's commands; it does not measure how reliably the learner establishes the final object–scanner configuration. A precise correction can have high task relevance without ranking highly by this score.

The missing starts have a specific meaning. Motion's preceding action chunks still cover 98.08% of the original action indices (Table 15). Some terminal actions therefore enter training as future targets conditioned on earlier observations. What is absent is a training example beginning from an observation inside the terminal phase. Such an example provides the learner with the current relative geometry when predicting its next actions. The distinction concerns observation conditioning and sampling frequency, even when future-action coverage is nearly complete.

Handcrafted: geometric windows span different functional roles. The Scan heuristic partitions the more active arm's cumulative path at its 60% and 90% crossings, subject to boundary constraints, then distributes $15/30/15$ starts across the three windows. It covers every phase in this portfolio, but its windows do not track all seven roles. Target positioning and holding receive 20.1 starts on average, compared with 3.7 for final pregrasp alignment and 5.6 for acquisition and verification. The holding-phase count ranges from 12 to 26 across demonstrations. A fixed path-based window can therefore assign quite different exposure to the same functional phase as the two arms' motions change. The heuristic captures broad geometry; the compiled representation distinguishes the interactions occurring within it.

FrameSkip-style: selected regions can be dense but disconnected. FS combines action variation, visual–action coherence, and a temporal prior with event preservation. Its exact-K ranking imposes no per-phase minimum. Here, approach, acquisition, and target holding receive 17.9, 13.0, and 20.8 starts on average, together accounting for 51.7 of the sixty starts. Final pregrasp alignment receives none in all ten demonstrations; scanner prealignment receives none in nine. Dense

supervision around selected events consequently coexists with gaps in the observations leading into other interactions. The terminal phase contains one start per demonstration. These counts describe the combined selector; attributing a particular gap to one score component would require a separate intervention.

RoboCompiler: allocate practice across roles, then resolve transitions. The compiled program retains all seven phases in every demonstration and gives the terminal phase three starts throughout the portfolio. Relative to Uniform, its mean allocation reduces coarse approach from 12.3 to 8.2, increases acquisition and verification from 7.4 to 9.4, and increases lift and clearance from 6.7 to 8.7. Target holding and scanner prealignment receive slightly fewer starts than under Uniform. The resulting distribution makes selective changes across the chain, including both early acquisition and terminal alignment.

Within these quotas, short corridors cross final pregrasp into acquisition, target stabilization into scanner prealignment, and prealignment into terminal scanning. They retain nearby observations around a change in interaction state. Since the learner samples starts independently, these corridors supply several conditioning states around a transition; they do not introduce sequence-level training. Phase quotas determine exposure to each role, and local placement determines which observations provide that exposure.

What the allocation explains. The selectors make different judgments about relevance: elapsed progress, action variation, geometric events, and combined visual–action scores each capture useful features of successful execution. The agent can relate those features to the learner's observed attempts, distinguish the roles of similar-looking intervals, and express its priorities under a coverage constraint. On Scan, the formal three-run means are 58.3% for RoboCompiler and 48.3% for Motion, the strongest fixed selector. The saved programs make the resulting differences in training exposure concrete. The ablations test coverage, allocation, and placement across tasks; this program analysis explains their training interfaces without assigning a downstream failure to an omitted start alone.

I Additional Supervision Visualizations

The selector panels show successful expert observations, phase boundaries, and literal training starts. Figure 4 shows Bell episode 0 ($L = 79$). The additional examples here are Alarm episode 2 ($L = 88$), Scan episode 4 ($L = 177$), Stamp episode 0 ($L = 143$), QR Code episode 0 ($L = 164$), and Fan episode 0 ($L = 155$), using zero-based portfolio positions. Each panel compares the same expert episode; compressed selectors share the exact budget, with Full as the dense reference.

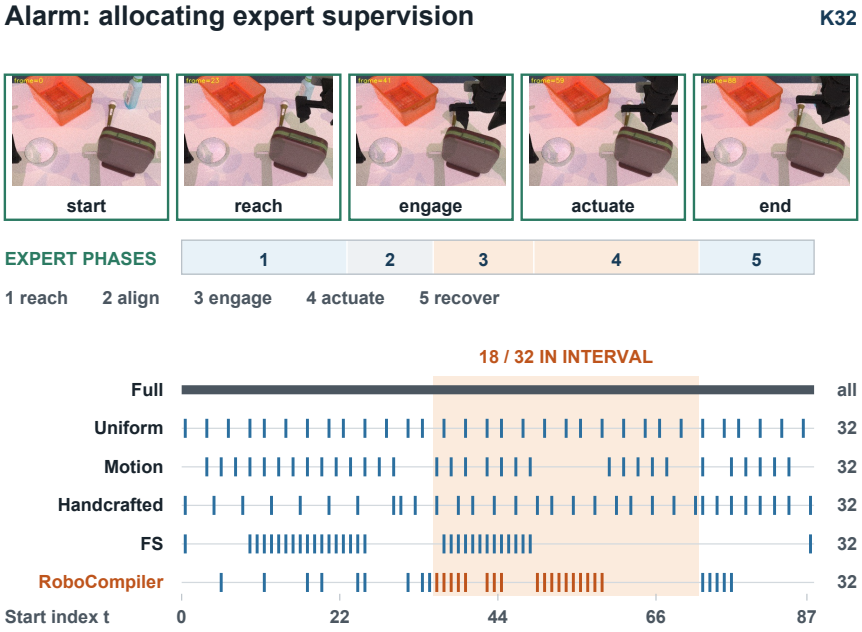

Figure 9: **Alarm K32.** Expert episode 2 has 88 legal starts; the highlighted engagement and actuation interval is $[35, 72)$. The saved program assigns 18 of 32 starts to this interval. All five compressed selectors use the same demonstration and budget.

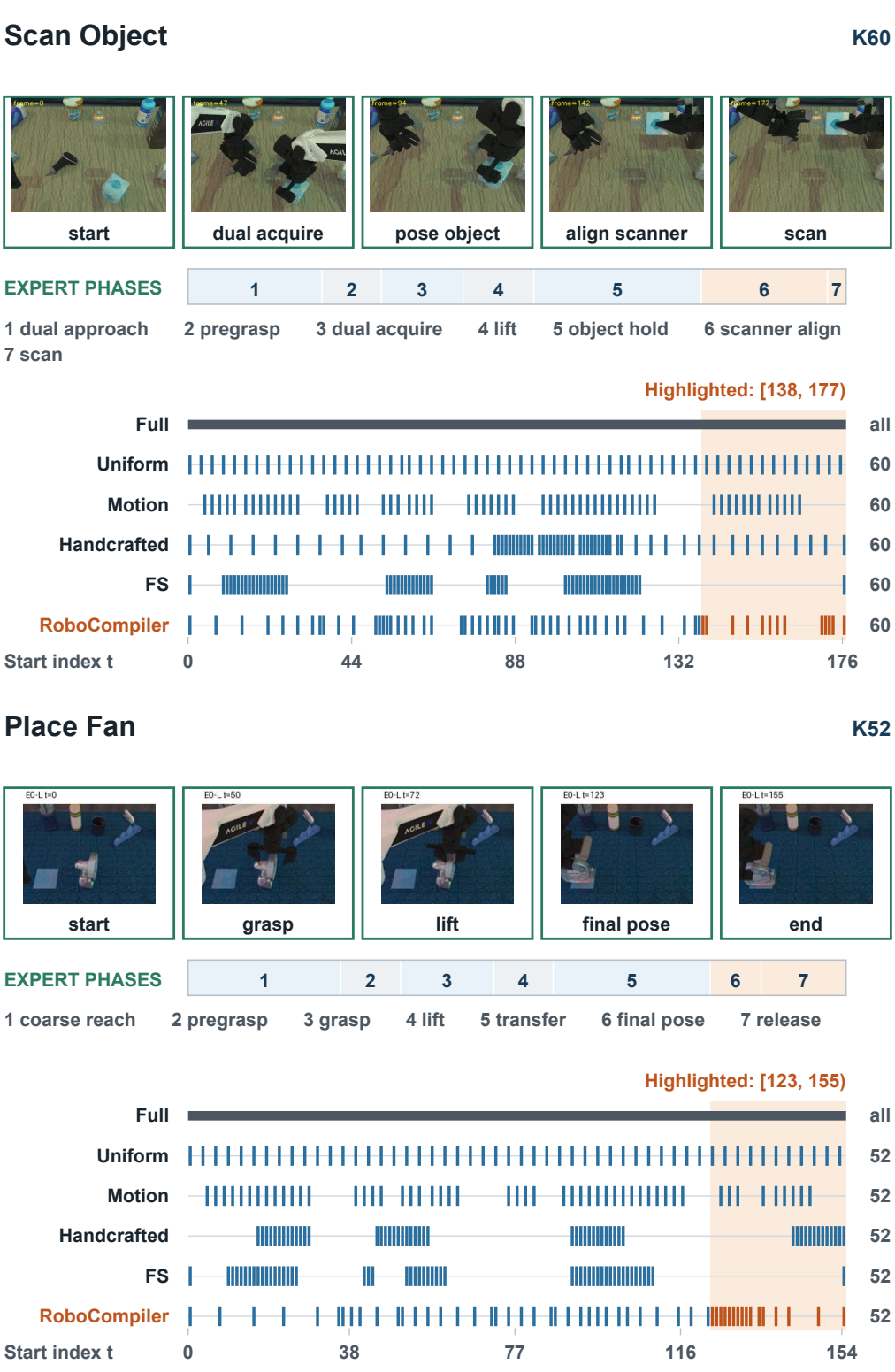

Figure 10: **Scan K60 (top).** The program retains observations through terminal alignment. Figure 8 summarizes phase allocations across all ten expert demonstrations. **Place Fan K52 (bottom).** The selectors distribute the same supervision budget across the successful demonstration. The phase strip locates the selected conditioning observations within the complete skill.

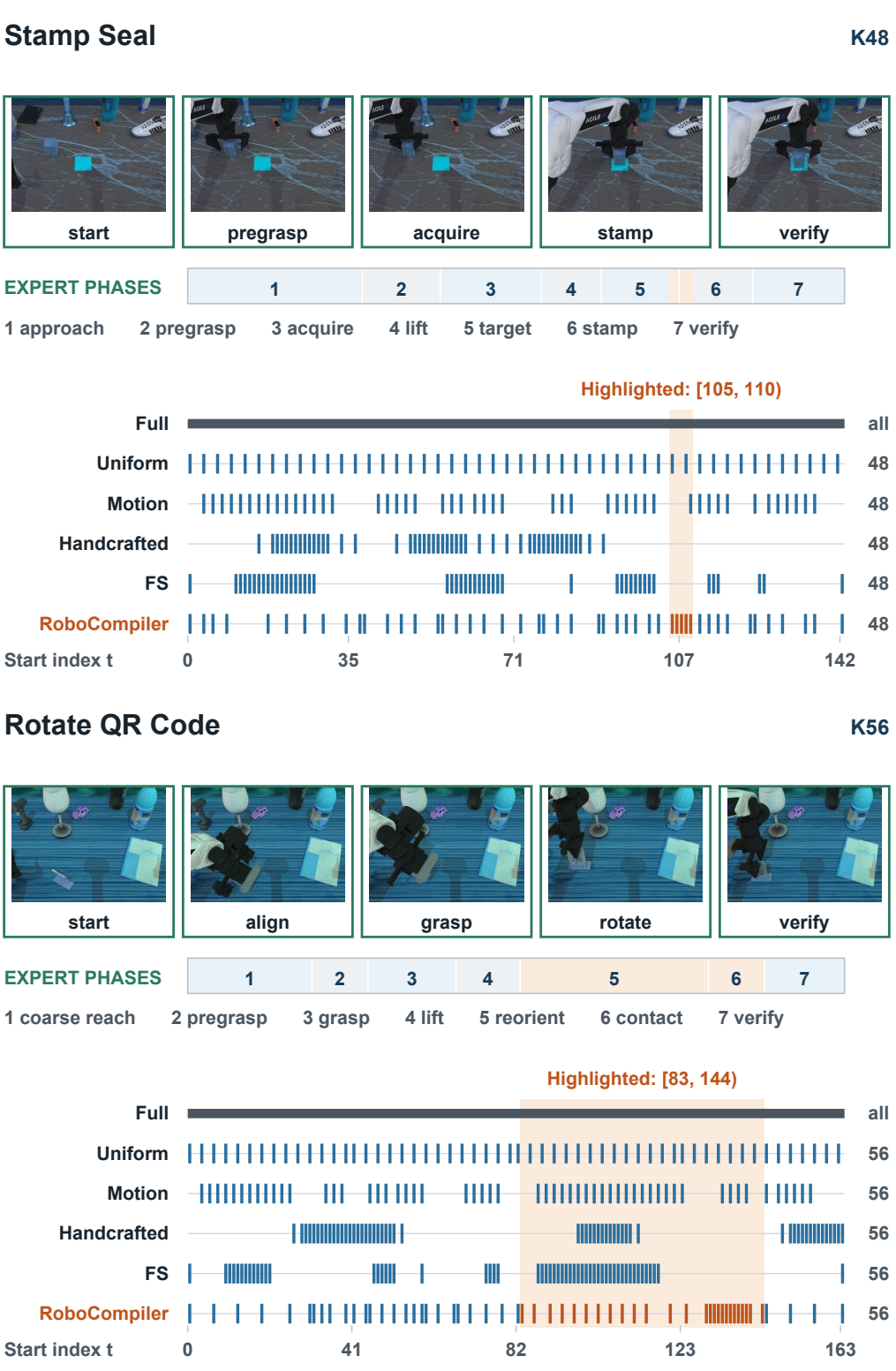

Figure 11: **Stamp Seal K48 (top) and Rotate QR Code K56 (bottom).** Expert observations identify the short interaction stages. Each orange row shows one compiled RoboCompiler program at the same budget as the fixed selectors.

J LIMITATIONS AND BROADER IMPACT

J.1 LIMITATIONS AND DIRECTIONS FOR FURTHER RESEARCH

Scope of the learning setting. The experiments study single-task post-training from ten successful expert demonstrations, using a fixed collection of policy rollouts for diagnosis. The two simulation learners have RoboTwin pretraining, and the physical study covers three ARX tasks. These settings establish that the compilation interface can work across learner recipes and with physical experience. Larger demonstration collections, multi-task training, and longer tasks introduce additional questions about evidence retrieval, shared phase structure, and allocation across skills. Reusing task-level rules while grounding only the relevant parts of new demonstrations could help extend the approach to such collections.

Interpreting the component evidence. The no-diagnostics comparison measures the value of policy experience in the complete procedure: removing it changes both phase grounding and allocation. The quota control holds phases and uniform placement fixed, isolating compiled quotas relative to duration-based quotas. Together, these controls support the use of policy evidence and explicit allocation. They do not fully separate the value of a diagnosis of the particular learner from task-level interaction priors. Likewise, selector panels show where supervision changes; they do not by themselves establish that each diagnosed failure mechanism was repaired. Comparisons across policies with different difficulties offer a direct way to examine how diagnoses, allocations, and subsequent behavior co-vary.

Agent execution and computational cost. The evaluated compiler uses one agent model and reasoning configuration. Direct selection and structured compilation share evidence and inspection tools, but induce different amounts of inspection and reasoning. Their performance comparison therefore concerns the complete authoring procedures. The separate instrumented runs document actual work and saved artifacts for three tasks; they do not estimate the distribution of compilation cost across every performance run. Program checks establish source binding, feasibility, and correct execution of the declared selection. The quality of phase interpretation and diagnosis is assessed through the resulting learning experiments. Studying alternative agent models and more economical inspection policies could clarify which capabilities matter most.

Precision of the reported effects. The simulation results average three independent training repetitions, each evaluating twenty held-out scene templates with separately randomized object positions. Shared realizations within each repetition support matched method comparisons. This design captures variation across training runs and within-template positions, although three observations provide a limited estimate of its distribution. Large aggregate gains and consistent directions across tasks are more informative than small differences in individual cells. The physical results use one training seed and twenty trials per condition, establishing feasibility under the reported settings while leaving across-training variability unmeasured. More repetitions would improve the precision of small task-level and local-placement effects.

Budget and repeated adaptation. The supplied budget controls the number of distinct expert starting observations under a fixed learner-update schedule. It does not measure a proportional reduction in expert collection or training compute: all demonstrations are collected, and retained starts are sampled repeatedly. Our operating budgets retain around one third of legal starts, inspired by the value of selective temporal supervision. The budget sweep leaves room for task-dependent choices. Jointly choosing the budget and allocation from execution evidence, or expressing the program through continuous sampling weights, would extend this formulation. Repeating compilation after further rollouts could also let supervision follow the learner's changing abilities over successive rounds of adaptation.

J.2 BROADER IMPACT ON ROBOT-LEARNING RESEARCH

Robot learning increasingly depends on how experience is organized between rounds of execution and training. The same demonstration can serve different purposes as a policy improves: an approach that initially needs practice may become reliable, while contact or completion remains

difficult. Experience compilation makes this changing relationship an explicit object of study. It connects progress in multimodal agents to a central robot-learning question: how should the next round of supervision respond to what the learner can already do?

The program is a useful research interface because it exposes the decisions between behavioral interpretation and gradient updates. Grounded intervals, evidence references, phase allocations, and local rules can be inspected, replayed, or changed individually. This makes it possible to compare different agents and diagnostic procedures while retaining a common learner, expert portfolio, and training objective. It also allows human knowledge and automated interpretation to be expressed in the same operational form. The expert remains the source of action targets, while the agent helps determine how that experience is used.

This direction can support more systematic policy iteration. Future systems could use compiled records to carry knowledge across collection rounds, identify where additional expert demonstrations would be useful, or coordinate supervision across related tasks. Such extensions would connect data acquisition, interpretation, and training through artifacts that can be evaluated at each stage. RoboCompiler supplies an initial implementation and controlled study of this connection, opening a path toward agents that help robots learn from their own experience.