WHEN IS A SUBTASK COMPLETE? CONSISTENCY-AWARE SUBTASK ANNOTATION FOR HIERARCHICAL ROBOT POLICIES

Anonymous authors
Paper under double-blind review

ABSTRACT

Hierarchical robot policies decompose long-horizon manipulation into a sequence of subtask to conduct a difficult task by high-level policy plans subtask instructions that a low-level policy executes. Training such policies requires decomposing long-horizon demonstrations into subtask segments, each labeled with an instruction and terminal state where the subtask is complete. In prior works, automated decomposition methods cannot ensure that the same subtask ends at the same state across demonstrations; one annotation may end "place the cup" at object release, while another ends it after arm retraction. Without a shared convention, the policy receives conflicting supervision about when to procede to the next subtask. In our pilot study, mixing two individually valid annotation sources lowers hierarchical policy success below that of either source alone (41.4% vs. 49.5% and 50.8%). To address this, we propose Consistency-Aware Subtask Decomposition (CASD), which learns an annotation convention from a small set of reference annotations and applies learned convention across the dataset. CASD over-proposes candidate boundaries from motion cues and scores every interval for the two properties a convention must fix: (1) which skill it shows and (2) whether it covers one complete occurrence of that skill. Because this completion criterion is learned from the references rather than inferred from per-episode visual change, it applies uniformly across demonstrations; a structured loss and dynamic programming then select a consistent labeled partition of each episode. On RoboCasa365 composite tasks, hierarchical policies trained on CASD annotations improve success rate by 10.0%p over the strongest automated annotation baseline and nearly matching simulator ground-truth annotations.

1 INTRODUCTION

Long-horizon manipulation is difficult to learn as a single policy, so hierarchical robot policies separate high-level planning from low-level control (ichter et al., 2023; Huang et al., 2023; Driess et al., 2023; Wang et al., 2023; Li et al., 2025). In recent vision-language-action (VLA) systems, a high-level policy selects the next subtask instruction in language, and a low-level policy executes it (Shi et al., 2025; Belkhale et al., 2024; Black et al., 2025; Lin et al., 2026; Physical Intelligence et al., 2026). Training such policies requires long-horizon demonstrations decomposed into subtask segments, each paired with an instruction. Each segment also fixes a terminal boundary: the state at which the instructed subtask counts as complete and the high-level policy should switch to the next instruction. How demonstrations are decomposed therefore determines both what the high-level policy learns to predict and when it learns to switch.

Because manual decomposition is costly, recent methods automate annotatino process: UVD places boundaries at changes in pretrained visual representations (Zhang et al., 2024), NILS segments demonstrations at keystates described by vision-language models (Blank et al., 2025), and RDD retrieves the reference segments whose endpoints best match each candidate (Yan et al., 2025). These methods produce decompositions that are plausible within each demonstration. However, their boundaries follow generic cues, such as visual change or endpoint similarity, rather than a specified completion convention, so nothing ensures that the same subtask ends at the same state across a dataset. As a result, one demonstration may end "place the cup" at object release while

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.

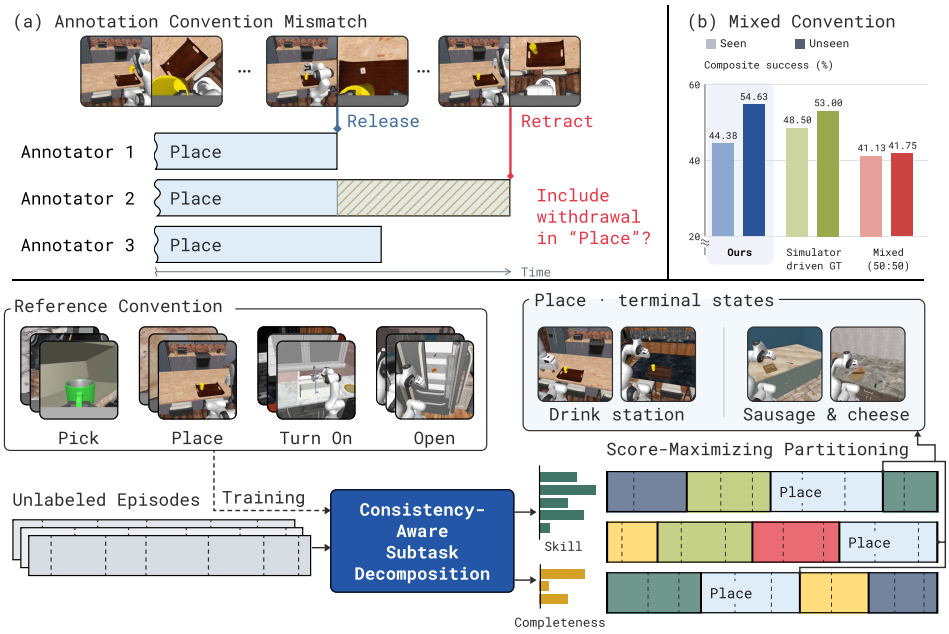

Figure 1: **Overview of CASD** (a) The same instruction can admit different completion boundaries: "place the cup" may end at object release or after arm withdrawal. (b) In our RoboCasa365 pilot, mixing annotation sources reduces policy success relative to either source alone on both composite-seen and composite-unseen tasks. (Bottom) CASD learns skill identity and completeness scores from reference annotations, then selects the highest-scoring segmentation covering the full episode.

Table 1: **Mixing annotation conventions can reduce policy performance.** Mixing CASD and simulator-driven ground-truth annotations 50:50 for the hierarchical policy yields lower success rates than either source alone on RoboCasa365 composite-seen and unseen tasks.

Annotation Source	CS (%)	CU (%)	Average (%)
CASD (Ours)	44.4	**54.6**	49.5
Sim.-driven GT	**48.5**	53.0	**50.8**
Mixed (50:50)	41.1	41.8	41.4

another ends it after arm retraction; even human annotators disagree substantially on such temporal bounds (Moltisanti et al., 2017). This inconsistency matters because the same instruction then comes with different completion states, which gives the policy conflicting supervision about when to switch subtasks. To test whether this affects hierarchical policies, we mix two individually valid annotation sources, simulator-driven ground truth and CASD annotations, at 50:50 on RoboCasa365 (Nasiriany et al., 2026); the mixture lowers average success to 41.4%, below either source alone (50.8% and 49.5%). Annotations that are valid within each episodes are therefore not sufficient; subtask supervision needs a shared convention across the dataset.

To address this consistency issue, we propose Consistency-Aware Subtask Decomposition (CASD), which learns one annotation convention from a small set of human-annotated reference demonstrations and applies it across the dataset. Unlike prior methods, CASD does not rely on generic cues to decide where a subtask ends; it learns the completion criterion from the references and applies the same criterion to every demonstration. Specifically, CASD over-proposes candidate boundaries from kinematic cues, so that intervals terminate at different plausible states are all available. It then scores each candidate interval for two properties: (1) which skill is performed (identity) and (2) whether it covers one complete occurrence of that skill (completeness). Completeness must be scored separately because an interval that stops before release or extends past retraction can still show the correct skill. A structured loss trains these scores on whole reference segmentation, and dynamic programming (Bellman, 1957) selects the highest-scoring labeled partition of each episode.

To validate CASD, we train hierarchical policies on its annotations in simulation and on real robots under identical policy training settings. On the RoboCasa365 composite suites, CASD annotations

improve average success by 10.0 percentage points over the strongest automated annotation baseline, nearly matching simulator-state ground truth (Table 2); on two long-horizon ALOHA tasks, they also yield the highest success. These gains coincide with more consistent completion boundaries: relative to shared grasp and release events, CASD annotations end each skill at markedly more consistent times than UVD and RDD (Section 5). Validated in four robot datasets, CASD consistently agrees with reference labels and boundaries. Our analysis further shows why completeness must be modeled separately from recognition. Removing the completeness component leaves skill recognition nearly intact but sharply reduces boundary accuracy (Section 5); recognizing which skill an interval shows is therefore not sufficient to decide where it ends.

Our main contributions are as follows:

- We show that mixing two individually valid annotation conventions lowers hierarchical policy success below either source alone.
- We propose CASD, which learns a single annotation convention from a small reference set by scoring skill identity and completeness and decoding a full labeled partition.
- We show that CASD annotations end skills at more consistent times across demonstrations and improve policy success in simulation and on real robots, and we evaluate annotation quality across multiple robot datasets.

2 Related Work

Subtask Supervision for Hierarchical Policies Subtask-level supervision provides training signals for language- and goal-conditioned policies (Blank et al., 2025; Zhang et al., 2024), as well as hierarchical policies for instruction following and language-guided recovery (Shi et al., 2025; Dai et al., 2025). In these systems, intermediate language commands connect task-level intent to executable actions (Belkhale et al., 2024), and the high-level policy learns when to issue the next command from where the annotations end each subtask. The annotations must therefore fix both what each subtask is and when it is complete. Our work addresses this annotation stage: a few human references specify the convention, and a learned scorer with dynamic programming applies it to the whole dataset.

Dense Robot Demonstration Annotation Recent methods automate robot demonstration annotation through visual subgoal discovery (Zhang et al., 2024; Liu et al., 2024), object-centric language labeling (Blank et al., 2025), and retrieval-based decomposition (Yan et al., 2025). UVD places boundaries where pretrained visual embeddings change, but these changes cannot be steered toward a chosen convention, and camera motion alone can trigger them. RDD retrieves reference segments whose endpoint embeddings and duration match each candidate; it can retrieve segments that do not match the demonstrated behavior, including segments from unrelated episodes under distribution shift, and because it compares only segment endpoints, it ignores the actions within a segment. CASD instead learns skill identity and completeness from a small reference set, using visual-language features together with robot states and actions, which makes agreement with the convention an explicit training objective.

3 Method

In this section, we present CASD. Prior annotators end each subtask wherever a generic cue indicates, so the same skill can end at different states across demonstrations. Applying one convention instead requires that the convention is specified, that intervals ending at the intended state are available, and that the choice among them follows the convention rather than per-episode cues. CASD meets these requirements with reference annotations that specify the convention (Section 3.1), overproposed candidate intervals scored for skill identity and completeness (Section 3.2), and a scorer trained and decoded over complete partitions of each episode (Sections 3.3 and 3.4; Figure 2).

3.1 Problem Formulation and Reference Annotations

We consider a demonstration $\tau = ((\boldsymbol{o}_t, \boldsymbol{s}_t, \boldsymbol{a}_t))_{t=0}^{T-1}$ paired with a task instruction ℓ, where $\boldsymbol{o}_t$, $\boldsymbol{s}_t$, and $\boldsymbol{a}_t$ denote observations, robot state, and action. A segment $I = [a, b)$ contains frames $a, \ldots, b-$

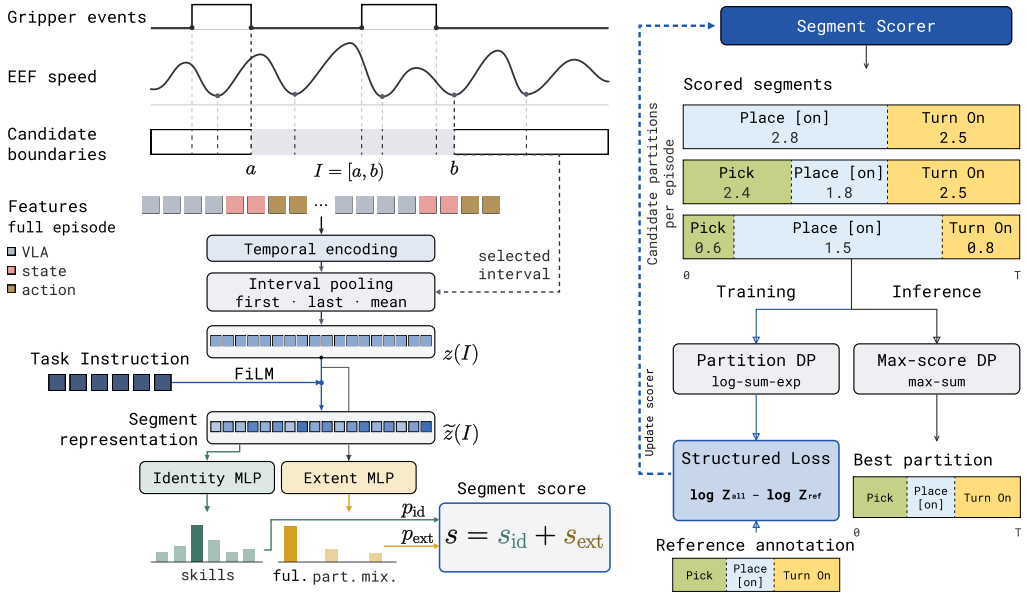

Figure 2: **Learning segment scores through complete episode partitions. Left:** Local minima in kinematic cues provide candidate boundaries. Temporally encoded VLA features, robot states, and actions are pooled within each candidate interval to predict skill identity and completenes. **Right:** During training, a structured loss couples segment scores through complete episode partitions, favoring paths consistent with reference skill sequences and temporal boundaries.

1, with integer boundaries $0 \leq a < b \leq T$. A reference annotation $P^{\text{ref}} = ((I_m^{\text{ref}}, k_m^{\text{ref}}))_{m=1}^{M}$ partitions the demonstration into M skill occurrences, where $I_m^{\text{ref}} = [b_{m-1}^{\text{ref}}, b_m^{\text{ref}})$, $0 = b_0^{\text{ref}} < \cdots < b_M^{\text{ref}} = T$, and k_m^{ref} belongs to the fixed skill vocabulary $\mathcal{K}$. Given a support set $\mathcal{D}_{\text{ref}}$ of annotated triples $(\tau, \ell, P^{\text{ref}})$, we learn to predict a labeled partition $\hat{P}$ for a new demonstration and instruction.

Reference selection and annotation Reference annotations specify the convention; because they define it for the whole dataset, they should cover the variation within it. For composite tasks, given an annotation budget B, we therefore select representative demonstrations using k-medoids clustering (Kaufman & Rousseeuw, 1987) with dynamic time warping distance (Sakoe & Chiba, 1978) between end-effector trajectories. Annotators then mark boundaries and assign skill labels from $\mathcal{K}$ to form $\mathcal{D}_{\text{ref}}$; Appendix B describes the skill templates and the annotation procedure.

3.2 Candidate Generation and Segment Scoring

This subsection addresses the second and third requirements: over-proposed candidates make intervals ending at different plausible states available, and identity and completeness scores choose among them according to the references.

Candidate segments. We over-propose boundaries at local minima in end-effector speed and gripper open–close transitions, using the same detection procedure across demonstrations (Appendix F.1). The detector only proposes changes in physical motion; the scorer decides which resulting intervals match the reference skill and completion definitions. Let $\mathcal{C} \subseteq \{0, \ldots, T\}$ denote these positions, including 0 and T. We restrict candidate intervals $I = [a, b)$ to pairs in $\mathcal{E} = \{(a, b) \in \mathcal{C} \times \mathcal{C} : b - a \geq d_{\min}\}$, where $d_{\min} > 0$ is the minimum duration in frames.

Segment representations. To represent both the action and its temporal extent, we combine features from the beginning, end, and interior of each candidate segment. For each frame, we extract a task-conditioned feature $\boldsymbol{v}_t = \phi_{\text{VLA}}(\boldsymbol{o}_t, \ell)$ from a frozen VLA backbone and concatenate it with

robot state and action, $\boldsymbol{x}_t = [\boldsymbol{v}_t; \boldsymbol{s}_t; \boldsymbol{a}_t]$. After fixed input normalization, a temporal encoder maps these inputs to frame features $\boldsymbol{h}_t \in \mathbb{R}^{d_h}$, where d_h is the hidden dimension. We parameterize the encoder with a frame-wise MLP followed by residual temporal convolutions. For a segment $I = [a, b)$, we concatenate its first, last, and mean frame features: $\boldsymbol{z}(I) = [\boldsymbol{h}_a; \boldsymbol{h}_{b-1}; \frac{1}{b-a}\sum_{t=a}^{b-1} \boldsymbol{h}_t] \in \mathbb{R}^{3d_h}$. We use the same encoder and pooling for candidate segments and reference-derived training intervals. Appendix A.2 gives the architecture details.

Identity and extent. A convention must fix both which skill an interval shows and where that skill ends. Identity is not given by correct boundaries: even with human-defined intervals, a proprietary vision-language model (Gemini) labels skills with only 68.5% macro-F1 (Section 5), partly because similar motions can serve different skills, as when reaching toward a burner precedes picking up a pot or turning on the burner. Completeness is not given by identity either, because intervals that stop before release or extend past retraction still show the same skill. We therefore score each segment for both skill identity and completeness. To condition skill identification on the task instruction, we apply FiLM (Perez et al., 2018) to the pooled segment representation $\boldsymbol{z}(I)$ using features from a frozen CLIP text encoder (Radford et al., 2021). A learned generator maps these features to channel-wise scale offsets and shifts, producing $\tilde{\boldsymbol{z}}(I)$. An MLP followed by a softmax predicts the skill probabilities $p_{\mathrm{id}}(k \mid I)$ for $k \in \mathcal{K}$ from this modulated representation. To evaluate completeness, a separate extent head predicts whether the segment contains a full skill occurrence, part of one, or multiple skills. It applies an MLP and softmax to the unmodulated $\boldsymbol{z}(I)$ to obtain $p_{\mathrm{ext}}(q \mid I)$ for $q \in \mathcal{Q} = \{\mathrm{full}, \mathrm{partial}, \mathrm{mixed}\}$. With trainable parameters $\boldsymbol{\theta}$, the two predictions give the score

$$s_{\boldsymbol{\theta}}(I, k) = \log p_{\mathrm{id}}(k \mid I) + \log \frac{p_{\mathrm{ext}}(\mathrm{full} \mid I)}{p_{\mathrm{ext}}(\mathrm{partial} \mid I) + p_{\mathrm{ext}}(\mathrm{mixed} \mid I)}. \tag{1}$$

The first term scores skill identity; the second scores completeness as full-versus-nonfull log odds.

3.3 Training the Segment Scorer

Scoring each interval separately does not guarantee that an episode is segmented consistently with the references. We therefore train the scorer at two levels: intervals and complete partitions.

Local supervision. To teach the scorer what a complete occurrence looks like, we supervise skill identity and temporal completeness using intervals derived from the reference annotations. Reference intervals provide full examples, while their prefixes and suffixes provide partial examples; both retain the reference skill label and supervise both heads. Unions of adjacent intervals with different skill labels provide mixed examples, which supervise only the extent head. We additionally augment identity supervision with small inward boundary perturbations, retaining the original skill labels to encourage stable predictions under boundary changes. These augmented examples receive no extent targets. Full examples retain human boundaries even when these are absent from the candidate boundary set $\mathcal{C}$. We minimize identity cross-entropy $\mathcal{L}_{\mathrm{id}}$ and extent cross-entropy $\mathcal{L}_{\mathrm{ext}}$, with extent classes weighted by inverse frequency in the training set.

Structured supervision. At the partition level, we add a structured objective that increases the probability of reference-compatible partitions. Let $\mathcal{P}$ denote the set of labeled partitions that cover the episode without gaps or overlaps. Each path $P \in \mathcal{P}$ consists of candidates (I, k), where $I = [a, b)$, $(a, b) \in \mathcal{E}$, and $k \in \mathcal{K}$. We score a path by summing its segment scores: $S_{\boldsymbol{\theta}}(P) = \sum_{(I,k)\in P} s_{\boldsymbol{\theta}}(I, k)$. Because proposed boundaries may differ from reference boundaries, we allow multiple reference-compatible paths. The subset $\mathcal{P}_{\mathrm{ref}}$ contains paths with the same segment count M and ordered skill labels as P^{ref}, and temporal IoU of at least 0.5 between corresponding intervals. For episodes with at least one compatible path, we minimize

$$\mathcal{L}_{\mathrm{struct}} = -\frac{1}{M} \log \frac{Z_{\mathrm{ref}}}{Z}, \qquad Z = \sum_{P\in\mathcal{P}} e^{S_{\boldsymbol{\theta}}(P)}, \quad Z_{\mathrm{ref}} = \sum_{P\in\mathcal{P}_{\mathrm{ref}}} e^{S_{\boldsymbol{\theta}}(P)}. \tag{2}$$

Here, Z_{ref}/Z is the probability mass assigned to reference-compatible paths under the normalized path scores. The factor $1/M$ normalizes the loss by the number of reference segments. Minimizing the loss increases this mass, coupling segment scores through complete paths while allowing alternative boundary placements consistent with the reference. Log-sum-exp dynamic programming computes both partition functions (Sarawagi & Cohen, 2004). If $\mathcal{P}_{\mathrm{ref}} = \emptyset$, we set $\mathcal{L}_{\mathrm{struct}} = 0$.

Table 2: **Downstream policy performance with different subtask annotations on RoboCasa365.** GR00T N1.6 uses a low-level policy without subtask supervision; all other rows use hierarchical policies. Each suite contains 16 tasks with 50 rollouts per task; seen/unseen denotes task exposure during pretraining. Success is the task success rate, and Progress measures normalized cumulative subtask completion. Avg. is the success rate averaged over both suites. †Reference using simulator-state ground-truth annotations. Per-task results are given in Table 11.

		Avg.	Composite Seen		Composite Unseen	
Subtask annotation	Hierarchical	Success (%)	Success (%)	Progress	Success (%)	Progress
GR00T N1.6		41.2	43.4	63.0	39.0	62.0
UVD (Zhang et al., 2024)	✓	39.5	41.8	63.4	37.3	60.5
RDD (Yan et al., 2025)	✓	36.9	34.4	56.4	39.5	62.4
CASD (Ours)	✓	**49.5**	**44.4**	**64.7**	**54.6**	**75.9**
Simulator-state GT†	✓	50.8	48.5	67.9	53.0	75.7

Joint objective. For each demonstration, we define the joint objective as

$$\mathcal{L} := \mathcal{L}_{\text{id}} + \mathcal{L}_{\text{ext}} + \mathcal{L}_{\text{struct}}. \tag{3}$$

We average this objective over demonstrations in each minibatch. We jointly optimize the temporal encoder, FiLM generator, and both prediction heads, while keeping the VLA and text encoders frozen. The batch average includes episodes without compatible paths, which contribute only their local losses. Local losses and pseudocode are given in Appendices D and E.

3.4 Global Decoding and Language Grounding

To apply the learned convention consistently within each episode, we select the labeled partition with the highest total score instead of labeling candidate intervals independently. Specifically, at inference we compute $\hat{P} = \arg\max_{P \in \mathcal{P}} S_{\boldsymbol{\theta}}(P)$ by max-score dynamic programming. Let $V(b)$ be the highest score of a labeled partition covering $[0, b)$. Starting from $V(0) = 0$, we process candidate boundaries $b \in \mathcal{C} \setminus \{0\}$ in increasing order:

$$V(b) = \max_{\substack{(a,b) \in \mathcal{E} \\ k \in \mathcal{K}}} \big[V(a) + s_{\boldsymbol{\theta}}([a, b), k) \big]. \tag{4}$$

Backtracking from T recovers $\hat{P}$, jointly selecting skill labels and boundaries while allowing proposed boundaries to be skipped. Decoding requires neither reference annotations nor a prescribed skill sequence or segment count. Skill labels fix the convention but not the objects and locations involved. To produce subtask instructions, a vision–language model receives the full-episode videos, the task instruction, and each decoded interval's skill template and argument roles, and outputs one grounded instruction per interval (Appendix F.2).

4 Experiments

Our experiments address three questions. (1) do CASD annotations improve hierarchical policy learning in simulation (Section 4.1) and on real robots (Section 4.2), and (2) how closely do they recover reference skill labels and temporal boundaries (Section 4.3)? Section 5 then examines (3) whether the annotations apply one completion convention and what determines their temporal extents.

4.1 Simulation Experiments

Experimental setup. We first evaluate the annotations as supervision for hierarchical policies on RoboCasa365 (Nasiriany et al., 2026). We train CASD on 640 human-labeled demonstrations spanning 300 `pretrain` tasks, then annotate the `target` demonstrations used for policy learning. Following RoboCasa365's policy learning setup, we use Qwen3-VL-4B-Instruct (Bai et al., 2025) as the high-level policy, predicting subtask instructions from observations and the full-task instruction,

Table 4: **Annotation quality (%).** Each dataset uses its own training and evaluation splits. B-F1 uses a 0.5 s tolerance; J-F1 requires matching skill labels and temporal IoU of at least 0.75. Edit measures skill-sequence similarity; higher is better for all metrics.

	RC365 ID (`pretrain`)			RC365 OOD (`target`)			Allex			ALOHA			AgiBotWorld-Alpha		
Method	B-F1	J-F1	Edit	B-F1	J-F1	Edit	B-F1	J-F1	Edit	B-F1	J-F1	Edit	B-F1	J-F1	Edit
UVD	10.1	15.1	64.4	12.7	13.8	68.4	21.5	9.3	90.6	29.6	14.1	65.8	7.4	10.8	96.4
RDD	15.8	14.7	62.7	20.6	19.0	62.6	56.0	24.9	87.2	**86.9**	**60.4**	98.7	59.0	72.8	92.5
CASD (Ours)	**33.3**	**66.7**	**93.3**	**46.9**	**56.6**	**81.6**	**87.9**	**89.7**	**99.0**	82.9	49.1	**99.7**	**65.3**	**90.8**	**100.0**

and GR00T N1.6 (NVIDIA GEAR Team, 2025) as the low-level policy. We pretrain the low-level policy for 80K steps on `pretrain`, then fine-tune both policy levels on `target` separately for the composite-seen and composite-unseen suites: 60K steps for the low-level policy and 6K for the high-level policy. See Appendices A.1 and C.1 for training and evaluation details.

Compared methods. We compare with UVD (Zhang et al., 2024) and RDD (Yan et al., 2025). For UVD and RDD, we follow the RDD pipeline to assign subtask instructions to predicted intervals; RDD uses the same human-labeled references as CASD. Each annotation pipeline supervises both policy levels. We also report two references: a low-level-only GR00T N1.6 and a hierarchical policy trained with simulator-driven ground-truth annotations.

Downstream policy performance. Policies trained with CASD annotations achieve higher success and progress than the evaluated automated annotation baselines on both suites (Table 2). Averaged over both suites, success improves by 10.0 percentage points over the strongest automated baseline; the gain is larger on composite-unseen tasks (15.1 points over RDD) than on composite-seen tasks (2.6 points over UVD). Relative to simulator-state annotations, CASD yields lower success on seen tasks and higher success on unseen tasks. These gains align with the problem raised in Section 1: UVD and RDD end the same skill at more dispersed times across demonstrations (Section 5), so their policies receive less consistent supervision about when to switch subtasks.

4.2 Real-World Policy Evaluation

We next evaluate whether the annotations also support long-horizon policy learning in real-world manipulation. We consider Table Setup and Table Bussing on ALOHA (Zhao et al., 2023), each comprising more than ten subtasks. For each task, we collect approximately 100 teleoperated demonstrations with object positions randomized within a fixed distribution.

Policies trained with CASD annotations achieve the highest success rates and progress scores on both tasks (Table 3). The success-rate advantage is larger on Table Setup: 43.3%, compared with 30.0% for UVD and 20.0% for GR00T N1.6. On Table Bussing, our annotation achieves highest progress score: 3.1, compared with that of baselines. Together with the simulation results, these comparisons support using CASD annotations for long-horizon policy learning. See Appendix C.2 for the evaluation protocol.

Table 3: **Real-world ALOHA results.** SR: success rate; Prog.: progress score.

	Table Setup		Table Bussing	
Method	SR (%)	Prog.	SR (%)	Prog.
GR00T N1.6	20.0	2.9	13.3	2.3
UVD	30.0	3.1	6.7	2.7
CASD	**43.3**	**3.7**	**16.7**	**3.1**

4.3 Annotation Quality

Evaluation setup. We separately evaluate whether the generated annotations recover each dataset's reference skill labels and boundaries. We consider RoboCasa365 (Nasiriany et al., 2026), Allex, ALOHA (Zhao et al., 2023), and AgiBotWorld-Alpha (Bu et al., 2025), training a separate annotator per dataset and comparing with UVD and RDD on the same evaluation episodes. Appendix C.3 details budgets, baseline configurations, and splits.

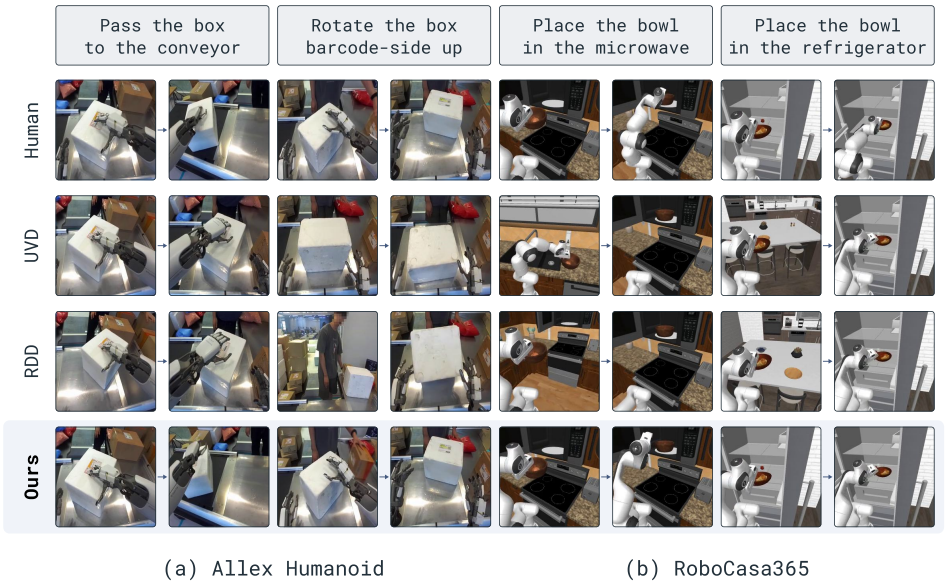

Figure 3: **Qualitative annotation examples.** Start and end frames of subtask segments on (a) Allex and (b) RoboCasa365, comparing human annotations with UVD, RDD, and CASD. Each column corresponds to the subtask instruction shown above.

Evaluation metrics. We measure boundary localization with Boundary F1 (B-F1), joint skill-and-interval agreement with Joint F1 (J-F1), and skill-sequence similarity with Edit. For each evaluation episode e, let $\mathcal{B}_e$ and $\mathcal{S}_e$ denote the reference internal boundary set and labeled segment set, respectively; hats denote predictions. For B-F1, we find the maximum number of one-to-one internal boundary matches within 0.5 s. For J-F1, a predicted segment $(\hat{I}, \hat{k})$ can match a reference (I, k) only if $\hat{k} = k$ and their $\mathrm{tIoU}(\hat{I}, I) \geq 0.75$. Each segment contributes to at most one match. Let $M_{B,e}$ and $M_{J,e}$ count the matched boundaries and labeled segments in episode e. We pool these counts over the evaluation episodes of each dataset. Edit measures skill order independently of boundary timing. After collapsing consecutive repeated labels, we compute 100 times one minus the Levenshtein distance divided by the longer sequence length, then average over episodes.

$$\text{B-F1} = \frac{2\sum_e M_{B,e}}{\sum_e(|\widehat{\mathcal{B}}_e| + |\mathcal{B}_e|)}, \qquad \text{J-F1} = \frac{2\sum_e M_{J,e}}{\sum_e(|\widehat{\mathcal{S}}_e| + |\mathcal{S}_e|)}. \tag{5}$$

Annotation accuracy. Table 4 shows higher joint F1 for CASD than for UVD and RDD on both RoboCasa365 splits, Allex, and AgiBotWorld-Alpha. On RoboCasa365 `pretrain`, joint F1 is 66.7, compared with 15.1 for the strongest baseline. ALOHA is an exception: RDD achieves higher boundary and joint F1, while CASD achieves higher Edit. Skill-sequence agreement and temporal localization therefore capture different aspects of annotation quality. Figure 3 shows examples.

5 Analysis

This section analyzes the two properties behind our central claim: whether CASD annotations apply one completion convention across demonstrations (RQ1), and whether completeness scoring determines their temporal extents (RQ2). Appendices H and I report further analyses.

RQ1: Do CASD annotations apply one completion convention across demonstrations, compared with UVD and RDD? Joint F1 measures agreement with reference boundaries, but not whether an annotator ends the same skill at the same point across demonstrations. We therefore annotate the same 48 RoboCasa365 `target` demonstrations with every source, so only the annotator differs, and measure when each ends a skill relative to a physical event shared by all sources:

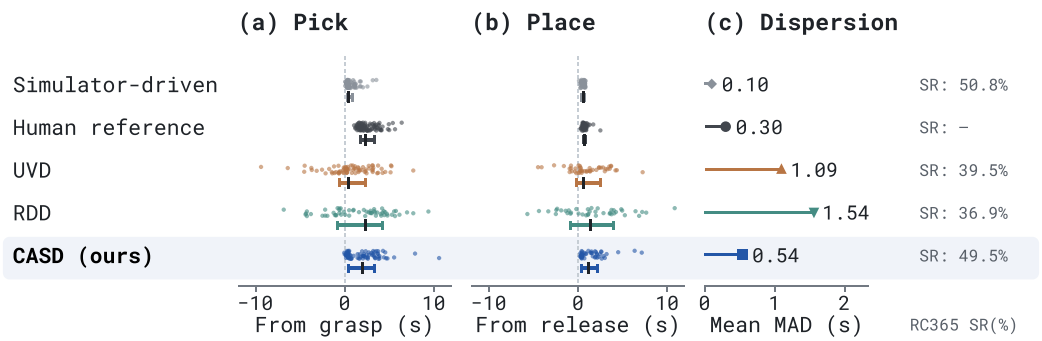

Figure 4: **Endpoint timing and dispersion.** (a,b) PICK and non-terminal PLACE endpoints relative to shared grasp/release commands (zero is the command, not a target boundary). Circles show all 619 matched endpoints, jittered vertically; n is each row's matched count; markers and lines show medians and interquartile ranges. (c) Mean MAD over 25 task $\times$ skill groups.

the grasp command for PICK and the release command for PLACE (138 PICK and non-terminal PLACE occurrences in 25 task $\times$ skill groups; Appendix G). If an annotator applies one convention, its endpoints for a given skill should fall at a similar offset from this event. We take the median absolute deviation (MAD) of the offsets within each group and average across groups. CASD has a mean MAD of 0.54 s, against 1.09 s for UVD and 1.54 s for RDD (Figure 4), while the human references (0.30 s) and simulator-state annotations (0.10 s) remain lower. The ordering persists on the 83 occurrences matched by all sources (0.54, 1.07, and 1.58 s). This result describes endpoint timing on this population; it does not show that lower dispersion causes the policy gains.

RQ2: Does completeness scoring, rather than skill recognition, determine where annotated skills end? Since CASD agrees closely with the references (Table 4), we ask whether recognition or completeness scoring drives this agreement. We remove one component at a time, keeping data, features, and decoding fixed, and evaluate on the same 80 in-domain episodes before language grounding (Table 5). Even frontier VLM even fail given ground truth segment as input. label macro-F1 (L-F1) score 68.5%, while ours identity head reach 90.4%, justifying our design choice of predicint skill identity together with boundary. Removing the extent head is critical impact in B-F1, its supervision, and its score term leaves L-F1 at 90.3%, yet boundary F1 falls from 33.3% to 8.8% and joint F1 from 65.4% to 17.3%. This 48.1-point joint F1 drop far exceeds those from removing the temporal convolutions (11.1 points), the structured loss (6.6), or identity FiLM (4.9), which leaves boundary F1 unchanged (33.5%) and mainly lowers L-F1. Recognizing a skill is therefore not sufficient to decide where it ends; completeness scoring determines the extent.

Table 5: **Component ablation** B-F1; L-F1: label macro-F1; J-F1 tIoU@0.75.

Variant	B-F1	L-F1	J-F1
Full	33.3	**90.4**	**65.4**
w/o FiLM	**33.5**	84.9	60.5
w/o Temporal	27.1	79.6	54.3
w/o Struct. loss	33.3	86.8	58.8
w/o Extent	8.8	90.3	17.3

6 Conclusion

In our RoboCasa365 pilot, mixing two individually valid annotation sources reduced policy success, suggesting that episode-level validity does not ensure effective supervision. We introduced CASD, which learns skill identity and temporal completeness from human-annotated references and uses global decoding to annotate full demonstrations. CASD improves agreement with reference annotations on several robot datasets, ends each skill at more consistent times across demonstrations than UVD and RDD, and yields policy gains over the evaluated baselines in simulation and on real robots. The method relies on a fixed skill vocabulary and references that specify the intended convention. Our results suggest applying one completion convention across demonstrations when supervising hierarchical policies.

AI USE STATEMENT

In this work, we used generative AI tools for draft paper outline, revise and edit grammer in manuscript, annotate data in VLM grounding stage. We have not used generative AI to draw the figure. All AI-generated code are tested under unit test and regression test. All AI-generated code is human-reviewed by authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All robot data were collected on our own robots. The policies were executed on the real robots under human supervision with joint and velocity limits in place. Human referenece dataset annotation doesn't collect any personal information or confidential information. All released training data is blurred with video contains human's face.

REPRODUCIBILITY STATEMENT

Full model code will released in github page. https://iclrcasd-coder.github.io/casd-site. We release model code at submission. Training data will be released through anonymous github page in the furture. Using this model architecture and training configuration in Appendix, other researchers are able to test our method in public dataset.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, et al. Qwen3-VL technical report. *arXiv preprint arXiv:2511.21631*, 2025.

Suneel Belkhale, Tianli Ding, Ted Xiao, et al. RT-H: Action hierarchies using language. In *Proceedings of Robotics: Science and Systems*, 2024.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

Kevin Black, Noah Brown, James Darpinian, et al. $\pi_{0.5}$: a vision-language-action model with open-world generalization. In *Conference on Robot Learning*, 2025.

Nils Blank, Moritz Reuss, Marcel Rühle, et al. Scaling robot policy learning via zero-shot labeling with foundation models. In *Conference on Robot Learning*, 2025.

Qingwen Bu, Jisong Cai, Li Chen, et al. Agibot world colosseo: A large-scale manipulation platform for scalable and intelligent embodied systems. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2025.

Yinpei Dai, Jayjun Lee, Nima Fazeli, and Joyce Chai. RACER: Rich language-guided failure recovery policies for imitation learning. In *IEEE International Conference on Robotics and Automation*, 2025.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, et al. PaLM-e: An embodied multimodal language model. In *International Conference on Machine Learning*, 2023.

Wenlong Huang, Fei Xia, Ted Xiao, et al. Inner monologue: Embodied reasoning through planning with language models. In *Conference on Robot Learning*, 2023.

brian ichter, Anthony Brohan, Yevgen Chebotar, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, 2023.

Leonard Kaufman and Peter J. Rousseeuw. Clustering by means of medoids. In *Statistical Data Analysis Based on the L1-Norm and Related Methods*. North-Holland, 1987.

Yi Li, Yuquan Deng, Jesse Zhang, et al. Hamster: Hierarchical action models for open-world robot manipulation. In *International Conference on Learning Representations*, 2025.

Fanqi Lin, Ruiqian Nai, Yingdong Hu, et al. Onetwovla: A unified vision-language-action model with adaptive reasoning. In *International Conference on Learning Representations*, 2026.

Ruizhe Liu, Qian Luo, and Yanchao Yang. InfoCon: Concept discovery with generative and discriminative informativeness. In *International Conference on Learning Representations*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Davide Moltisanti, Michael Wray, Walterio Mayol-Cuevas, and Dima Damen. Trespassing the boundaries: Labeling temporal bounds for object interactions in egocentric video. In *IEEE International Conference on Computer Vision*, 2017.

Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations*, 2026.

NVIDIA GEAR Team. GR00T N1.6: An improved open foundation model for generalist humanoid robots. NVIDIA Research Blog, December 2025. URL https://research.nvidia.com/labs/gear/gr00t-n1_6/.

Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron Courville. FiLM: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*, 2018.

Physical Intelligence et al. $\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities, 2026. URL https://arxiv.org/abs/2604.15483.

Alec Radford, Jong Wook Kim, Chris Hallacy, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.

Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26:43–49, 1978.

Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *Advances in Neural Information Processing Systems*, 2004.

Lucy Xiaoyang Shi, Brian Ichter, et al. Hi Robot: Open-ended instruction following with hierarchical vision-language-action models. In *International Conference on Machine Learning*, 2025.

Chen Wang, Linxi Fan, Jiankai Sun, et al. Mimicplay: Long-horizon imitation learning by watching human play. In *Conference on Robot Learning*, 2023.

Mingxuan Yan, Yuping Wang, Zechun Liu, and Jiachen Li. Rdd: Retrieval-based demonstration decomposer for planner alignment in long-horizon tasks. In *Advances in Neural Information Processing Systems*, 2025.

Zichen Zhang, Yunshuang Li, Osbert Bastani, et al. Universal visual decomposer: Long-horizon manipulation made easy. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6973–6980. IEEE, 2024.

Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. In *Proceedings of Robotics: Science and Systems*, 2023.

A Model Initialization and Hyperparameters

A.1 Training the hierarchical policy

We fine-tune the language model of Qwen3-VL-4B-Instruct with the hyperparameters in Table 6, keeping its vision encoder and multimodal projector frozen. The policy predicts the current subtask instruction from camera observations and the full-task instruction. Figure 5 summarizes the information flow between the two policy levels.

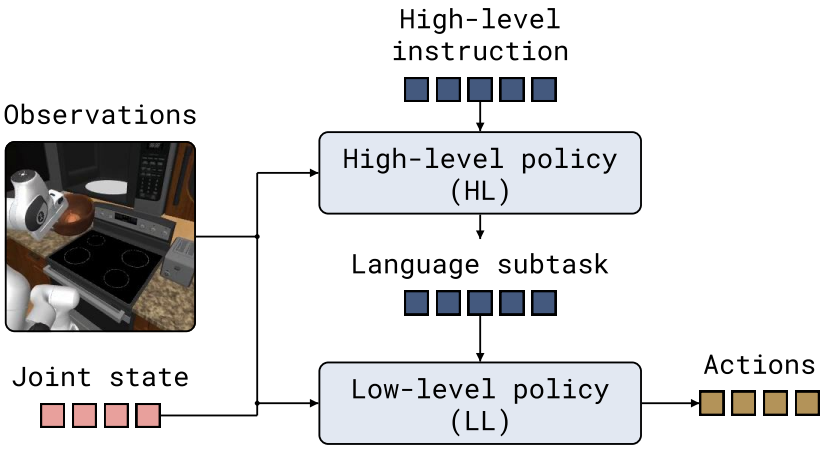

Figure 5: **Hierarchical policy used for downstream evaluation.** The high-level policy maps the current observations and full-task instruction to a language subtask. The low-level policy conditions on the observations, robot joint state, and active subtask to produce actions.

Table 6: High-level policy hyperparameter. Dataset-specific values are listed as RoboCasa365 / ALOHA.

Hyperparameter	Value
Optimizer	AdamW
(β_1, β_2)	$(0.9, 0.999)$
ϵ	10^{-8}
Learning rate	6×10^{-5}
Weight decay	0
Gradient clip norm	1.0
LR schedule	Cosine
Warmup ratio	0.05
Effective batch size	128
Training steps	6,000 / 1,000
Camera view	Agent left, right, wrist / High only
View composition	Horizontal concatenation / Single view
Gradient checkpointing	Enabled

We train a separate high-level policy for each ALOHA task and retain the pretrained model's image processor.

Table 7 lists the low-level policy hyperparameters. We initialize from GR00T N1.6 and fine-tune the projector and action diffusion model, keeping the vision and language backbone frozen. On RoboCasa365, we pretrain with full-task instructions on `pretrain`, then fine-tune separate composite-seen and composite-unseen policies with subtask instructions on `target`. Each ALOHA task policy is initialized directly from GR00T N1.6.

Table 7: Low-level policy training configuration. Stage-specific values are listed as RoboCasa365 pretraining / RoboCasa365 fine-tuning / ALOHA. Color jitter tuples denote (brightness, contrast, saturation, hue).

Hyperparameter	Value
Optimizer	AdamW
(β_1, β_2)	$(0.9, 0.999)$
ϵ	10^{-8}
Learning rate	3×10^{-5} / 10^{-4} / 10^{-4}
Weight decay	10^{-5}
Gradient clip norm	1.0
LR schedule	Cosine
Warmup ratio	0.05
Effective batch size	128 / 128 / 64
Training steps	80,000 / 60,000 / 40,000
Gradient accumulation	1
Gradient checkpointing	Disabled
Action chunk length	16 / 16 / 50
Camera views (count)	3 / 3 / 4
Robot state input	Enabled
ALOHA action frequency	25 Hz
Color jitter	$(0.3, 0.4, 0.5, 0.08)$ / $(0.3, 0.4, 0.5, 0.08)$ / $(0.5, 0.6, 0.5, 0.08)$
State dropout	0.0 / 0.0 / 0.7

A.2 Training the CASD scorer

The hyperparameters for training the CASD scorer are listed in Table 8. We train the temporal encoder, FiLM generator, skill-identity head, and extent head jointly from scratch, using cached features from frozen GR00T N1.6 and CLIP ViT-B/32 encoders.

Table 8: Shared training configuration for the CASD scorers.

Hyperparameter	Value
Optimizer	AdamW
(β_1, β_2)	$(0.9, 0.999)$
ϵ	10^{-8}
Learning rate	3×10^{-4}
Weight decay	10^{-4}
Gradient clip norm	1.0
LR schedule / warmup	Constant / none
Batch size	8 demonstrations
Training epochs	30
Hidden dimension	128
Dropout	0.1
Identity / extent / structured loss weights	1 : 1 : 1
Random seed	0
Precision / GPUs	FP32 / 1
Checkpoint used	Final epoch
Validation-based selection	Disabled

The frame encoder consists of LayerNorm and a two-layer MLP with GELU, followed by three residual temporal convolutions with kernel size 3 and dilations $(4, 8, 16)$. We concatenate the first, last, and mean frame features to obtain a 384-dimensional segment representation. The FiLM generator maps the 512-dimensional CLIP feature to 768 scale-and-shift parameters and has a zero-initialized final layer. Both prediction heads use two-layer MLPs with the same hidden dimension as the frame encoder; FiLM modulates only the identity-head input.

Table 9 lists the reference budgets and output vocabularies for each dataset. Local targets and the structured objective are described in Appendices D and E, and proposal settings in Appendix F.

Table 9: Scorer training runs. The completion extension adds `TASK_COMPLETE` to the RoboCasa365 vocabulary. ALOHA annotation-quality evaluation uses a three-class scorer; policy-label generation uses a separate instruction-level vocabulary.

Dataset / use	References	Classes	Updates
RoboCasa365: annotation quality	640	24	2,400
RoboCasa365: completion extension	640	25	2,400
Allex	64	4	240
ALOHA: annotation quality	32	3	120
AgiBotWorld-Alpha	152	3	570
ALOHA: policy-label generation	32	18	120

B Data Annotation Procedure

B.1 Reference demonstration selection

For RoboCasa365 composite `pretrain` tasks, we select references per task by k-medoids clustering of robot trajectories under a dynamic time warping (DTW) distance and choose the number of clusters with an elbow criterion, as described below. Dataset-specific annotation budgets are listed in Table 12.

Trajectory representation. Each episode is represented by an 11-dimensional trajectory-shape sequence computed from the 16-dimensional proprioceptive state and resampled from 20 Hz to 5 Hz: base position and heading expressed relative to the initial base pose, the base-relative end-effector position and orientation (a unit quaternion) as recorded in the state, and gripper aperture normalized to $[0, 1]$. Expressing the base pose relative to its initial value compares how the robot moves rather than where it starts.

Distance. The cost between two frames is the equally weighted sum of four terms: the root-sum-square of the scaled base displacement and the heading change divided by π, the scaled end-effector position error, the end-effector rotation angle divided by π, and the gripper aperture difference. Each position error is divided by a single scale per group, the Euclidean norm of the per-coordinate 5–95% quantile ranges over all frames of the 235 composite `pretrain` tasks, so that all terms lie on a comparable scale. The distance between two episodes is the DTW alignment cost under this frame cost, computed without a warping window and divided by the length of the optimal alignment path, which makes episodes of different durations comparable. We compute this distance for every pair of episodes within a task.

Clustering and elbow criterion. Within each task with N episodes, we isolate outlying episodes whose mean distance to their nearest neighbors exceeds a robust threshold, at most $\lfloor 0.01N \rfloor$ episodes, and cluster the remaining episodes with k-medoids for every $K \in \{1, \ldots, 8\}$. Starting from the greedy PAM BUILD initialization, each run alternates between assigning episodes to their nearest medoid, or, when that leaves a cluster below the minimum size of $\lceil 0.05N \rceil$ episodes, to the minimum-cost assignment satisfying that bound, and replacing each medoid by the member with the smallest total distance to its cluster, until the medoids stop changing; $K = 1$ selects the episode with the smallest total distance to all others. Let J_K denote the sum of distances from the non-outlying episodes to their assigned medoids. We select the smallest K for which adding one more cluster reduces this objective by less than 10%, that is, the first K with $(J_K - J_{K+1})/J_K < 0.1$. The K medoids are the reference demonstrations of the task. Under this rule, 64 of the 235 composite `pretrain` tasks receive one reference, 137 two, 29 three, and 5 four or five, for 446 composite references in total.

B.2 SKILL VOCABULARY

For RoboCasa365, we use the fixed 24-class skill vocabulary in Table 10. Each unqualified skill forms one label, while every listed skill–qualifier pair forms a separate label.

Table 10: RoboCasa365 reference skill vocabulary. Each qualifier defines a separate label; a dash denotes an unqualified skill. The completion extension adds `TASK_COMPLETE`.

Skill	Qualifiers
PICK	—
PLACE	IN, ON, UNDER, RELATIVE, STACK
NAVIGATE	—
ARTICULATE	OPEN, CLOSED, EXTENDED, RETRACTED
SET_CONTROL	ON, OFF, SET
AGITATE	ROTATE, TILT
POUR	IN, ON
HOLD	—
PUSH	—
PRESS	—
SCRUB	—
STIR	—
WAIT	—

B.3 ANNOTATION INTERFACE AND REVIEW PROCEDURE

The browser interface displays the episode instruction, a synchronized three-view video, and a frame-indexed segmentation timeline. Annotators step through the video, insert or adjust boundaries, and choose a template for each interval. Boundaries can be placed at any frame and are not restricted to the proposal set $\mathcal{C}$. The campaign does not display episode-specific simulator annotations or a prescribed subtask sequence. Two annotators contributed to the exported reference set; each saved episode is associated with its annotator and source recording.

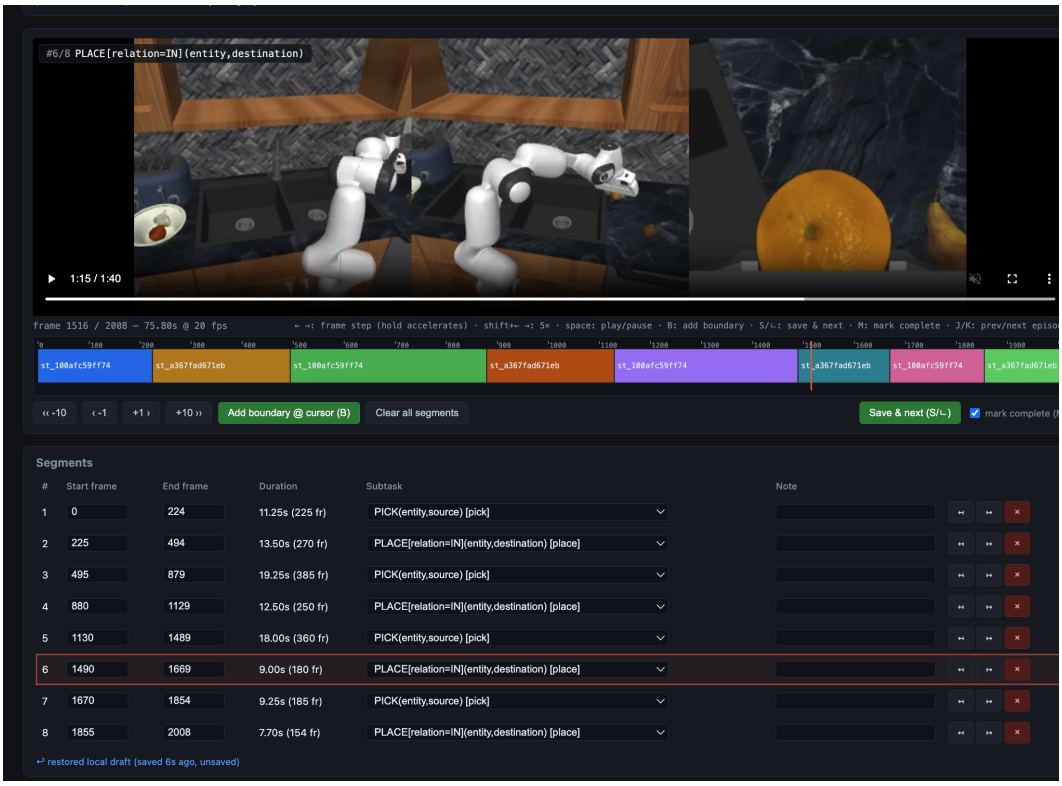

Figure 6: **Human annotation tool interface.** The synchronized three-view video, frame-indexed segmentation timeline, and editable interval list support frame-level boundary placement and skill-template selection.

Before accepting a completed annotation, the server verifies the episode identity, frame rate, frame count, and template identifiers. It requires nonempty, ordered intervals covering the full recording without gaps or overlaps, and permits at most one terminal interval, placed last. Annotators can revise the intervals and labels before saving. These checks enforce structural validity; skill identity and completion timing are assigned from the video according to the annotation conventions.

The interface stores inclusive frame endpoints; export increments each stored end frame by one to obtain the half-open intervals used throughout the paper. Placement relations are canonicalized using the task's placement template and a fixed destination mapping, with the episode instruction resolving unspecified destinations. Explicit `UNDER`, `RELATIVE`, and `STACK` relations are preserved; unresolved cases retain their selected relation. This label normalization does not move the annotated boundaries.

For the 24-class scorer, terminal tails are absorbed into the preceding skill, subject to a maximum of 120 frames (6 s). One collected episode exceeds this limit and is excluded, leaving 640 references. The completion extension uses the same episodes but retains their annotated terminal intervals as `TASK_COMPLETE`, as specified in Table 9. Human reference annotation is separate from the later language-grounding stage, which fills object and location arguments for automatically decoded intervals (Appendix F.2).

C Experiment Evaluation Details

C.1 Simulation policy evaluation

We evaluate on the composite-seen and composite-unseen suites of RoboCasa365, using the v1.0.1 simulator task registry. Each suite contains 16 tasks; seen and unseen refer to task exposure during policy pretraining. Both suites use the `target` object split and ten paired kitchen layouts and styles. For each method, we run 50 episodes per task, yielding 800 rollouts per suite. Five parallel environments, seeded from 0 to 4, each contribute ten episodes under the same seed schedule across methods. Episodes end upon task success or the task-specific registry horizon.

For hierarchical policies, the high-level policy receives the current camera observations and full-task instruction and predicts a subtask instruction every 0.8 s of simulated time. We use synchronous inference and switch instructions after two consecutive identical predictions. The low-level policy receives the active subtask instruction and executes 16 actions per prediction at 20 Hz. All annotation-based hierarchical policies share these execution settings. The low-level-only GR00T N1.6 reference instead receives the full-task instruction and executes the first eight actions of each predicted 16-action chunk. Training configurations are given in Appendix A.1.

Success is determined by RoboCasa365's official task-completion check. To measure partial completion, we additionally report normalized cumulative progress from task-specific simulator-state milestones, such as grasping or placing a target object. Each milestone earns one point on its first achievement, and official task success earns one additional point. Credit is retained if a milestone is later undone, with no extra credit for repeated achievement. We divide the earned points by the total available points for that episode, excluding inapplicable milestones such as opening an already-open fixture. This diagnostic does not alter task success or provide input to the policy. In Table 2, both metrics are expressed as percentages, averaged over episodes within each task and then equally over the 16 tasks. Average success gives equal weight to the two suites (Table 11).

Table 11: Per-task policy evaluation on RoboCasa365. Success rate (SR) and normalized cumulative progress (Prog.) are percentages over 50 rollouts per task, using the protocol in Appendix C.1. Suite means reproduce Table 2; the overall mean weights all 32 tasks equally. GR00T N1.6 is the low-level-only reference, and Sim.-state GT uses simulator-state ground-truth annotations. Bold marks the best value for each metric in each row, including ties.

	GR00T N1.6		UVD		RDD		**CASD (Ours)**		Sim.-state GT	
Task	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.	SR	Prog.
Composite-seen										
Deliver Straw	2.0	15.5	**16.0**	28.5	14.0	30.0	8.0	**30.5**	6.0	25.5
Get Toasted Bread	2.0	28.0	**24.0**	**44.0**	14.0	31.5	12.0	33.5	**24.0**	42.0
Kettle Boiling	62.0	65.5	34.0	39.5	36.0	41.5	56.0	62.5	**66.0**	**70.0**
Load Dishwasher	**74.0**	**90.7**	56.0	83.3	54.0	77.7	**74.0**	83.3	58.0	84.7
Pack Identical Lunches	8.0	30.4	**10.0**	40.4	4.0	36.8	6.0	38.0	8.0	**41.6**
Pre Soak Pan	52.0	79.0	**68.0**	83.7	42.0	71.0	64.0	**84.3**	66.0	83.3
Prepare Coffee	32.0	53.5	22.0	52.0	14.0	41.0	**34.0**	**58.5**	24.0	52.5
Rinse Sink Basin	70.0	82.8	60.0	74.0	50.0	72.8	68.0	76.8	**74.0**	**88.0**
Scrub Cutting Board	56.0	75.2	50.0	**78.0**	36.0	56.0	56.0	72.8	**62.0**	72.8
Searing Meat	**38.0**	**62.0**	20.0	49.2	14.0	48.4	22.0	52.8	18.0	54.4
Set Up Cutting Station	36.0	50.4	28.0	53.6	38.0	56.4	40.0	57.6	**54.0**	**70.8**
Stack Bowls Cabinet	64.0	81.7	62.0	78.7	66.0	76.3	80.0	85.0	**88.0**	**94.7**
Steam In Microwave	**64.0**	**82.3**	54.0	78.6	40.0	64.0	48.0	78.6	52.0	74.9
Stir Vegetables	18.0	60.0	26.0	67.0	16.0	57.5	28.0	67.5	**36.0**	**69.8**
Store Leftovers In Bowl	58.0	87.4	68.0	92.0	60.0	82.6	52.0	86.6	**76.0**	**93.1**
Wash Lettuce	58.0	63.2	**70.0**	**72.4**	52.0	58.4	62.0	66.8	64.0	68.8
Composite-seen mean	43.4	63.0	41.8	63.4	34.4	56.4	44.4	64.7	**48.5**	**67.9**
Composite-unseen										
Arrange Bread Basket	64.0	75.7	66.0	72.5	68.0	78.8	**80.0**	**89.4**	70.0	81.1
Arrange Tea	44.0	68.7	44.0	71.0	42.0	67.0	**70.0**	**84.3**	42.0	72.3
Bread Selection	54.0	78.0	46.0	71.6	60.0	74.0	**74.0**	**87.6**	62.0	80.4
Categorize Condiments	24.0	52.4	**34.0**	62.0	24.0	54.8	30.0	61.2	22.0	**63.6**
Cutting Tool Selection	58.0	64.7	64.0	74.0	60.0	68.0	72.0	75.3	**78.0**	**86.7**
Garnish Pancake	40.0	60.5	36.0	58.5	42.0	60.0	76.0	**89.5**	**80.0**	**89.5**
Gather Tableware	14.0	39.5	22.0	45.0	24.0	49.5	24.0	47.0	**30.0**	**52.5**
Heat Kebab Sandwich	16.0	54.9	8.0	47.7	24.0	59.1	**26.0**	**69.7**	**26.0**	68.3
Make Ice Lemonade	16.0	39.0	36.0	60.7	32.0	61.7	**52.0**	**80.0**	44.0	70.3
Pan Transfer	24.0	54.0	24.0	47.0	18.0	49.0	28.0	59.0	**38.0**	**63.5**
Portion Hot Dogs	10.0	41.6	14.0	48.8	8.0	44.4	28.0	65.6	**48.0**	**73.2**
Recycle Bottles By Type	66.0	80.0	44.0	70.3	54.0	69.4	**98.0**	**99.1**	78.0	92.6
Separate Freezer Rack	30.0	53.2	40.0	60.4	34.0	59.2	34.0	66.4	**48.0**	**73.2**
Waffle Reheat	74.0	85.0	60.0	68.3	64.0	72.7	86.0	92.3	**90.0**	**95.7**
Wash Fruit Colander	42.0	73.5	36.0	60.8	42.0	71.0	**52.0**	**81.8**	38.0	71.0
Weigh Ingredients	48.0	71.0	22.0	49.0	36.0	59.0	44.0	65.5	**54.0**	**77.5**
Composite-unseen mean	39.0	62.0	37.3	60.5	39.5	62.4	**54.6**	**75.9**	53.0	75.7
Overall mean	41.2	62.5	39.5	62.0	36.9	59.4	49.5	70.3	**50.8**	**71.8**

C.2 REAL-WORLD POLICY EVALUATION

Experimental setup and evaluation. To evaluate real-world policy learning from subtask annotations, we compare CASD with UVD and GR00T N1.6 on ALOHA **Table Setup** and **Table Bussing**.

The high-level policy uses the overhead camera view, while the low-level policy uses four camera views and the robot state. Camera images are captured at 30 fps, and the robot is controlled at 25 Hz.

A trial is successful only if all subtasks are completed successfully. The progress score awards credit for each successfully completed subtask, including subtasks completed after an earlier subtask has failed.

Task descriptions. Figure 7 illustrates the two tasks. In **Table Setup**, the robot arranges two plates and cutlery on the placemat and places a cup on the coaster. In **Table Bussing**, it transfers the tableware to the bussing tub, scrubs the placemat with a sponge, and puts the sponge in the tub.



Figure 7: Task sequences for **Table Setup** and **Table Bussing**. Each step pairs a subtask instruction with wrist and overhead camera views. Within each task, the sequence proceeds from left to right and from top to bottom.



Figure 8: Real-world experimental setup and task objects. Left: the ALOHA robot and workspace. Right: the objects used in the two tasks.

C.3 ANNOTATION QUALITY EVALUATION

We train a separate annotator for each dataset. Table 12 lists the number of labeled training episodes B used by CASD and RDD. We compare with UVD and RDD on the same evaluation episodes; RDD uses CASD's labeled training set as its reference database. Gemini assigns skill labels to baseline intervals; CASD is evaluated after language grounding on RoboCasa365. Both RoboCasa365 splits use the same annotator. OOD denotes the `target` split, whose task types also occur in annotator training.

Appendix C.3.1 specifies the AgiBotWorld-Alpha protocol following RDD's released data preparation and baseline implementation.

Table 12: Annotation budgets. Labeled training episodes B used by CASD and RDD for each dataset in Table 4.

	RC365 ID (`pretrain`)	RC365 OOD (`target`)	Allex	ALOHA	AgiBotWorld-Alpha
B	640	640	64	32	152

C.3.1 AGIBOTWORLD-ALPHA: PROTOCOL FOLLOWING RDD

We follow RDD (Yan et al., 2025) on the AgiBotWorld-Alpha supermarket task (task 327, "Pickup items in the supermarket"). Following its released converter, we take the first 190 episodes in the dataset metadata order, use the first 152 as labeled references, and evaluate on the remaining 38.[1] The dataset annotations specify six alternating Pick/Place occurrences per demonstration. The same 152 references train CASD and form RDD's retrieval database.

We run UVD and RDD with the released implementation using LIV features from the head camera. Baseline preprocessing retains one frame in six from the 30 Hz videos, giving 5 Hz inputs. The reported RDD baseline uses retrieval weight $\alpha = 1$ and prior weight $\beta = 0$. CASD uses the head camera with robot state and action at the native frame rate; its training settings are listed in Tables 8 and 9. Thus, reference episodes and evaluation episodes are shared, while each method retains its own feature representation.

For consistency with the other datasets, Table 4 reports B-F1, J-F1, and Edit as defined in Section 4.3. Gemini assigns Pick/Place labels to the baseline intervals. We merge CASD's predicted terminal Idle interval into the preceding subtask, and evaluate J-F1 and Edit only over the annotated Pick/Place span, clipping all methods to the same interval and merging contiguous repeated labels. These metrics are distinct from RDD's original decomposition mIoU, which averages each predicted segment's maximum IoU with any reference segment, first within each episode and then across episodes, without requiring a skill-label match.

Table 13 reports this prediction-centred mIoU together with RDD's boundary F1 for the reproduced baselines and for CASD, all on the same 38 test demonstrations. RDD's boundary F1 matches the five internal boundaries of the annotated spans one-to-one within 15 native frames (0.5 s at 30 fps), clipping predictions to the annotated span and merging adjacent equal-label predictions. With the unannotated tail kept as a separate idle segment, that segment has zero IoU with every annotated span under the prediction-centred metric; merging it into the last subtask, the convention used in Table 4, removes this penalty. The published RDD and UVD values were measured on 37 demonstrations of unknown identity; our reproduction on the 38-demonstration split matches them within 0.1 point. RDD's released configuration file sets $\beta = 0.25$ for this dataset while its documentation uses $\beta = 0$, so both are reported. Adding the two hand cameras does not improve over the head-camera model.

D LOCAL SUPERVISION

D.1 LOCAL TRAINING EXAMPLES

All interval boundaries are integer frame positions, as in Section 3.1.

[1] RDD reports 37 test demonstrations; its released converter with 190 episodes and an 80:20 split yields 38. We retain all 38 and reproduce both baselines on this same split.

Table 13: AgiBotWorld-Alpha supermarket task under RDD's decomposition metrics (%). Prediction-centred mIoU and RDD-style boundary F1 at 0.5 s on the same 38 test demonstrations; UVD and RDD are reproduced with the official RDD code. Published values are from the RDD paper.

Method	Setting	mIoU	B-F1
UVD	published (37 demos)	50.6	—
RDD	published (37 demos)	70.6	—
UVD	reproduced	50.7	7.4
RDD	$\beta = 0$	70.7	59.0
RDD	$\beta = 0.25$	71.1	45.5
CASD	head camera, tail kept	75.1	61.7
CASD	head camera, tail merged	**82.4**	**65.3**
CASD	head and hand cameras, tail kept	73.8	54.7
CASD	head and hand cameras, tail merged	81.2	58.4

For each episode, BUILDLOCALTARGETS constructs R local examples $\mathcal{S} = \{(I_r, k_r, q_r)\}_{r=1}^{R}$ from maximal contiguous reference skill occurrences. Here, k_r and q_r denote identity and extent targets ($\perp$: unsupervised). Full examples retain reference endpoints even outside $\mathcal{C}$.

Example	Interval	Identity	Extent
Full	Reference interval $[a, b)$	Reference skill	full
Partial	Prefix $[a, c)$ and suffix $[c, b)$	Reference skill	partial
Mixed	Union of two adjacent, different-skill occurrences	$\perp$	mixed
Perturbed	One endpoint of $[a, b)$ moved inward	Reference skill	$\perp$

For each reference interval $[a, b)$, we split at the proposal $c \in \mathcal{C}$ nearest to its midpoint, requiring $c - a \geq 20$ and $b - c \geq 20$. Ties favor the earlier boundary; no partial examples are added if no valid split exists.

We move one endpoint inward to $\mathcal{C}$, selecting the largest perturbation on each side with temporal IoU of at least 0.8 with the full interval. Before selection, discard perturbations whose anchor slice $\chi([a, b)) = (\lceil a/4 \rceil, \lceil b/4 \rceil)$ matches that of the full interval or any partial example in the episode. We retain at most one perturbation per slice per reference occurrence and score the selected intervals in native frames.

D.2 LOCAL LOSSES

Let $\mathcal{R}_{\text{id}} = \{r : k_r \neq \perp\}$ and $\mathcal{R}_{\text{ext}} = \{r : q_r \neq \perp\}$ index the supervised examples in one episode. If N_q is the number of extent targets of class q across the training set, set $w_q \propto 1/N_q$. The per-episode losses are

$$\mathcal{L}_{\text{id}} = -\frac{1}{|\mathcal{R}_{\text{id}}|} \sum_{r \in \mathcal{R}_{\text{id}}} \log p_{\text{id}}(k_r \mid I_r), \tag{6}$$

$$\mathcal{L}_{\text{ext}} = -\frac{\sum_{r \in \mathcal{R}_{\text{ext}}} w_{q_r} \log p_{\text{ext}}(q_r \mid I_r)}{\sum_{r \in \mathcal{R}_{\text{ext}}} w_{q_r}}. \tag{7}$$

E TRAINING PSEUDOCODE

Algorithms 1 and 2 describe scorer training and its differentiable partition computation. The procedure uses cached features from frozen VLA and text encoders; the temporal encoder, FiLM generator, and both prediction heads are optimized jointly.

E.1 JOINT OPTIMIZATION

Algorithm 1 uses the same segment score $s_{\boldsymbol{\theta}}$ for structured training and inference. Reference annotations define the training targets and constrain the compatible-path numerator; inference-time

decoding requires no reference annotations. For RoboCasa365, we use $d_{\min} = 20$ frames and train for 30 epochs with minibatches of 8 episodes, AdamW (Loshchilov & Hutter, 2019) with learning rate 3×10^{-4} and weight decay 10^{-4}, and gradient-norm clipping at 1.

Algorithm 1 Training the CASD segment scorer

Require: Reference demonstrations $\mathcal{D}_{\text{ref}}$; proposal boundaries; cached VLA and text features
Ensure: Trained scorer parameters $\boldsymbol{\theta}$

for each training episode n **do**
  Form $\mathcal{E}_n = \{(a, b) \in \mathcal{C}_n^2 : b - a \geq d_{\min}\}$
  $\mathcal{S}_n \leftarrow \textsc{BuildLocalTargets}(P_n^{\text{ref}}, \mathcal{C}_n)$ ▷ Appendix D.1
end for
Compute extent weights $w_q \propto 1/N_q$ from all training targets
for each training minibatch $\mathcal{N}_{\text{batch}}$ **do**
  for each episode $n \in \mathcal{N}_{\text{batch}}$ **do**
    Encode the episode with the temporal encoder
    Pool and score all local intervals in $\mathcal{S}_n$ and proposal intervals in $\mathcal{E}_n$
    Compute $\mathcal{L}_{\text{id}}^{(n)}$ and $\mathcal{L}_{\text{ext}}^{(n)}$ using their supervised targets
    $\mathcal{L}_{\text{struct}}^{(n)} \leftarrow 0$
    if a reference-compatible path exists **then**
      $(\log Z_n, \log Z_{\text{ref},n}) \leftarrow \textsc{LogPartitions}(s_{\boldsymbol{\theta}}, \mathcal{C}_n, \mathcal{E}_n, P_n^{\text{ref}})$ ▷ Algorithm 2
      $\mathcal{L}_{\text{struct}}^{(n)} \leftarrow (\log Z_n - \log Z_{\text{ref},n})/M_n$
    end if
  end for
  $\mathcal{L} \leftarrow \dfrac{1}{|\mathcal{N}_{\text{batch}}|} \displaystyle\sum_{n \in \mathcal{N}_{\text{batch}}} \left(\mathcal{L}_{\text{id}}^{(n)} + \mathcal{L}_{\text{ext}}^{(n)} + \mathcal{L}_{\text{struct}}^{(n)}\right)$
  Update $\boldsymbol{\theta}$ with AdamW using $\nabla_{\boldsymbol{\theta}} \mathcal{L}$
end for
return $\boldsymbol{\theta}$

Here, $\mathcal{N}_{\text{batch}}$ indexes the demonstrations in a minibatch, and M_n is the number of reference occurrences in episode n. Episodes without compatible paths retain both local losses and remain in the minibatch denominator.

E.2 Differentiable partition computation

Algorithm 2 computes the two log partition functions without enumerating complete paths. The forward value $\alpha(b)$ accumulates all labeled paths ending at boundary b. The value $\beta_m(b)$ accumulates compatible paths that end at b after matching the first m reference occurrences. Both use the segment scores defined in Section 3.2, with no transition potential. Write LSE for log-sum-exp, taking the result for an empty or entirely unreachable set to be $-\infty$ with zero gradient.

Algorithm 2 LogPartitions: all-path and reference-compatible log partition functions

Require: Segment scores $s_{\boldsymbol{\theta}}$, boundaries $\mathcal{C}$, legal pairs $\mathcal{E}$, reference P^{ref}
Ensure: $\log Z$, $\log Z_{\text{ref}}$ for the episode

Initialize $\alpha(0) \leftarrow 0$ and $\alpha(b) \leftarrow -\infty$ for $b \in \mathcal{C} \setminus \{0\}$
for $b \in \mathcal{C} \setminus \{0\}$ in increasing order **do**
  $\alpha(b) \leftarrow \text{LSE}_{(a,b)\in\mathcal{E},\, k\in\mathcal{K}} \left[\alpha(a) + s_{\boldsymbol{\theta}}([a, b), k)\right]$
end for
$\log Z \leftarrow \alpha(T)$
Initialize $\beta_0(0) \leftarrow 0$ and $\beta_0(b) \leftarrow -\infty$ for $b \in \mathcal{C} \setminus \{0\}$
for $m = 1, \ldots, M$ **do**
  $\mathcal{E}_m \leftarrow \{(a, b) \in \mathcal{E} : \text{tIoU}([a, b), I_m^{\text{ref}}) \geq 0.5\}$
  for $b \in \mathcal{C}$ **do**
    $\beta_m(b) \leftarrow \text{LSE}_{(a,b)\in\mathcal{E}_m} \left[\beta_{m-1}(a) + s_{\boldsymbol{\theta}}([a, b), k_m^{\text{ref}})\right]$
  end for
end for
$\log Z_{\text{ref}} \leftarrow \beta_M(T)$
return $(\log Z, \log Z_{\text{ref}})$

For two intervals, temporal IoU is their intersection length divided by their union length. A reference-compatible path has exactly M intervals with the prescribed skill order and satisfies the overlap constraint at every occurrence. Its existence can be checked before training by applying the same layered recurrence with Boolean reachability in place of log-sum-exp. The all-path computation places no restriction on segment count or skill order and allows adjacent segments with the same label. Reference boundaries are never inserted into $\mathcal{C}$ to create compatible paths.

Log-sum-exp aggregates competing paths to compute the structured training loss. Inference instead requires a single labeled partition, so we replace log-sum-exp in the all-path recurrence with maximization and backtrack the selected edges and labels. Because Z is constant across paths for a given episode, maximizing $S_{\boldsymbol{\theta}}(P)$ also maximizes $\exp S_{\boldsymbol{\theta}}(P)/Z$. Thus, max-score decoding returns the maximum a posteriori path under the same normalized path distribution used for training.

F Annotation Implementation

F.1 Candidate boundary generation

For RoboCasa365, we compute candidate boundaries from the raw proprioceptive state, before feature normalization, at the dataset frame rate of 20 Hz. We obtain the world-frame end-effector position by rotating the base-relative end-effector position with the base orientation and adding the base translation, and we compute its speed by central differences. After smoothing the speed with a Savitzky–Golay filter (window 5, polynomial order 2), we take its local minima whose prominence exceeds 5% of the range between the 5th and 95th percentiles of the smoothed speed. We detect gripper transitions from the finger separation with a hysteresis rule: the gripper becomes closed when the separation falls below 0.030 m and open when it rises above 0.050 m. Each transition is added as a candidate with a salience higher than that of any speed minimum. After merging candidates at the same frame, we greedily prune the candidate set until consecutive boundaries, including the episode endpoints 0 and T, are at least 20 frames (1 s) apart; within the closest violating pair, the candidate with lower salience is removed. Gripper transitions are therefore retained unless they lie within 20 frames of another transition or an episode endpoint. We do not cap the number of candidates. Every ordered pair of candidates at least $d_{\min} = 20$ frames apart defines a candidate interval (Section 3.2), and decoding may skip candidates, so a proposed boundary need not appear in the final annotation. The same procedure and parameters are applied to every demonstration. Proposal coverage results are in Table 15.

F.2 Language grounding procedure

We describe the RoboCasa365 grounding pipeline used to produce the final policy-training annotations. After global decoding, the VLM receives three synchronized full-episode videos, the full-task instruction, and the ordered skill intervals. Each interval includes its frame and second bounds, skill class, sentence template, and required and optional argument roles. The videos share the original episode time origin; simultaneous camera views are explicitly identified as such. Table 14 lists the request settings used in both grounding passes.

Table 14: VLM request settings for RoboCasa365 language grounding. The model name is the identifier recorded by the annotation service.

Setting	Value
Model identifier	`gemini/gemini-3.7-flash`
Camera views	Agent left, wrist, agent right
Video input	Full episode, 256×256
Requested sampling / resolution	1 FPS / medium
Temperature / reasoning effort	0 / low

The first pass fills the argument roles and returns one imperative instruction per interval. It retains the supplied skill unless visual evidence supports a correction among the five `PLACE` relations. Unresolved required arguments or an uncorrected skill–video mismatch produce a null instruction. The response records the skill, arguments, instruction, issue flags, and any correction reason.

The second pass revisits only intervals with null instructions. It receives the same videos and task instruction, the previous annotations, and the complete skill catalog. It may select another skill from

the catalog and fill its corresponding roles. Uncertain choices are marked with an inference flag and an evidence note. Both passes preserve interval IDs, order, and boundaries; non-null first-pass annotations are retained. Responses are checked for the requested IDs, permitted classes, required and optional role keys, and consistent correction fields. Invalid responses are retried with validation feedback.

For the final composite-seen release, we additionally normalize instruction wording using reviewed, task-specific text rules. These rules standardize synonyms and simplify selected object descriptions, including generic source locations and descriptions of a single task-relevant bowl or pot. They can remove identifying modifiers retained by the grounding prompt, so the final instructions are not verbatim VLM responses. This language normalization changes only the instruction text and its sequence representation; the skill labels, structured arguments, and temporal boundaries are retained. Separately, the release incorporates terminal `TASK_COMPLETE` predictions from the completion-aware annotator: the terminal region replaces the corresponding tail of the existing annotations, while interior completion predictions are ignored. No additional VLM prompt is used for these postprocessing steps.

G Endpoint Timing Analysis

G.1 Population and event anchors

We audit 48 demonstrations from 16 RoboCasa365 composite-seen task types in the `target` split. The eligible population contains 84 PICK and 54 non-terminal PLACE occurrences from 15 tasks, forming 25 task $\times$ skill-kind groups. PLACE includes `PLACE[IN]`, `PLACE[ON]`, and `PLACE[UNDER]`; episode-final PLACE and `PLACE[STACK]` are excluded. One task has no eligible occurrence.

For each reference interval, the event extractor finds the first appropriate gripper command from its start to 40 frames after its exclusive end. PICK uses the close/grasp command and PLACE the open/release command. All sources share the anchor selected for that reference occurrence. The offset is the exclusive segment endpoint minus the anchor, divided by the 20 Hz frame rate. This command-relative quantity is distinct from the simulator-state completion time; zero is not assumed to be the desired annotation boundary.

G.2 Matching and dispersion

For CASD, UVD, and RDD, predicted and reference segments are matched one-to-one within each episode and exact skill label by descending positive temporal overlap, breaking ties by segment starts. Simulator-driven text labels are mapped to broad PICK/PLACE kinds before the same overlap-based matching. The broader mapping means that coverage fractions are not a uniformly defined annotation-accuracy score. Human reference endpoints supply the fifth source.

Within each task and skill kind, we compute the median absolute deviation of matched offsets from their own median, without a Gaussian scaling factor. The reported mean gives each of the 25 groups equal weight. In contrast, the distribution panels pool individual occurrences. In Human reference/CASD/Simulator-driven/UVD/RDD order, 0/14/0/16/41 of the 138 eligible occurrences are unmatched and excluded from the dispersion statistic. All 619 finite matched endpoints remain in the figure. One RDD group contains one matched occurrence and consequently has zero MAD.

Restricting all sources to their 83 commonly matched occurrences retains all 25 groups. The mean MADs are 0.30/0.54/0.10/1.07/1.58 s in the same source order, preserving the primary ordering. This sensitivity check does not establish that missing endpoints are ignorable. The analysis describes endpoint timing on this population; it does not isolate a causal effect of dispersion on downstream policy success.

H Additional Analyses

H.1 Component ablations

Each variant in Table 5 is retrained without one component while the training data, features, and decoding stay fixed. Label macro-F1 scores skill labels on the 513 human-defined intervals of the in-domain split, so it isolates recognition from boundary placement; both our identity head and Gemini use a 24-label vocabulary, and macro-F1 averages the 12 observed classes. The role of each component in Section 3 gives a prediction for its removal: the extent head should affect boundaries but not labels; identity FiLM should affect labels but not boundaries; the structured loss, which compares complete partitions, should affect joint agreement; and the temporal convolutions, which shape every segment feature, should affect both.

Structured supervision improves joint agreement without moving boundaries. Removing the structured loss keeps the local losses and inference-time dynamic programming. Boundary F1 stays at 33.3%, while joint F1 falls to 58.8% and label macro-F1 to 86.8%. We observe little difference in boundary placement; we hypothesize that local scores already determine which proposals decoding selects, whereas path-level training sharpens the identity scores that label them. Partition-level supervision thus contributes to joint skill-and-interval agreement beyond local interval supervision.

Temporal context affects both boundaries and labels; instruction conditioning affects labels. Disabling the temporal convolutions lowers boundary F1 to 27.1% and label macro-F1 to 79.6%. Disabling identity FiLM lowers label macro-F1 to 84.9% but leaves boundary F1 at 33.5%, consistent with its use only in the identity head. Together with the extent and structured-loss removals, these results match the predicted roles: completeness scoring places boundaries, instruction conditioning refines labels, and partition-level training improves their joint agreement. Each removal measures a component's contribution in the presence of the others and does not estimate interactions.

H.2 Candidate boundary coverage

Table 15: Candidate boundary recall (%), pooled over each evaluation set; trajectory endpoints excluded.

Dataset	0.5 s	1.0 s
RC365 pretrain	50.8	87.3
RC365 target	68.6	90.5
Allex	92.1	100.0
ALOHA	85.9	99.6
AgiBot	88.9	99.5

Candidate proposals bound boundary recall. Decoding selects only among proposed boundaries, so candidate recall is an upper bound on the recall of decoded boundaries (Table 15). At 0.5 s, candidate recall is lowest on the two RoboCasa365 splits (50.8% and 68.6%), which also have the lowest boundary F1 in Table 4 (33.3% and 46.9%). At 1.0 s, candidate recall is at least 87.3% on every set. This pattern is suggestive of a coverage limit at fine tolerance on RoboCasa365.

Coverage explains little of the ALOHA joint F1 gap. However, this explanation holds only in part for ALOHA, where RDD attains higher boundary and joint F1 than CASD (Table 4). There, candidate recall at 0.5 s is 85.9%, just below RDD's boundary F1 of 86.9%, so coverage may account for part of the boundary gap (82.9% versus 86.9%). The joint F1 gap is larger (49.1% versus 60.4%). Because CASD reaches 99.7 Edit on ALOHA, its skill order is nearly correct; the joint F1 gap therefore more likely reflects interval extents that miss the 0.75 temporal IoU threshold than missing boundaries. Recall also does not measure selection precision, which we leave to future analysis.

I Cross-Task Skill Representations

I.1 Retrieval protocol

We extract all representations in evaluation mode from the same 513 human-defined intervals in HW80 (80 episodes and 16 task types), using the deployed PT640 SegmentFiLM checkpoint. Training and HW80 episode identities do not overlap, although the task types occur in training and HW80 was used during development. Human boundaries define the pooling intervals; skill labels do not enter feature extraction and are used only for retrieval evaluation. VLA mean pools the frozen 2048-dimensional VLA frame features. VLA F/L/M concatenates their first, last, and mean values (6144 dimensions), whereas Input F/L/M applies the same pooling to the training-normalized VLA, robot-state, and action inputs (6228 dimensions). Pre-FiLM and Post-FiLM are the learned 384-dimensional segment features immediately before and after instruction conditioning, respectively.

We L2-normalize each segment vector and compute cosine similarity in its original feature space; no learned projection or dimensionality reduction enters Table 16. Thirty intervals have no same-skill example in another task and are excluded as queries, leaving 483 queries across six canonical skills. For 1-NN and average precision (AP), the gallery contains the other HW80 intervals except those from the query's task type, and relevance is equality of the human skill label. Exact 1-NN ties select the lowest row index, while AP groups equal scores at a common threshold. Pair compares every same-skill interval from another task against every different-skill interval from another episode of the query's task, assigning one, zero, or one half for a higher, lower, or tied positive similarity. Each metric is averaged first over queries within a skill and then equally over the six supported skills.

I.2 Representation comparison

AP rises from 38.0% for frozen VLA mean features to 46.9% with F/L/M pooling (46.6% when state and action are added), 64.0% for the learned features before FiLM, and 71.5% after FiLM. Post-FiLM features exceed pre-FiLM features on 1-NN accuracy (83.9% versus 74.4%) and AP, but fall short on Pair (89.3% versus 91.5%). We hypothesize that instruction conditioning adds task information that raises similarity within a task, slightly lowering this pair-ordering score while improving skill recall. These results hold with supplied boundaries on this fixed population and gallery; the learned representation combines temporal modeling, state and action inputs, supervision, and FiLM, which this comparison does not separate.

Table 16: Skill representations. Skill-macro scores (%, higher is better).

Representation	1-NN	AP	Pair
VLA mean	55.9	38.0	67.0
VLA F/L/M	61.5	46.9	79.0
Input F/L/M	63.2	46.6	73.3
Pre-FiLM	74.4	64.0	91.5
Post-FiLM	83.9	71.5	89.3

J Qualitative Annotation Comparisons

J.1 Picking subtasks

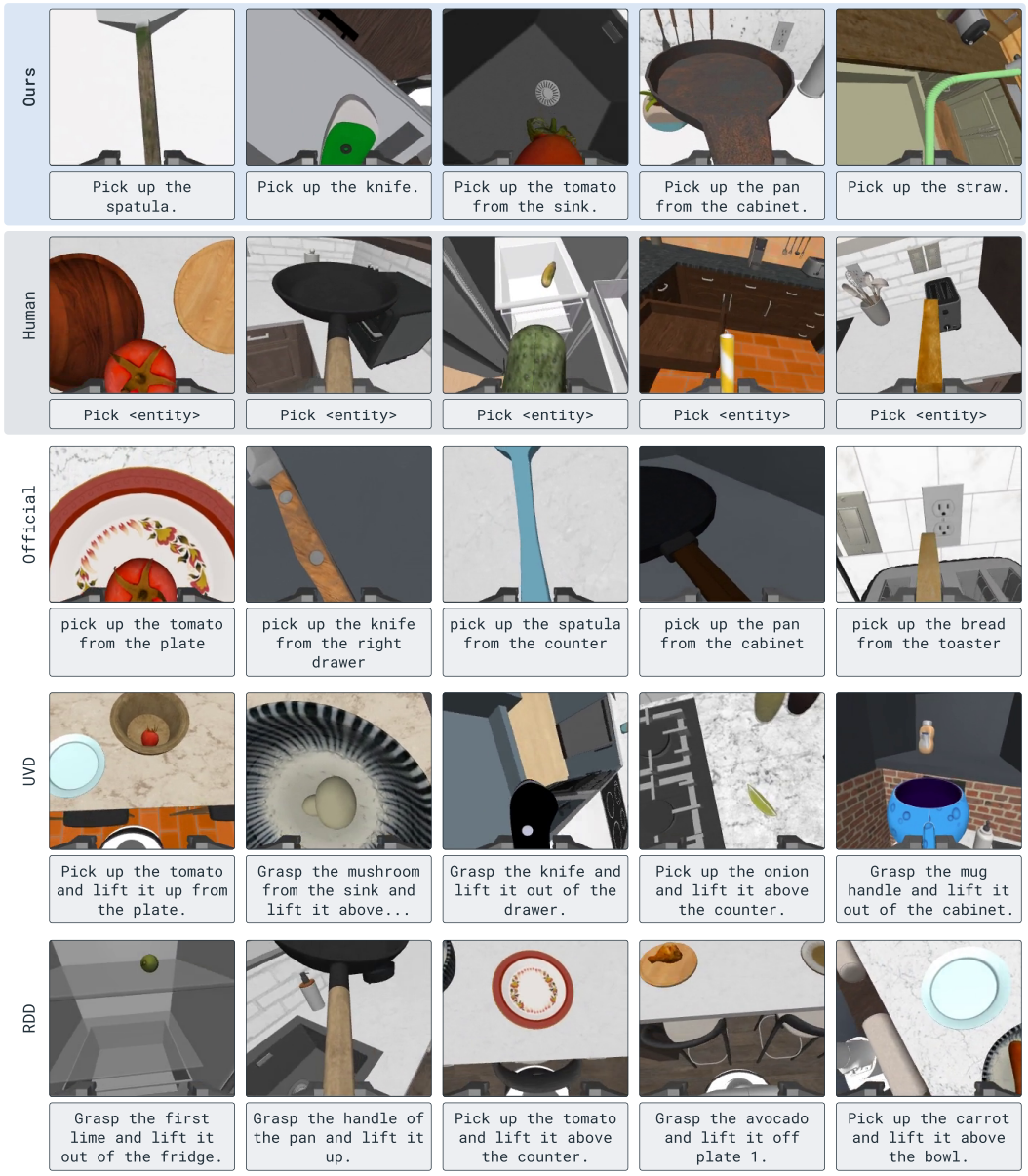

Figure 9: Consistency of terminal states for picking subtasks. Each row shows examples from different episodes, with the terminal frame and its associated annotation. Human denotes evaluation annotations, and Official denotes annotations provided by RoboCasa365. Examples are selected independently across rows and are not matched by column. Ours annotation exhibit consistent completion states across objects and scenes, whereas UVD and RDD show greater variation in where picking subtasks terminate.

J.2 ACTIVATION SUBTASKS

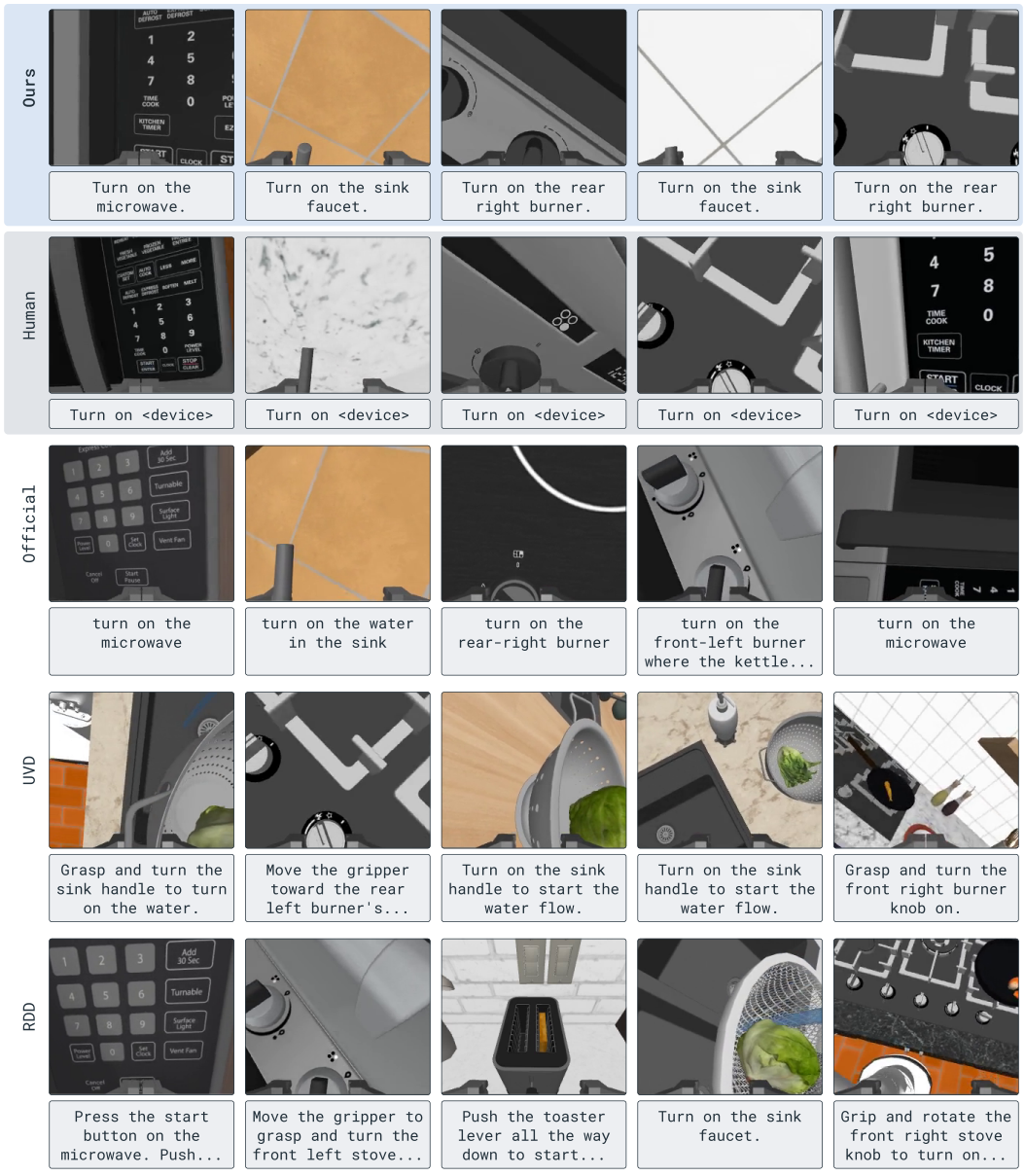

Figure 10: Consistency of terminal states for turn-on subtasks. Each row shows terminal frames and associated annotations from different episodes involving appliance or faucet activation. Examples are selected independently across rows and are not matched by column. Ours, the human evaluation annotations, and the official RoboCasa365 annotations exhibit consistent completion states for the activation skill across different targets. UVD and RDD show more variable terminal states, sometimes extending into subsequent robot motion despite retaining an activation-related instruction.

J.3 PLACEMENT SUBTASKS



Figure 11: Consistency of terminal states for placement subtasks. The left three columns show placing a knife on a cutting board; the right three show placing a bowl in a microwave. Ours annotation show consistent completion states across episodes. In contrast, UVD and RDD terminate at varying stages of placement and subsequent arm motion, despite describing the same placement goal.

J.4 REAL-WORLD SPONGE PICKING

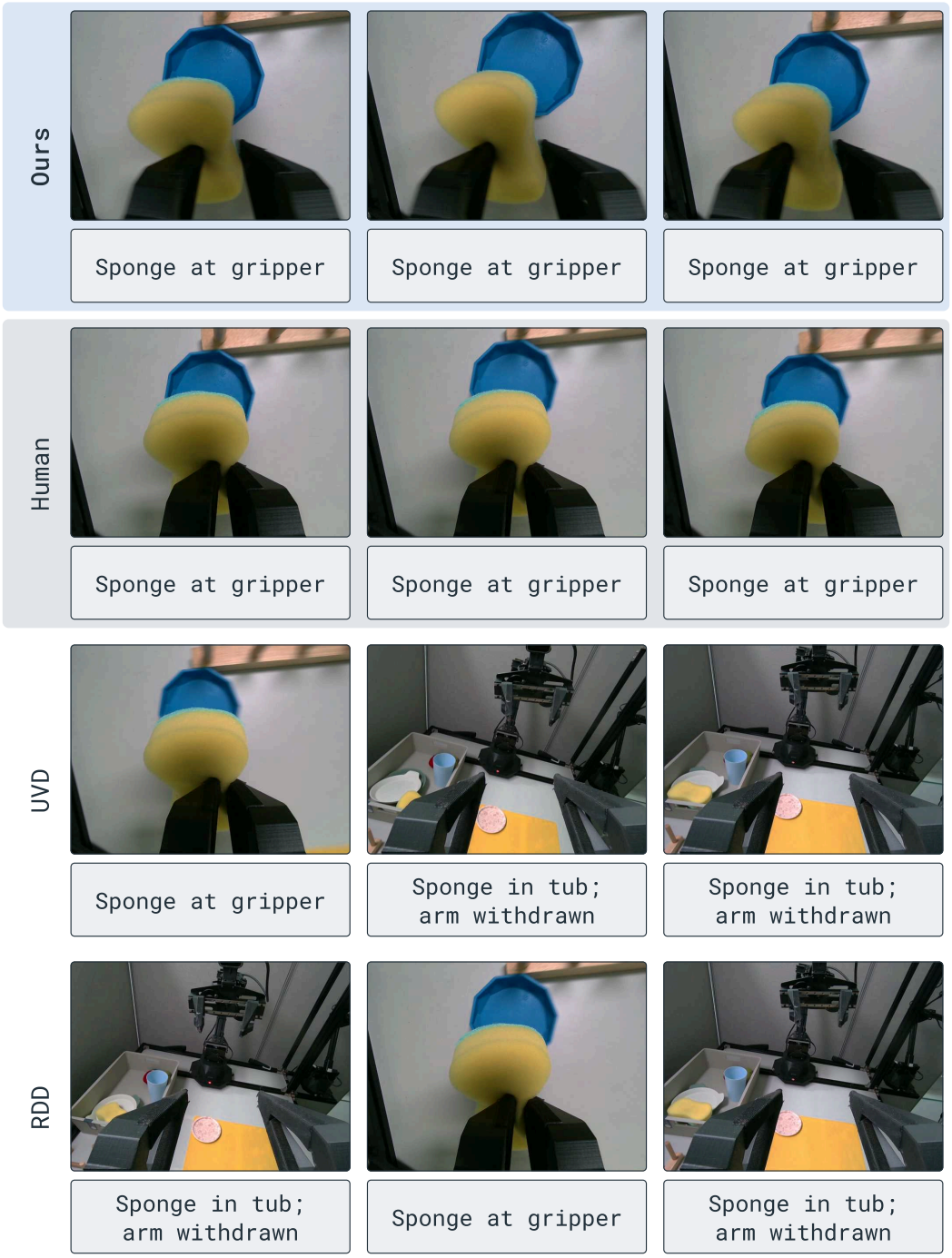

Figure 12: Comparison of terminal states for the sponge-picking subtask in the real-world Table Bussing experiment. The examples shown for Ours and Human consistently end with the sponge held by the gripper. UVD and RDD show more variable terminal states, including later states in which the sponge is already in the tub and the arm has withdrawn.