
Self-Supervised Learning of a Visual Object-Centric Representation for Robotic Manipulation

**Selbstüberwachtes Lernen einer visuellen objektzentrierten Repräsentation für die
robotergestützte Manipulation**

Master thesis in the department of Computer Science by Yi Li

Date of submission: March 23, 2026

1. Review: Prof. Jan Peters Ph.D.
2. Review: Prof. Liming Chen Ph.D.
3. Review: Alap Kshirsagar
4. Review: Alexandre Chapin
Darmstadt



TECHNISCHE
UNIVERSITÄT
DARMSTADT



LIRIS

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Yi Li, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 23. März 2026



Y. Li

Abstract

Pre-trained vision models for robotic manipulation produce either global scene embeddings or dense spatial grids, both of which mix task-relevant and task-irrelevant features. Object-centric representations offer a structured alternative by grouping visual features into discrete per-object slots. Each slot binds to a distinct scene region but operates in 2D feature space and lacks absolute spatial coordinates, limiting placement precision in millimeter-level manipulation tasks.

This thesis investigates how frozen object-centric representations can be integrated into continuous control policies under standard hardware constraints. We adapt the SPOT encoder, combining a frozen DINO ViT-B/16 backbone with Slot Attention, in a dual fixed-camera behavior cloning setup. Slots are combined with proprioceptive state and explicit 2D spatial goal tokens and processed by a causal Transformer policy with a short temporal window and multi-step action prediction, precomputed offline to remain trainable on an 11 GB GPU. On `PickCube-v1` in `ManiSkill3`, injecting projected goal coordinates raises success rate from 31.0% to 55.0%; native 224×224 rendering further improves the best configuration to $68.7 \pm 4.2\%$. Ablations against DINO baselines confirm that object-centric grouping outperforms uniform spatial sampling under matched token budgets, and a kinematic failure taxonomy shows that spatial grounding resolves placement failures without affecting grasping.

Zusammenfassung

Vortrainierte visuelle Modelle für die Roboter Manipulation erzeugen entweder globale Szeneneinbettungen oder dichte räumliche Gitter, die beide aufgabenrelevante und aufgabenirrelevante Merkmale vermischen. Objektzentrierte Repräsentationen bieten eine strukturierte Alternative, indem sie visuelle Merkmale in diskrete Objekt-Slots gruppieren. Jeder Slot bindet sich an eine bestimmte Szenenregion, arbeitet jedoch im zweidimensionalen Merkmalsraum ohne absolute räumliche Koordinaten, was die Platzierungsgenauigkeit bei millimetergenauer Manipulation einschränkt.

Diese Arbeit untersucht die Integration eingefrorener objektzentrierter Repräsentationen in kontinuierliche Steuerungsstrategien unter standardmäßigen Hardwarebeschränkungen. Wir adaptieren den SPOT-Encoder, der ein eingefrorenes DINO-ViT-B/16-Backbone mit Slot-Attention kombiniert, in einem Dual-Kamera-Setup für Behavior Cloning. Slots werden mit propriozeptivem Zustand und expliziten 2D-Ziel-Tokens kombiniert und von einer kausalen Transformer-Strategie mit kurzem Zeitfenster und mehrschrittiger Aktionsvorhersage verarbeitet, die offline vorberechnet auf einer 11 GB-GPU trainierbar bleibt. Auf `PickCube-v1` in `ManiSkill3` steigert das Einbinden projizierter Zielkoordinaten die Erfolgsrate von 31,0% auf 55,0%; natives 224×224 -Rendering verbessert die beste Konfiguration weiter auf $68,7 \pm 4,2\%$. Ablationen gegenüber DINO-Baselines bestätigen, dass objektzentrierte Gruppierung gleichmäßigem räumlichem Sampling überlegen ist, und eine kinematische Fehleranalyse zeigt, dass räumliches Grounding Platzierungsfehler behebt, ohne die Greifphase zu beeinträchtigen.

Acknowledgements

I thank Prof. Jan Peters and Prof. Liming Chen for enabling this collaboration between the Intelligent Autonomous Systems group at TU Darmstadt and the LIRIS laboratory at École Centrale de Lyon. I am grateful for this opportunity. The collaboration gave me access to perspectives and expertise I would not have found in one place alone.

Alexandre Chapin and Alap Kshirsagar supervised this work week by week from the start. Over more than sixty meetings, they answered every question I brought, reviewed my code alongside me, and helped me find direction after each failure. The challenges here were different from what I expected: there is no standard benchmark, no obvious evaluation protocol, and a lot of engineering that nobody writes down in papers. Alexandre and Alap helped me navigate all of it. The roadmap in this thesis reflects a long and winding path, and I could not have walked it without them.

I also thank Tim and Sabine at IAS, who helped me whenever I needed it—with registration, cluster access, and much else besides.

Chris and my parents were with me through all of it. Chris listened to every update, every setback, and every small win across the entire project. Their kind of steady presence matters more than I can properly say here.

AI Disclosure. I used Claude and Gemini throughout this project for code debugging, test script generation, experiment logging and code structure review. For writing, I used Claude, Gemini, and NotebookLM to verify reasoning, restructure drafts, and improve language. All research decisions, experimental design, analysis, and scientific conclusions remain entirely the author's own work.

Contents

Acknowledgements

1. Introduction	2
1.1. The Core Problem: Holistic vs. Object-Centric Representations	2
1.2. The Representation Gap in Physical Control	2
1.3. Proposed Approach	4
1.4. Research Questions	5
1.5. Contributions	5
2. Foundations and Related Works	7
2.1. Vision Foundation Models in Embodied AI	7
2.2. Object-Centric Learning: The Structural Bottleneck	8
2.3. Multi-View Fusion and Spatial Grounding	10
2.4. Sequence Models for Continuous Control	10
2.5. Object-Centric Representations in Robotic Manipulation	11
2.6. Summary and Identified Gaps	12
3. Methodology and Architectural Design	14
3.1. Problem Formulation and Simulation Environment	14
3.2. Proposed Method	15
3.2.1. Object-Centric Encoding and Spatial Grounding	16
3.2.2. Autoregressive Multimodal Sequence Policy	17
3.3. Systematic Evaluation Protocol	20
3.3.1. Unseen Seeds Protocol and Checkpoint Sweeping	20
3.3.2. Automated Kinematic Failure Taxonomy	20
4. Experiments and Implementation Details	22
4.1. System Infrastructure and Data Engineering	22
4.1.1. Encoder Evaluations	22

4.1.2.	Simulation Migration and Determinism	25
4.1.3.	Data Pipeline and Memory Optimization	27
4.1.4.	State and Action Space Engineering	28
4.2.	Visual Integration and Sequence Policy Architecture	29
4.2.1.	Proprioception Injection and Dimensionality	29
4.2.2.	Multi-View Integration and Fusion Interventions	30
4.2.3.	Autoregressive Multimodal Policy Refinement	34
4.3.	Algorithmic Interventions for Spatial Grounding	34
4.3.1.	Spatial Grounding via Goal Injection	35
4.3.2.	Algorithmic Interventions for the “Last Millimeter”	36
4.4.	Baseline Comparisons and Task Generalization	37
4.4.1.	The Core Object-Centric Pipeline	37
4.4.2.	Ablation Study: Visual Representations	38
4.4.3.	Task Generalization: The StackCube Challenge	39
5.	Results and Ablation Studies	41
5.1.	The Impact of Explicit Spatial Grounding	41
5.1.1.	Quantitative Success Rate Comparison	41
5.1.2.	Failure Taxonomy	42
5.1.3.	Effect of Camera Calibration and Rendering Resolution	47
5.2.	Baseline Comparison: Object-Centric vs. Global and Dense Features	48
5.2.1.	Quantitative Comparison	48
5.3.	Architecture Ablations	50
5.3.1.	Single-Camera Slot Processing	50
5.3.2.	Multi-View Fusion and the Generalization Gap	51
5.3.3.	Temporal Context Window (T)	51
5.3.4.	Proprioceptive State Dimensionality	53
5.4.	Algorithmic Interventions for the “Last Millimeter”	53
5.4.1.	Performance and Stability	54
5.4.2.	Failure Mode Shifts	54
5.5.	Task Generalization: The StackCube Challenge	56
5.5.1.	Native Setup vs. Aligned Configuration	56
5.5.2.	Failure Mode Analysis	57
6.	Discussion	59
6.1.	Representation Limits: Object-Centric Grouping vs. Dense Features	59
6.2.	Challenges in Multi-View Slot Integration	60
6.3.	The “Last Millimeter” Bottleneck and Spatial Grounding	61

6.4. Generalization and Temporal Logic in Multi-Stage Tasks	62
6.5. Evaluating Representations: Process Metrics vs. Binary Success	64
7. Conclusion and Future Work	66
7.1. Conclusion	66
7.2. Limitations	67
7.3. Future Directions	68
A. Dual-Camera Configuration and Projection Parameters	73
B. Offline Feature Caching Pipeline	75
B.1. Design Rationale	75
B.2. Chunked Dataset Format	75
B.3. Loading Strategy	76
C. Action Space Distribution Statistics	77
D. Failure Taxonomy: Classification Rules	79
E. SPOT Slot Attention Visualizations	81

List of Figures

1.1. High-level overview of the proposed system. Visual observations from dual cameras are encoded into object-centric slot representations by the frozen SPOT encoder. The Autoregressive Multimodal Sequence Policy maps the resulting token sequence to action chunks executed by the robot. A detailed architectural diagram is provided in Figure 3.1.	4
2.1. Slot Attention mechanism [2]. A feature extractor maps the input image to a dense $H \times W$ patch feature map, which serves as keys and values. K slot vectors act as queries and compete via softmax-normalized cross-attention. After each iteration, slots are updated through a Gated Recurrent Unit (GRU). The process repeats for T iterations, causing each slot to bind to a distinct image region. The output is K discrete slot vectors with corresponding attention masks, forming a compact, permutation-invariant scene representation.	9
3.1. Architecture of the proposed pipeline. RGB images from two fixed cameras are independently encoded by the frozen SPOT encoder into 7 object slots each. The 14 visual slots, a proprioception token, and a 2D goal token are projected to a shared 256-dimensional space and concatenated into a 17-token sequence per timestep. An Autoregressive Multimodal Sequence Policy decodes this sequence into a 10-step action chunk.	15
3.2. Evaluation pipeline. Checkpoints are saved every 50 epochs and evaluated offline against a pool of 1,000 held-out seeds (index $\geq 10,000$, never seen during training), sampling 200 episodes per run. Episodes terminating in fewer than 10 steps are discarded as trivial successes. The best checkpoint undergoes a stability test of three independent runs of 300 episodes each; we report mean \pm standard deviation. Failed episodes from a separate 200-episode failure analysis run are classified by an automated kinematic script into five categories: Idle, No Grasp, Drop, Near Miss, and Unknown.	21



4.1. Four auxiliary camera configurations evaluated for the dual-camera setup. Top-left: table-level side view, eliminated due to excessive dark background. Top-right: front-low view, eliminated due to overlap with the Base Camera. Bottom-left: wide-angle side view ($\mathbf{eye} = [0.5, 0.5, 0.4]$), eliminated because objects appeared too small. Bottom-right: diagonal side view ($\mathbf{eye} = [0.4, 0.3, 0.5]$, $\mathbf{target} = [0, 0, 0.2]$), selected for its complementary Z-axis coverage.	32
4.2. Goal definition for the StackCube-v1 task. The 3D goal coordinate is defined as the top-centre of the green cubeB: $\mathbf{g} = \mathbf{pos}_{\text{cubeB}} + [0, 0, 2 \times \text{half_size}]$. This is the target position that the centre of the red cubeA must reach for the task to succeed.	39
5.1. Training dynamics for the three spatial grounding configurations. Left: training loss over epochs, smoothed with a 10-epoch rolling average. Right: average reward per episode during training-time evaluation. Policies with explicit goal conditioning converge to lower loss and higher reward. Held-out Success Rate (SR) for each configuration is reported separately in Table 5.1.	43
5.2. Mean episode length across all evaluation episodes for the three spatial grounding configurations. Error bars show standard deviation over three independent evaluation runs of 300 episodes each. The 3D oracle reduces mean episode length by 42% compared to the pure visual baseline (182 ± 9 vs. 313 ± 6 steps), reflecting fewer exploratory adjustments during final placement.	44
5.3. Failure mode counts for the three spatial grounding configurations (200 episodes each, failures only). Near Miss episodes decrease sharply from 62 in the pure visual baseline to 5 with the 3D oracle, confirming that this failure mode is primarily caused by missing spatial information. No Grasp counts remain stable across all three configurations, indicating that spatial grounding specifically resolves the placement bottleneck without affecting the grasping phase. Success counts in this figure reflect single failure analysis runs and may differ slightly from the stability test results in Table 5.1.	45



5.4. Three representative Near Miss episodes from the 2D-Spatial-Proj configuration (top: Base Camera, bottom: Side Camera). In all three cases, the robot successfully grasps and transports the red cube to the goal vicinity but fails to achieve the required placement tolerance. The green sphere marks the 3D goal position. Near Miss accounts for 21.0% of all episodes in this configuration (Table 5.2).	46
5.5. Held-out SR for object-centric (SPOT) and holistic DINO representations on PickCube-v1. All configurations use the same policy and 2D goal conditioning. Error bars show standard deviation over three evaluation runs of 300 episodes each; the DINO 14 × 14 bar reports the sweep peak only (116 epochs trained in 48 hours).	49
5.6. Failure mode counts for loss weighting interventions (200 episodes each, failures only). U-Shaped weighting increases Drop failures from 33 to 54, indicating that over-emphasizing the placement phase destabilizes the grasping sub-policy. Dim-Decoupled scheduling reduces Near Miss from 42 to 30 but does not eliminate it, and increases No Grasp from 15 to 26. . .	55
5.7. Sub-task conversion funnel for Stack-Aligned (200 held-out episodes). Each stage reports the number of episodes that reached that sub-task milestone. The largest attrition occurs at the grasping stage (200 → 56), indicating that multi-object scene complexity disrupts the grasping sub-policy more severely than in the single-object PickCube-v1 task. The dominant failure mode is No Grasp (72% of all episodes).	58
6.1. Spatial quantization at 128 × 128 (left) vs. 224 × 224 (right) native rendering. Each grid cell represents one pixel. At 128 × 128, the cube and goal occupy the same pixel cell, making the spatial offset imperceptible to the encoder. At 224 × 224, finer pixel boundaries allow the offset to be detected, enabling more precise placement.	63
E.1. Case 1 (Clean Separation): Trajectory 988, Step 40. The cube and goal marker occupy different slots. Base Camera Slot 1 captures the goal marker; Slot 6 captures the cube. Slot 2 dominates (≈120), covering the table surface.	82
E.2. Case 2 (Merged Slot): Trajectory 20, Step 40. The cube and goal marker are close. Base Camera Slots 1 and 6 and Side Camera Slot 3 all show mixed attention covering both objects. Side Camera Slot 1 shows no goal response at this step.	83



E.3. Case 3 (Goal Collapse Under Occlusion): Trajectory 259, Steps 5–60. Goal slot response is lost from Step 40 onward and does not recover for the remainder of the episode.	84
E.4. Case 4 (Merged from Start): Trajectory 5, Step 5. The goal marker is visible only from the Base Camera. It appears merged with the cube in Base Camera Slot 6. No slot in either view isolates the goal alone at the start of the episode.	85
E.5. Case 5 (Slot Instability): Trajectory 786, Steps 5–60. The slot index for the same physical object shifts across frames. The encoder applies slot attention independently per frame with no temporal consistency constraint.	86

List of Tables

3.1. Summary of architectural hyperparameters for the proposed pipeline. The SPOT encoder is frozen throughout training. The policy Transformer operates on a sequence of 85 tokens constructed from 17 tokens per timestep over a temporal window of $T = 5$ frames. Goal conditioning parameters apply to the ablation experiments in Section 5.1.	19
4.1. Experimental development roadmap. Each cell represents a development stage in the order decisions were made. Bold items indicate the adopted solution at each stage; grey items were explored and discarded. The bottom row shows ablation experiments conducted after the final model was established.	23
4.2. Comparison of candidate simulation frameworks. CortexBench [14] relies on MuJoCo 2.0 and <code>mujoco-py</code> , incompatible with Python 3.10. LIBERO [27], CALVIN, and RLbench provide expert demonstrations but share legacy dependency constraints. Navigation-focused frameworks (BEHAVIOR-1K, ManipulaTHOR) support modern dependencies but target indoor navigation rather than tabletop manipulation. ManiSkill3 [9] is the only framework satisfying all requirements for this thesis, offering GPU-parallelized physics and native expert datasets for imitation learning.	26
4.3. Visual representation configurations. Visual tokens per timestep counts tokens from both cameras combined. Sequence length is computed over T timesteps, including 3 additional tokens per timestep (proprioception, goal, action). The DINO 14×14 dense configuration uses $T = 1$ and $H = 1$ due to memory constraints.	38

5.1. Comparison of spatial grounding configurations on <code>PickCube-v1</code> . All three policies use the same frozen SPOT encoder, dual-camera setup (128×128 rendering), and Autoregressive Multimodal Sequence Policy ($T = 5$, $H = 10$). SR and mean episode length are reported as mean \pm standard deviation over three independent evaluation runs of 300 held-out episodes each.	42
5.2. Failure mode distribution for the three spatial grounding configurations, evaluated over 200 held-out episodes each. Near Miss denotes episodes where the robot grasped and transported the cube to within 0.1 m of the goal but failed the placement tolerance. Drop denotes episodes where the cube was grasped but released before reaching the goal. No Grasp denotes episodes where the cube was never lifted. Unknown denotes episodes with ambiguous kinematic profiles.	43
5.3. Effect of camera calibration accuracy and rendering resolution on <code>PickCube-v1</code> SR. <code>SPOT-Approx-128</code> uses approximate camera parameters for the 2D goal projection at 128×128 native rendering. <code>SPOT-Exact-128</code> replaces these with exact extrinsic and intrinsic matrices at the same resolution. <code>SPOT-Exact-224</code> uses exact parameters and renders natively at 224×224 , eliminating the upscaling step. All other settings are identical ($T = 5$, $H = 10$, 1000 demonstrations). SR is reported as mean \pm standard deviation over three independent evaluation runs of 300 held-out episodes each.	47
5.4. Comparison of visual representation approaches on <code>PickCube-v1</code> . All configurations use the frozen <code>dino_vitb16</code> encoder or SPOT (which also uses a frozen DINO ViT-B/16 backbone), the same Autoregressive Multimodal Sequence Policy, and 2D goal projection. Visual tokens per timestep counts both cameras combined. The DINO 14×14 configuration uses $T = 1$ and $H = 1$ due to memory constraints. Training time per epoch is measured on an RTX 2080 Ti (11 GB); the DINO 14×14 configuration uses a reduced batch size of 4 due to memory constraints. SR is reported as mean \pm std over three evaluation runs of 300 episodes each, except DINO 14×14 which reports the sweep peak (116 epochs trained in 48 hours).	48

5.5. Comparison of single-camera slot processing architectures on <code>PickCube-v1</code> . All configurations use 7 SPOT slots from a single Base Camera, no explicit goal conditioning, and 7-Degrees of Freedom (DoF) proprioception. Parameter count refers to the slot processing and action decoding modules only, excluding the frozen SPOT encoder. SR is recorded from training-time evaluation runs of 30 episodes and is intended for qualitative comparison only.	50
5.6. Generalization gap for multi-view fusion strategies on <code>PickCube-v1</code> , evaluated <i>without</i> explicit goal conditioning. Training SR is measured on seen scene initializations during training. Held-out SR is measured over 300 episodes with held-out seeds (index $\geq 10,000$). Training SR values are approximate, recorded during training-time evaluation. Both strategies converge to the same held-out SR, confirming that architectural complexity does not improve generalization without an explicit spatial anchor.	51
5.7. Effect of temporal context window size on <code>PickCube-v1</code> , evaluated without explicit goal conditioning. All configurations use SPOT slots and the dual-camera setup. Sequence length is $17 \times T$ tokens. Training time per epoch is measured on an RTX 2080 Ti (11 GB). All runs trained for approximately 24 hours. Peak epoch and reward are recorded at the checkpoint with highest held-out SR. The identical SR across all three configurations indicates that temporal window size does not determine the performance ceiling without spatial goal information.	52
5.8. Effect of proprioceptive state composition on <code>PickCube-v1</code> . The 9-DoF configuration appends the two gripper finger joint positions (<code>qpos[7:9]</code> , range 0.018–0.040 m) to the 7 arm joint angles (<code>qpos[:7]</code>). The finger positions take only two values in practice (open/closed) and have near-zero variance ($\sigma \approx 0.0108$ m). Two normalization strategies were tested for the 9-DoF input. All configurations use the dual-camera Autoregressive Multimodal Sequence Policy without explicit goal conditioning. SR is the mean over three evaluation runs of 300 held-out episodes each, except where noted.	53
5.9. Success rates and evaluation variance for heuristic loss weighting interventions on <code>PickCube-v1</code> . All configurations use SPOT slots, dual-camera setup (128×128), approximate camera parameters, and 2D goal conditioning. The baseline is the <code>SPOT-Approx-128</code> configuration with standard Mean Squared Error (MSE) loss. SR is reported as mean \pm standard deviation over three independent evaluation runs of 300 held-out episodes each.	54



5.10. Comparison of StackCube-v1 configurations. The native setup uses the default Base Camera and dynamic wrist-mounted hand camera without goal conditioning. The aligned setup replaces the wrist camera with the fixed Diagonal Side Camera used in the PickCube-v1 experiments and adds a 2D goal token (target position defined as the top-centre of the green cube). Both configurations use the same Autoregressive Multimodal Sequence Policy ($T = 5$, $H = 10$) and 1000 expert demonstrations. SR for Stack-Native is reported over a single run of 200 held-out episodes. SR for Stack-Aligned is reported as mean \pm standard deviation over three independent runs of 300 held-out episodes each. 56

5.11. Failure mode distribution for Stack-Aligned, evaluated over 200 held-out episodes. No Grasp denotes episodes where the red cube was never lifted. Drop denotes episodes where the cube was grasped but released before reaching the target. Grasp No Stack denotes episodes where the robot lifted the red cube but never transported it to the vicinity of the green cube; this category is specific to the two-object stacking task. Near Miss denotes episodes where the cube reached within 0.1 m of the stacking target but failed the placement condition. 57

A.1. Camera intrinsic parameters for the two rendering resolutions used in this thesis. All experiments use square images ($W = H$). 73

A.2. Camera position parameters for the dual-camera setup. `eye` is the camera origin in world coordinates (metres). `target` is the point the camera looks at. 73

C.1. Action space statistics computed over 77,634 expert demonstration frames. Dimensions 0–6 are the 7 joint position targets (radians). Dimension 7 is the single gripper command, which outputs only two values: -1.0 (open) and $+1.0$ (closed). Its standard deviation ($\sigma = 0.9958$) is high because it is a binary signal, not a continuous one. Note that this is the *action output* dimension; the variance mismatch problem in Section 5.3.4 arises from the *proprioceptive input*, specifically the two gripper finger joint positions (`qpos [7 : 9]`), which have $\sigma \approx 0.0108$ m — far smaller than the joint angle targets. 77

D.1. Numerical thresholds used in the failure classification script. All thresholds are fixed across all experiments. 79

List of Abbreviations

ARI	Adjusted Rand Index.
BC	Behavior Cloning.
BCE	Binary Cross Entropy.
DoF	Degrees of Freedom.
FOV	Field of View.
GRU	Gated Recurrent Unit.
IL	Imitation Learning.
mIoU	mean Intersection over Union.
MSE	Mean Squared Error.
OCL	Object-Centric Learning.
OOM	Out-of-Memory.
SOCR	Slot-based Object-Centric Representation.
SR	Success Rate.
SSL	Self-Supervised Learning.
VFM	Vision Foundation Model.
VLA	Vision-Language-Action.

1. Introduction

1.1. The Core Problem: Holistic vs. Object-Centric Representations

Vision-based robotic manipulation typically relies on holistic scene representations. In this standard approach, Convolutional Neural Networks (CNNs) or Vision Transformers (ViTs) compress raw RGB images into a single global feature vector. While this method works well in simple, single-object environments, it struggles in cluttered scenes. Encoding the robotic manipulator, target objects, and background noise into one feature space limits compositional generalization. This entanglement of task-relevant and task-irrelevant features makes it harder for the policy to isolate target objects [1].

Object-Centric Learning (OCL) addresses this issue by decomposing the visual scene into discrete, permutation-invariant entities called slots [2]. Combined with recent advances in Self-Supervised Learning (SSL) [3], [4], it is now possible to extract object-centric semantic features directly from raw pixels. By representing objects independently, OCL provides a cleaner and more structured state space for downstream robotic control. Recent work shows that slot-based representations generalize better than global or dense features under visual distribution shifts [1].

1.2. The Representation Gap in Physical Control

Recent Vision Foundation Models (VFMs) provide dense, high-quality visual descriptors. However, deploying these models in high-frequency closed-loop control reveals a representation gap between computer vision features and kinematic execution. We identify three primary bottlenecks in this integration.

First, a **Dimensionality and Memory Bottleneck**. VFMs, such as the DINO architectures [3], [5], output dense spatial grids (e.g., 196 patch tokens per image frame). When deploying a multi-camera setup across a temporal context window, feeding these uncompressed sequences into a policy network causes high memory consumption. This makes training difficult on standard hardware.

Second, a **Task Objective Misalignment**. VFMs are optimized for static recognition tasks such as image classification or feature matching. Strong performance on these benchmarks does not predict success in continuous robotic control, where the relevant signal is not semantic similarity but precise kinematic execution.

To address the first two bottlenecks, dense VFM features can be substituted with object-centric slot representations [4], [6]. This reduces the input dimensionality significantly. However, this introduces a third, distinct bottleneck.

Third, the **“Last Millimeter” Spatial Bottleneck**. Purely visual object-centric slots lack absolute 3D coordinate awareness. This causes a systematic failure we term the “Near Miss”: the robot successfully grasps and transports the object but cannot place it within the required spatial tolerance due to depth ambiguity in the 2D visual input.

These bottlenecks together motivate the use of frozen-encoder approaches with explicit spatial grounding. State-of-the-art architectures such as Diffusion Policy [7] and the Action Chunking Transformer (ACT) [8] achieve high SR on complex continuous control benchmarks. However, they rely on end-to-end visual fine-tuning, which updates the vision backbone alongside the policy and requires substantial computational resources.

Standard Behavior Cloning (BC) using frozen visual features avoids this cost but performs poorly without further architectural support. Here, standard BC refers to a frozen visual encoder paired with a direct MLP mapping to actions, without sequence modeling or visual fine-tuning. The ManiSkill3 benchmark [9] reports 3% SR for this setup on `PickCube-v1` with 1000 demonstrations.

This thesis focuses on frozen-encoder BC as a lightweight alternative. We argue that in this setting, the key challenge is not feature quality but representation alignment: adapting existing object-centric features to the geometric and action-space requirements of physical control.

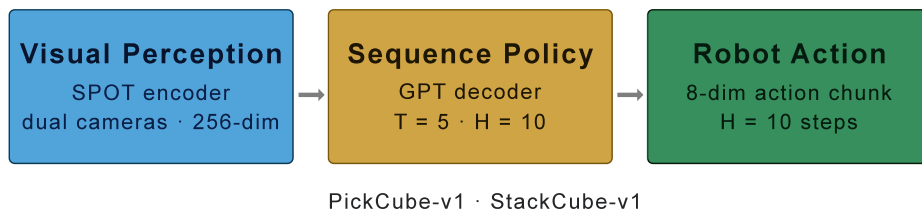


Figure 1.1.: High-level overview of the proposed system. Visual observations from dual cameras are encoded into object-centric slot representations by the frozen SPOT encoder. The Autoregressive Multimodal Sequence Policy maps the resulting token sequence to action chunks executed by the robot. A detailed architectural diagram is provided in Figure 3.1.

1.3. Proposed Approach

To address these bottlenecks, this thesis adapts the SPOT (Self-Training with Patch-Order Permutation) encoder [6] to extract compact object slots and integrates them with a sequence-based policy. The system consists of four components.

- **Dimensionality Reduction via SPOT.** The multi-view pixel space is compressed into 14 discrete object slots (7 per camera view), avoiding the $\mathcal{O}(L^2)$ sequence scaling of dense features. We compare this against global (CLS token) and patch-based (4×4 spatial grid) DINO representations to isolate the contribution of the slot bottleneck.
- **Autoregressive Sequence Policy.** An autoregressive decoder processes the unordered multi-view slots, proprioceptive state, and spatial goal as a unified token sequence. Causal self-attention handles cross-view geometric relationships without requiring fixed-order feature aggregation.
- **Explicit Spatial Grounding.** To diagnose the Near Miss bottleneck, we inject spatial goal coordinates as an additional input token. We evaluate two conditioning variants: a 3D oracle (direct simulator state) and a 2D projection (goal coordinates projected onto the camera planes). Both serve as diagnostic probes to quantify how much of the placement failure stems from missing spatial information. This setup relies on ground-truth coordinates; replacing the projection with a 2D object detector is a natural direction for future work.

-
- **Decoupled Action Decoding.** Continuous joint positions and the binary gripper state are normalized and decoded separately to prevent the high variance of the gripper from destabilizing joint position learning.

1.4. Research Questions

The central research question of this thesis is: **To what extent can frozen, self-supervised VFMs combined with object-centric representations be adapted for continuous robotic manipulation?**

Three specific questions guide the empirical investigation:

- **RQ1 (Spatial Grounding):** Are object-centric slots sufficient for precise object placement, or does injecting explicit spatial coordinates reduce the Near Miss error?
- **RQ2 (Multi-View Fusion):** How should permutation-invariant slots from multiple fixed cameras be fused in a sequence policy to reduce occlusion-related failures?
- **RQ3 (Action Space Heterogeneity):** How does the coexistence of continuous joint positions and a discrete binary gripper state in the action vector affect BC training stability, and how can the two be decoupled?

1.5. Contributions

The contributions of this thesis are:

- **Sequence-Based Integration of Object-Centric Slots.** We implement an autoregressive multimodal policy that processes 17 tokens per timestep, combining multi-view object slots, proprioceptive state, and a spatial goal token with Action Chunking. We compare this against global and patch-based DINO representations to evaluate the contribution of the slot bottleneck. Our decoupled pipeline—frozen SPOT encoder with offline feature caching—achieves $68.7 \pm 4.2\%$ held-out SR on PickCube-v1 on a single RTX-2080-Ti (11 GB), without end-to-end visual fine-tuning.

-
- **Diagnostic Isolation of the Spatial Bottleneck.** We use 3D oracle and 2D projected goal coordinates as diagnostic probes to confirm that the absence of absolute spatial information is the primary cause of Near Miss failures. This isolates the geometric limitation of unsupervised slot representations from other policy design factors.
 - **Structured Failure Analysis.** We apply a held-out seed evaluation protocol and a kinematic failure taxonomy that separates grasp failures from placement errors. This provides a more informative characterization of policy behavior than binary success rates alone.

2. Foundations and Related Works

This chapter provides the background for our proposed architecture. We first review VFMs and their computational costs in robotics (Section 2.1). Next, we introduce OCL as a method to compress and structure visual data (Section 2.2). We then discuss multi-view fusion and spatial grounding (Section 2.3), followed by sequence-based policy networks (Section 2.4). We then review recent applications of Slot-based Object-Centric Representations (SOCRs) in robotic manipulation (Section 2.5) and summarize the specific research gaps this thesis addresses (Section 2.6).

2.1. Vision Foundation Models in Embodied AI

Recent research in computer vision is primarily driven by SSL and Vision Transformers (ViTs). Models such as the DINO family [3], [5] extract robust visual features without human annotations. They naturally capture object boundaries and part-level semantics directly from raw pixels.

Consequently, recent Embodied AI literature frequently adopts these task-agnostic representations for continuous robotic control. Visual representations like R3M [10] and VIP [11] leverage large-scale pre-training to provide general-purpose features. Building directly on DINO’s pixel-level semantics, DINOBot [12] uses DINO features for image-level retrieval and pixel-level end-effector alignment in imitation learning. Similarly, THEIA [13] distills knowledge from multiple VFMs into a unified spatial token format to support robotic systems. A systematic evaluation across multiple pre-trained representations [14] found that no single model consistently outperforms others across all manipulation and navigation tasks.

However, transitioning these models to real-time physical control introduces a major computational bottleneck. Models like THEIA and DINOv2 output dense spatial grids (e.g.,

196 tokens per image). Modern robotic policies typically utilize Transformer networks. The self-attention mechanism in these networks scales quadratically ($\mathcal{O}(L^2)$) with the sequence length L . Feeding uncompressed dense grids across a multi-frame observation window induces high memory consumption. This computational cost renders raw dense features impractical for high-frequency control loops on standard hardware.

This scaling problem is particularly relevant for Vision-Language-Action (VLA) models [15], [16], which feed visual tokens from a DINO or ViT backbone directly into a large language model for action prediction. Concatenating dense visual token sequences with long language contexts makes the $\mathcal{O}(L^2)$ cost especially prohibitive in this setting.

2.2. Object-Centric Learning: The Structural Bottleneck

To resolve the dimensionality problem of dense features, OCL decomposes complex scenes into distinct entities. A scalable approach for this decomposition is Slot Attention [2].

Slot Attention utilizes a competitive cross-attention algorithm (see Figure 2.1). It initializes a set of permutation-invariant vectors called slots. These slots compete to explain different parts of the input feature map. Over successive iterations, each slot binds to a specific object or background element, compressing a complex image into a few semantic vectors. Subsequent works have extended this approach to video data (e.g., SAVi and SAVi++ [17], [18]) to track objects through dynamic occlusions.

Integrating SSL foundation models with Slot Attention significantly improves object discovery. For example, DINOSAUR [4] demonstrated that using frozen DINO features as reconstruction targets for Slot Attention outperforms using raw RGB pixels. SPOT (Self-Training with Patch-Order Permutation) [6] further refined this pipeline by incorporating autoregressive Transformers and knowledge distillation. More recent work integrates slot-based representations with diffusion models [19], improving generation quality on complex real-world scenes. Our work does not address scene generation but uses frozen slot representations directly for downstream control.

In robotic manipulation, models like SPOT serve as an effective structural bottleneck. By compressing 196 dense patches into 7 object slots, OCL prevents the $\mathcal{O}(L^2)$ sequence explosion in downstream Transformers (see Section 3 for configuration details). Moreover, this discrete representation isolates the robotic manipulator, the target object, and the background, providing a disentangled state space for the policy.

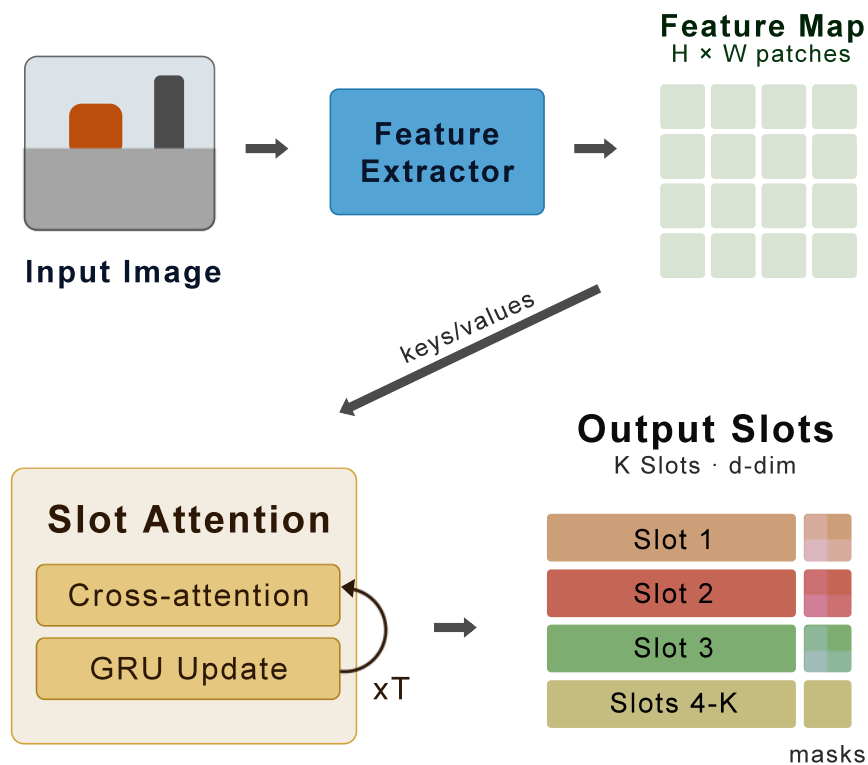


Figure 2.1.: Slot Attention mechanism [2]. A feature extractor maps the input image to a dense $H \times W$ patch feature map, which serves as keys and values. K slot vectors act as queries and compete via softmax-normalized cross-attention. After each iteration, slots are updated through a GRU. The process repeats for T iterations, causing each slot to bind to a distinct image region. The output is K discrete slot vectors with corresponding attention masks, forming a compact, permutation-invariant scene representation.

2.3. Multi-View Fusion and Spatial Grounding

While OCL successfully isolates objects, single-view 2D slots lack absolute 3D spatial information. This depth ambiguity causes precision issues in manipulation tasks. The policy often correctly identifies the target object but struggles to determine the exact placement coordinates.

Multi-view camera configurations are commonly employed to reduce occlusions and restore spatial information. However, fusing permutation-invariant slots from different cameras presents a topological challenge. Standard concatenation breaks the spatial correspondence between views. To address this, recent works explore explicit matching mechanisms based on temporal consistency [20] to track entities consistently across cameras.

Recent work on multi-view object-centric learning [21] addresses slot correspondence across views by learning identifiable representations, but requires camera pose supervision. Our approach instead relies on causal self-attention to learn cross-view alignment implicitly from data.

Beyond multi-view matching, explicitly providing 3D geometry is important for kinematic precision. VGGT [22] is a feed-forward network that predicts 3D scene attributes—including camera parameters, depth maps, and point clouds—directly from multiple input views. Foundation models like Depth Anything V2 [23] provide accurate monocular depth estimates from single images. These approaches demonstrate that explicitly recovering 3D geometry is critical for precise spatial reasoning. Building on this insight, while avoiding the computational overhead of dense depth prediction, we propose that injecting explicit spatial coordinates directly into the policy provides a lightweight and practical route to improve placement precision.

2.4. Sequence Models for Continuous Control

After extracting object slots, the policy network must translate them into physical actions. Early BC policies utilized Multi-Layer Perceptrons (MLPs). However, flattening discrete object slots into a 1D vector for MLP processing destroys their set topology and cross-view relationships.

To address this, the field has moved towards sequence models. This includes Transformer-based VLA architectures like RT-2 [15] and OpenVLA [16], as well as multi-task policy transformers like BAKU [24]. While these architectures process dense image patches as tokens, the Transformer architecture inherently supports unordered sets. This makes it well-suited for our approach: integrating OCL slots, proprioceptive states, and spatial goals without relying on a fixed spatial grid.

Predicting actions frame-by-frame often introduces mechanical jittering and compounding errors. To ensure smooth kinematic execution, modern sequence policies adopt Action Chunking, as used in the Action Chunking Transformer (ACT) [8] and Diffusion Policy [7]. Instead of predicting a single action a_t , the decoder predicts a short trajectory window, providing the temporal smoothness required for precise manipulation.

Results from the ManiSkill3 benchmark [9] highlight the cost of visual fine-tuning. State-based variants of ACT and Diffusion Policy achieve high SR on tasks such as PickCube-v1. Their visual counterparts require end-to-end fine-tuning of the vision backbone to achieve comparable results: with 100 demonstrations, ACT and Diffusion Policy reach only 28% and 76% SR respectively on PickCube-v1, rising to 98% and 100% with 1000 demonstrations [9]. Standard BC with frozen visual features achieves only 3% SR with 1000 demonstrations and 0% with 100, confirming that frozen features alone are insufficient without architectural support. The ManiSkill3 benchmark reports ACT and Diffusion Policy on an RTX-4090 (24 GB VRAM) [9]; end-to-end gradient flow through the vision backbone requires high-end hardware to maintain sufficient batch size for stable training.

2.5. Object-Centric Representations in Robotic Manipulation

Most state-of-the-art robot policies still use global or dense features from pre-trained vision models. These features often mix task-relevant and irrelevant signals, reducing robustness to visual distribution shifts [1].

To overcome this, some pipelines use supervised models such as the Segment Anything Model (SAM) [25] to extract localized object masks. These methods provide clear boundaries but require external supervision and additional inference compute.

A large-scale study by Chapin et al. [1] compared global, dense, and SOCR approaches across simulated (MetaWorld [26], LIBERO [27]) and real-world manipulation tasks. SOCRs outperformed both global and dense representations under visual distribution shifts,

including changes in lighting, texture, and the presence of distractors. Separately, Jiang et al. [28] showed that pre-training on large-scale robot-specific video data improves the manipulation relevance of visual representations, complementing the structural benefits of object-centric approaches.

Earlier work by Heravi et al. [29] and Haramati et al. [30] demonstrated that object-aware representations improve sample efficiency and generalization in visuomotor control compared to global or raw image-based approaches. However, both works rely on supervised object detectors or privileged state information to obtain object-centric inputs, rather than purely self-supervised decomposition from raw pixels.

Recent architectures such as SlotVLA [31] and SOLD [32] further validate this direction by incorporating object-relation representations and slot-based latent dynamics. Despite these structural benefits, a critical gap remains for tasks requiring geometric precision. Purely unsupervised SOCRs operate in 2D feature space and lack absolute 3D coordinates. No systematic method currently exists to ground these discrete slots into the spatial geometries required for millimeter-level manipulation.

Some object-centric models do incorporate 3D spatial structure. OSRT [33] learns 3D-consistent object decompositions through novel view synthesis. However, such methods are designed for scene representation and rendering, and have not been applied to closed-loop robotic control.

2.6. Summary and Identified Gaps

Existing research establishes a solid foundation for VFMs and SOCRs, but applying these models to closed-loop robotic control reveals three specific gaps.

First, a gap in **absolute spatial grounding**. Unsupervised SOCRs excel at separating objects from the background, but they lack 3D spatial awareness. How to achieve millimeter-level placement precision using 2D slots without falling back on dense features remains an open problem.

Second, a gap in **object-centric sequence integration**. Sequence models are now standard for continuous control, but the methodology to tokenize, fuse, and decode unordered multi-view slot entities into stable kinematic trajectories requires further investigation.

Finally, a gap in **representation evaluation**. Visual encoders are often evaluated together with customized policy networks. This tight coupling makes it difficult to determine

whether performance gains come from the visual representation or from the policy architecture.

This thesis addresses these gaps by developing a unified sequence-based policy that integrates explicit spatial grounding for SOCRs. Evaluating this architecture on a decoupled simulation infrastructure allows us to isolate the empirical contribution of frozen object-centric representations in continuous manipulation.

3. Methodology and Architectural Design

This chapter details the methodology and architecture developed to evaluate object-centric visual representations in robotic manipulation. We formally define the simulation environments, the state-action spaces, the dual-camera visual perception pipeline, and the sequence-based policy network. Finally, we establish an evaluation protocol to ensure objective performance measurement.

3.1. Problem Formulation and Simulation Environment

We formulate the robotic manipulation task as a discrete-time Markov Decision Process (MDP) defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. At each timestep t , the agent receives a state observation $s_t \in \mathcal{S}$ and executes an action $a_t \in \mathcal{A}$. While a full MDP utilizes transition dynamics \mathcal{P} and a reward function \mathcal{R} , we employ BC, a supervised Imitation Learning (IL) method that bypasses explicit reward modeling by reducing the problem to supervised learning over state-action pairs.

Consider a dataset of N expert demonstrations $\mathcal{D} = \{\tau_i\} * i = 1^N$. Each trajectory $\tau_i = \{(s_t, a_t^*)\} * t = 1^{T_i}$ consists of sequential state observations s_t and corresponding ground-truth expert actions a_t^* . The objective is to train a parameterized policy π_θ that minimizes the MSE:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s_t, a_t^*) \sim \mathcal{D}} [\|\pi_\theta(s_t) - a_t^*\|^2] \quad (3.1)$$

where $\pi_\theta(s_t)$ is the action predicted by the policy given the current state.

All experiments are conducted within the ManiSkill3 simulation framework [9] using the deterministic `physx_cpu` backend. We use two manipulation tasks:

- **PickCube-v1**: A single-object task requiring a Franka Panda arm to grasp a red cube and transport it to a target 3D coordinate.

- **StackCube-v1**: A multi-object task requiring the robot to grasp a red cube and place it stably on top of a green cube.

3.2. Proposed Method

Figure 3.1 provides an architectural overview of the proposed pipeline. The system operates in two stages: extracting structured visual features via an object-centric encoder, and translating these discrete slot representations into physical trajectories via an Autoregressive Multimodal Sequence Policy.

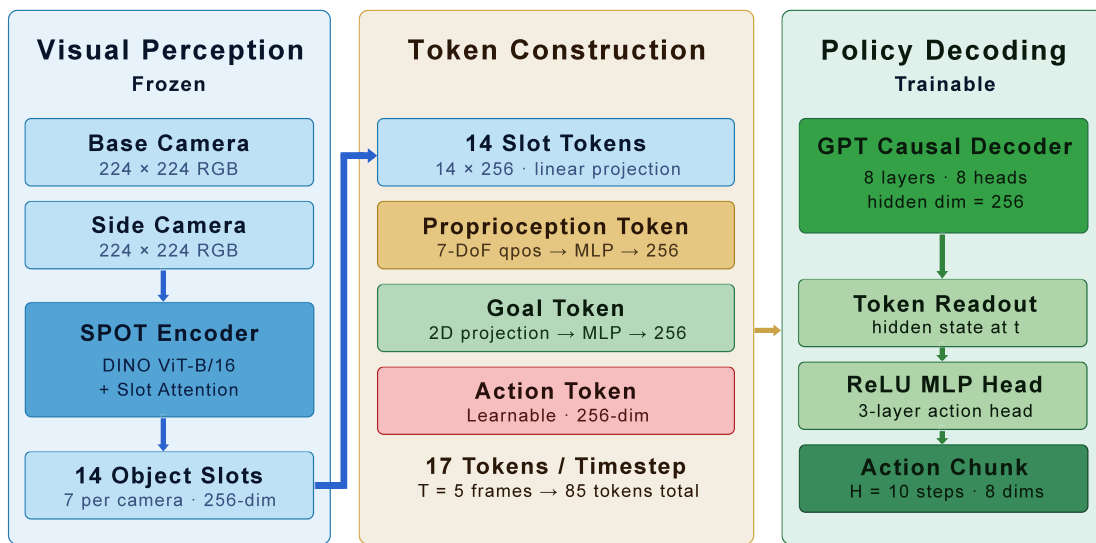


Figure 3.1.: Architecture of the proposed pipeline. RGB images from two fixed cameras are independently encoded by the frozen SPOT encoder into 7 object slots each. The 14 visual slots, a proprioception token, and a 2D goal token are projected to a shared 256-dimensional space and concatenated into a 17-token sequence per timestep. An Autoregressive Multimodal Sequence Policy decodes this sequence into a 10-step action chunk.

3.2.1. Object-Centric Encoding and Spatial Grounding

We employ a fixed dual-camera configuration comprising a Base Camera (frontal top-down view) and a Diagonal Side Camera (lateral elevation view) to capture complementary geometric profiles. Both cameras render RGB images at 224×224 pixels.

The raw images are independently processed by the frozen SPOT encoder (Self-Training with Patch-Order Permutation) [6], which uses a DINO backbone [3] followed by a Slot Attention module. The encoder produces $K = 7$ object slots per camera view ($S \in \mathbb{R}^{K \times 256}$). This value of K is adopted from the official SPOT weights pre-trained on the COCO dataset and covers all distinct objects present in our tabletop scenes. Following established practice in visual-motor policy learning, the encoder weights are kept frozen throughout training, so that observed performance differences reflect the properties of the visual representation rather than task-specific fine-tuning [10], [34].

The encoding process follows two stages:

- **Dense Feature Extraction:** The frozen DINO backbone processes the 224×224 image into a dense spatial grid of 196 patches, producing input features $X \in \mathbb{R}^{N_{\text{patches}} \times D_{\text{in}}}$.
- **Slot Attention Bottleneck:** A permutation-invariant Slot Attention module iteratively routes these dense patches into $K = 7$ discrete object-centric slots.

The input features are linearly mapped to keys $k(X)$ and values $v(X)$, while the slots act as queries $q(S)$. The attention mechanism normalizes over the slot dimension, so that slots compete to explain different parts of the feature map. The attention matrix M and normalized weights W are:

$$M_{i,j} = \frac{k(X_i) \cdot q(S_j)}{\sqrt{D}} \quad (3.2)$$

$$W_{i,j} = \frac{\exp(M_{i,j})}{\sum_{l=1}^K \exp(M_{i,l})} \quad (3.3)$$

where D is the latent feature dimension. The slots are updated via a Gated Recurrent Unit (GRU):

$$S^{(t)} = \text{GRU} \left(S^{(t-1)}, W^\top v(X) \right) \quad (3.4)$$

The above describes the core Slot Attention mechanism [2]. SPOT extends this with an autoregressive patch-order permutation decoder and self-training via attention mask distillation [6].

The resulting slot features from both cameras are concatenated to form the visual state matrix $S_{\text{vis}} \in \mathbb{R}^{14 \times 256}$.

While these slots successfully separate semantic entities, 2D features inherently lack absolute 3D spatial information. To study the effect of spatial grounding, we evaluate three goal-conditioning paradigms:

- **Implicit Visual Baseline:** No coordinate information is provided. The policy relies entirely on the visual slots.
- **Explicit 3D Oracle:** The ground-truth 3D world coordinates of the target are directly injected as a goal token.
- **2D Spatial Projection:** The 3D target coordinates are projected into the 2D pixel space of each camera using the simulator-provided intrinsic matrix K and extrinsic matrices (R, t) . A 3D world point P_{world} is first transformed to the camera frame: $P_{\text{cam}} = R \cdot P_{\text{world}} + t$. The 2D pixel coordinates (u, v) are then computed via perspective projection:

$$u = f_x \frac{X_{\text{cam}}}{Z_{\text{cam}}} + c_x, \quad v = f_y \frac{Y_{\text{cam}}}{Z_{\text{cam}}} + c_y \quad (3.5)$$

where f_x, f_y are the focal lengths and c_x, c_y is the principal point. The resulting coordinates are clipped and normalized based on the native rendering resolution, yielding a 4D vector $g = [u_{\text{base}}, v_{\text{base}}, u_{\text{side}}, v_{\text{side}}]$, which is encoded via a 3-layer MLP into a 256-dimensional Goal Token.

3.2.2. Autoregressive Multimodal Sequence Policy

The core causal self-attention and Transformer block implementation follows the GPT architecture provided by our supervisor as a starting codebase. Building on this foundation, we redesigned the multimodal tokenization pipeline, replacing the original language conditioning with a 2D spatial goal token, implementing the 17-token sequence structure for dual-camera slot inputs, replacing the bounded Tanh action head with an unbounded ReLU output, and adding the decoupled gripper normalization strategy described in Section 4.1.4.

To fuse unordered multi-view slots and multimodal state inputs, we replace a standard MLP mapping with an Autoregressive Multimodal Sequence Policy based on a causal Transformer decoder. The Transformer treats each visual slot, proprioceptive state, and

goal coordinate as an independent token, preserving the set structure of the slots without requiring fixed-order concatenation.

At each timestep t , all inputs are projected to a uniform 256-dimensional latent space and concatenated into a 1D sequence. A single temporal frame comprises 17 tokens:

- **14 Visual Slot Tokens:** Aggregated from dual-camera views.
- **1 Proprioception Token:** Encoding the 7-DoF joint state.
- **1 Goal Token:** Projected from the selected spatial conditioning paradigm.
- **1 Learnable Action Token:** Appended at the end of the sequence to query the action prediction.

Learned positional embeddings are added to the sequence to preserve modality and ordering information.

Temporal Context and Causal Masking. The policy maintains an observation window of $T = 5$ historical frames, giving a total sequence length of $17 \times 5 = 85$ tokens. The core architecture is an 8-layer, 8-head Transformer decoder with hidden dimension 256. A lower-triangular causal attention mask ($M_{i,j} = -\infty$ if $j > i$) ensures that each prediction depends only on past and present observations.

Readout and Action Chunking. After passing through the Transformer layers, we extract the hidden state of the Learnable Action Token at the current timestep t . This 256-dimensional vector is fed into a 3-layer ReLU MLP that predicts a trajectory chunk of $H = 10$ future timesteps. Action Chunking [7], [8] reduces compounding errors from frame-by-frame prediction and improves trajectory smoothness.

Heterogeneous Action Decoding. The 7 continuous joint dimensions are normalized with Z-score normalization and decoded by a ReLU MLP. The 1-dimensional gripper state bypasses this normalization and is passed through a hard step function, outputting binary values (-1.0 or 1.0). This decoupling prevents the high variance of the gripper dimension from destabilizing joint position learning.

Table 3.1.: Summary of architectural hyperparameters for the proposed pipeline. The SPOT encoder is frozen throughout training. The policy Transformer operates on a sequence of 85 tokens constructed from 17 tokens per timestep over a temporal window of $T = 5$ frames. Goal conditioning parameters apply to the ablation experiments in Section 5.1.

Component	Parameter	Value
SPOT Encoder	Backbone	DINO ViT-B/16
	Slots per camera K	7
	Slot dimension D	256
Token Projection	Hidden dimension	256
	Tokens per timestep	17
Transformer Decoder	Layers	8
	Attention heads	8
	Temporal window T	5 frames
	Total sequence length	85 tokens
Action Chunking	Chunk size H	10 steps
	Action dimension	8
	Continuous dims (joints)	7
	Discrete dim (gripper)	1
Goal Conditioning	2D projection input dim	4
	MLP output dim	256

3.3. Systematic Evaluation Protocol

3.3.1. Unseen Seeds Protocol and Checkpoint Sweeping

As illustrated in Figure 3.2, we use a combination of checkpoint sweeping and held-out simulation seeds. This procedure is designed to prevent in-distribution memorization. Model weights are extracted at regular intervals during training and evaluated offline on a pool of 1,000 held-out seeds (index $\geq 10,000$) that are never seen during training. Each evaluation run samples 200 episodes from this pool. We implement a trivial success filter: episodes where the target is initialized at the goal position and terminates in fewer than 10 steps are discarded. The best checkpoint from the sweep undergoes a stability test consisting of three independent evaluation runs of 300 episodes each. We report the mean SR and standard deviation ($\mu \pm \sigma$).

3.3.2. Automated Kinematic Failure Taxonomy

We develop an automated classification script to diagnose failure modes beyond binary success metrics. The script analyses kinematic logs from each episode (action variance, gripper boolean state, and Euclidean distance to goal) and assigns each failed episode to one of five categories:

- **Idle:** The robot fails to initiate meaningful movement ($\sigma_{\text{action}}^2 < 0.001$).
- **No Grasp:** The robot moves but the `is_grasped` flag remains `False` throughout the episode.
- **Drop:** The robot grasps the object (`is_grasped` transitions to `True`) but releases it before task completion.
- **Near Miss:** The robot grasps and transports the object to within 0.1 m of the target but fails to meet the placement tolerance. This indicates a spatial precision failure in the final placement phase.
- **Unknown:** Episodes that do not match any of the above profiles.

The script operates solely on kinematic logs and is independent of the visual encoder, making it applicable across different simulation environments.

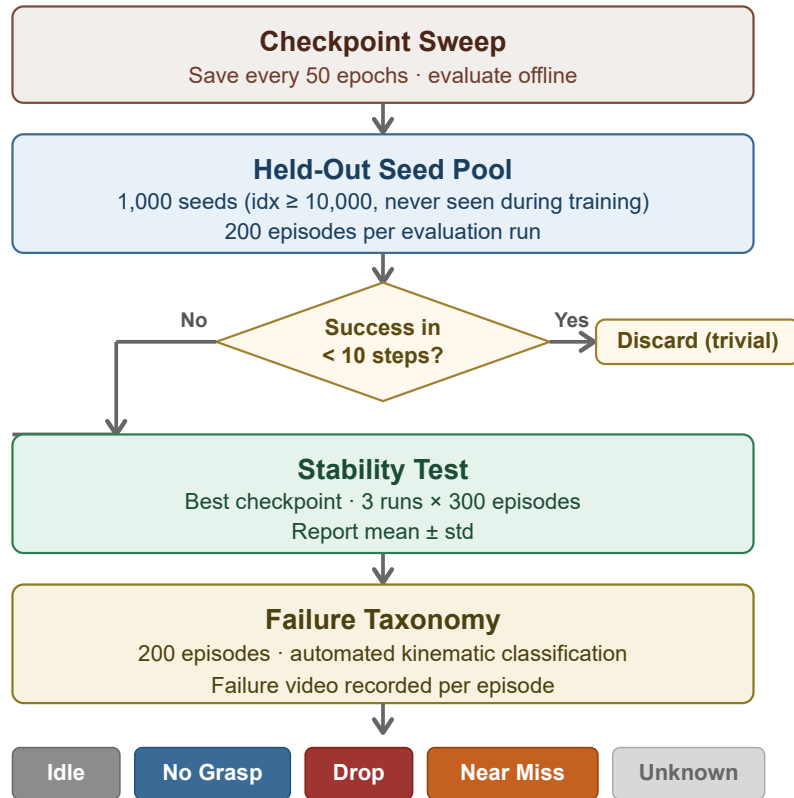


Figure 3.2.: Evaluation pipeline. Checkpoints are saved every 50 epochs and evaluated offline against a pool of 1,000 held-out seeds (index $\geq 10,000$, never seen during training), sampling 200 episodes per run. Episodes terminating in fewer than 10 steps are discarded as trivial successes. The best checkpoint undergoes a stability test of three independent runs of 300 episodes each; we report mean \pm standard deviation. Failed episodes from a separate 200-episode failure analysis run are classified by an automated kinematic script into five categories: Idle, No Grasp, Drop, Near Miss, and Unknown.

4. Experiments and Implementation Details

This chapter documents the experimental process in the order decisions were made. Table 4.1 summarizes the full development path. We begin with the simulation infrastructure and data pipeline (Section 4.1), then describe the early visual encoder explorations (Section 4.1.1). We then detail the incremental policy development (Section 4.2), the spatial grounding interventions (Section 4.3), and finally present the core experimental configurations and task generalization experiments that establish our best-performing model (Section 4.4).

4.1. System Infrastructure and Data Engineering

This section details the infrastructure established for our experiments. We document the simulation environment setup and the data engineering required to handle high-dimensional visual features under hardware constraints.

4.1.1. Encoder Evaluations

Following the literature review, we selected SPOT [6] as the foundational object-centric framework. Additionally, we evaluated THEIA [13], a model that aggregates several VFMs via knowledge distillation. We hypothesized that integrating THEIA’s backbone or Translator heads into the Slot Attention mechanism of SPOT could yield object-centric slots with richer semantic priors.

We initially employed the DINO pretrained model (`dino_vitb16`) [3]. For THEIA, we used its open-sourced Base version, which includes the DeiT backbone and several pre-trained Translator heads (e.g., `dinov2-large` and `clip-vit-large-patch14`). All visual encoders were kept frozen during training. We only updated the Slot Attention

Table 4.1.: Experimental development roadmap. Each cell represents a development stage in the order decisions were made. Bold items indicate the adopted solution at each stage; grey items were explored and discarded. The bottom row shows ablation experiments conducted after the final model was established.

Early Encoder Exploration	Single-Camera Slots	Action Configuration
Backbone-Only Integration Translator Constraint End-to-End Translator Encoding Native DINO-SPOT	Naive Flattening DeepSets Aggregation Per-Slot Weighted Pooling Slot Transformer	Relative Control Absolute Control L1 / Huber / BCE Loss MSE Loss Unified Normalization Decoupled Normalization
Proprioception Injection	Dual-Camera Setup	Multi-View Slot Fusion
Direct Concatenation Learnable Vector Gating Cross-Attention Fusion 9-DoF State 7-DoF Truncation	Table-Level / Front-Low / Wide-Angle Diagonal Side Camera Wrist Hand Camera Train-Eval Shift Fixed	MLP / Mean Pooling / Cross-Attention Camera Embeddings Slot Filtering · Temporal GRU Simple Token Concatenation
Sequence Policy	Spatial Grounding	Final Model
Tanh Action Head ReLU Head T=40 / T=20 T=5 ResNet18 End-to-End Bidirectional Cross-Attention Simple Concatenation	Visual Goal Marker (RGB) 3D Oracle (upper bound) 2D Spatial Projection Time-Weighted Loss U-Shaped Loss Dimension-Decoupled Loss	SPOT · Dual Camera Autoregressive Sequence Policy T=5, H=10, Action Chunking 2D Spatial Goal Projection Decoupled Gripper
Resolution & Calibration	Representation Ablation	StackCube Generalization
Approximate · 128x128 Exact · 128x128 Exact · 224x224	DINO Global [CLS] DINO 4x4 Patches DINO 14x14 Dense (T=1, H=1) SPOT 14 Slots	Hand Camera · no goal Side Camera · 2D Goal

module and the downstream decoder network. THEIA and SPOT require different normalization protocols, so we implemented a preprocessor to apply the correct normalization per encoder type.

To validate this hypothesis, we conducted three sequential explorations on the Pascal VOC dataset:

- **Backbone-Only Integration:** We extracted the `last_hidden_state` of the THEIA backbone. We stripped the [CLS] token and reshaped the remaining 1D sequence into a 2D spatial grid via bilinear interpolation to 14×14 , matching SPOT’s input shape, and applied a linear projection to 768 dimensions.
- **Multimodal Translator Constraint:** To capture high-level semantics, we designed an alignment constraint in the feature space. Because THEIA’s Translator requires a complete spatial sequence of length 197 (1 CLS token + 196 spatial tokens), we duplicated a single isolated slot vector 196 times and used the slot itself as a pseudo-CLS token to construct a sequence of length 197. We computed an MSE consistency loss between this pseudo-sequence passed through the Translator and the target features generated from the original images. This auxiliary loss was weighted at 0.1 and jointly optimized with the image reconstruction loss.
- **End-to-End Translator Encoding:** We directly extracted the final output of THEIA’s `DINOv2-large` Translator. Because this outputs a 1024-dimensional sequence that does not preserve original spatial correspondences, we stripped its [CLS] token, applied 1D adaptive average pooling to compress the sequence to length 196, and applied a linear projection to 768 dimensions.

These explorations showed the difficulty of integrating THEIA into the Slot Attention framework. Interpolated backbone features alone failed to disentangle objects. The auxiliary Translator constraint provided no meaningful learning signal. End-to-end Translator encoding with 1D pooling destroyed the 2D spatial topology that Slot Attention requires, causing representation collapse.

Diagnostic Evaluation. Standard segmentation metrics (Adjusted Rand Index (ARI), mean Intersection over Union (mIoU)) did not fully characterize these failure modes. We developed a diagnostic script that computes four metrics directly from the slot attention maps:

- **Slot Activation:** mean attention value per slot over the spatial grid, indicating how actively each slot is used.

-
- **Slot Overlap:** for slots i and j , computed as $(A_i \cdot A_j) / \min(\sum A_i, \sum A_j)$. Lower values indicate better disentanglement between slots.
 - **Spatial Coherence:** mean absolute difference between horizontally and vertically adjacent pixels in each attention mask. Lower values indicate that each slot forms a more spatially connected region.
 - **Slot Entropy:** Shannon entropy of the normalized attention distribution per slot. Lower values indicate more focused attention.

These metrics confirmed the collapse quantitatively. Applying 1D adaptive pooling to the THEIA Translator features decreased the ARI from the SPOT baseline of 14.9% to 6.4%, and substantially increased slot overlap values. This shows that the 1D pooling operation disrupted the spatial topology required for object disentanglement. THEIA’s Translator is optimized for semantic alignment across modalities rather than for preserving the 2D patch structure that Slot Attention relies on.

We therefore continued with the native DINO-SPOT architecture and migrated our evaluation pipeline to 3D robotic simulation to assess downstream manipulation performance.

4.1.2. Simulation Migration and Determinism

We selected ManiSkill3 [9] as our simulation framework. Table 4.2 compares the key selection criteria against other candidate frameworks.

The Unified Workflow. Because the visual encoders are kept frozen throughout training, feature extraction for any given image is deterministic and needs to be computed only once. We therefore established an offline-to-online workflow: all expert demonstration frames are passed through the frozen encoder once during an offline preprocessing phase, and the resulting embeddings are saved to disk. During training, the policy reads cached embeddings directly without invoking the encoder. Without caching, extracting multi-view SPOT features on-the-fly required 1–2 hours per epoch for the dual-camera configuration. Pre-computing and caching the embeddings reduced this to approximately 1–2 minutes per epoch for the full sequence policy ($T = 5$, $H = 10$, sequence length 85). We then trained the BC policy on these cached features and evaluated performance online.

End-to-end visuomotor methods such as Diffusion Policy [7] and ACT [8] backpropagate gradients through the vision backbone jointly with the policy. The ManiSkill3 benchmark

Table 4.2.: Comparison of candidate simulation frameworks. CortexBench [14] relies on MuJoCo 2.0 and `mujoco-py`, incompatible with Python 3.10. LIBERO [27], CALVIN, and RLBench provide expert demonstrations but share legacy dependency constraints. Navigation-focused frameworks (BEHAVIOR-1K, ManipulaTHOR) support modern dependencies but target indoor navigation rather than tabletop manipulation. ManiSkill3 [9] is the only framework satisfying all requirements for this thesis, offering GPU-parallelized physics and native expert datasets for imitation learning.

Framework	Physics Engine	Task Focus	Python 3.10+	Det. Replay	Multi-view	Expert Demos	GPU Parallel
CortexBench [14]	MuJoCo 2.0	Manipulation	No	Partial	No	No	No
LIBERO [27]	MuJoCo	Manipulation	No	No	No	Yes	No
CALVIN	PyBullet	Manipulation	No	No	Yes	Yes	No
RLBench	CoppeliaSim	Manipulation	No	No	Yes	Yes	No
BEHAVIOR-1K	OmniGibson	Navigation	Yes	Partial	Yes	Yes	Yes
ManipulaTHOR	Unity	Navigation	Yes	No	Yes	No	No
ManiSkill3 [9]	SAPIEN/PhysX	Manipulation	Yes	Yes	Yes	Yes	Yes

reports these baselines on an RTX-4090 (24 GB VRAM) [9], a GPU with more than twice the VRAM of our RTX-2080-Ti (11 GB). Our decoupled offline caching pipeline eliminated the backbone from the training forward pass entirely, making the full sequence policy trainable on consumer-grade hardware.

We evaluated the original expert actions to establish an upper-bound SR. While the dataset labels indicated a 100% success rate, replaying these trajectories locally yielded approximately 10% success. Without exact environment state alignment, floating-point errors accumulated, causing the robotic arm to miss targets. We identified and resolved three technical issues:

- **Issue 1: Horizon Truncation.** The `PickCube-v1` environment hardcoded a 50-step episode limit to optimize RL exploration. Expert trajectories typically complete between steps 69 and 73. We overrode this with `CPUGymWrapper(ignore_terminations=True)`.
- **Issue 2: Physics Backend Alignment.** Expert demonstrations were recorded on the `auto` backend. Replaying on a local `physx_cpu` backend caused physical drift. We specified the `--use-env-states` option to guarantee consistent dynamics.
- **Issue 3: Lexicographical Seed Mismatch.** Python’s `h5py` library reads dataset keys in lexicographical order (e.g., `traj_0`, `traj_1`, `traj_10`, `traj_2`), causing a mismatch between the trajectory index and the environment seed. We

wrote a custom episode loader to parse the exact `episode_seed` from the JSON metadata and sort numerically.

After applying these fixes, the replay pipeline achieved a 100.0% expert success rate across all 1,000 trajectories.

4.1.3. Data Pipeline and Memory Optimization

Preserving the $H \times W$ spatial geometric topology is important for fine-grained manipulation. Global Mean Pooling collapses features into a 1D vector and destroys spatial correspondence. However, precomputing and storing the full spatial token grid for all training frames posed a practical challenge. The THEIA dense features for all 77,976 frames of `PickCube-v1` require approximately 47 GB of disk storage ($77,976 \times 196 \times 768 \times 4$ bytes). Random access into a 47 GB file on disk is slow enough to negate the benefit of offline precomputation. This further motivated our transition to SPOT, which compresses 196 patches into 7 slots, reducing the precomputed storage to approximately 1.1 GB ($77,976 \times 14 \times 256 \times 4$ bytes)—small enough to load chunk-by-chunk into CPU RAM with minimal IO overhead.

Native Resolution Rendering. To avoid spatial quantization errors from upscaling low-resolution inputs, we configured the `ManiSkill3` cameras to render natively at 224×224 pixels. At the previous 128×128 resolution, a single pixel corresponds to a larger physical distance, obscuring millimeter-level spatial differences. High-resolution native rendering aligns with the SPOT encoder’s input requirements and preserves geometric details.

Chunked Data Loading. We implemented a custom `ChunkedDataset`. During offline preprocessing, we serialized the extracted features into multiple chunk files. During training, macro-sequential chunk loading kept only one chunk in RAM at a time, while micro intra-chunk shuffling maintained training stochasticity. Peak RAM usage remained under 1 GB.

Distributional Shift Correction. Initial policy trials converged to 0% SR with high prediction errors. This was traced to overlapping normalization operations between PyTorch Vision transformations and the frozen backbone’s internal processor, which compressed the input tensor range to $[-0.089, 1.043]$ instead of $[-1.0, 1.0]$. We implemented a dynamic

transform routing function to apply the correct normalization protocol per encoder type. Correcting this reduced the action prediction error by approximately 98%.

4.1.4. State and Action Space Engineering

With a reliable simulation and data pipeline in place, we began constructing the policy. The first design decision concerned the action space.

Relative control (`pd_joint_delta_pos`) is common in Sim2Real transfer. However, in ManiSkill3, replaying demonstrations in relative control mode does not support continuous environment state injection (`--use-env-states`). Without strict state-matching, floating-point errors accumulated rapidly during open-loop replay, misaligning the arm and objects. Absolute control (`pd_joint_pos`) supports `--use-env-states` and eliminates this drift. Switching to absolute control raised the SR from 2.0% to 12.0% in our vision-only setup.

Decoupled Gripper Action. During absolute control trials, the gripper oscillated in a half-open position without grasping the target. Analysis of the 8-dimensional action space revealed a variance conflict: the 7 continuous arm joints exhibited low variance ($\sigma \approx 0.05\text{--}0.2$), whereas the binary gripper state exhibited high variance ($\sigma \approx 0.999$). Applying a unified MSE loss caused the network to predict the statistical mean for the gripper, resulting in mechanical oscillation.

We implemented a decoupled strategy. We restricted Z-score normalization to the 7-DoF continuous joints, preserving the gripper targets in their native binary distribution ($[-1.0, 1.0]$). During denormalization, the raw gripper prediction passes through a hard step function (`torch.where(action > 0.0, 1.0, -1.0)`). This eliminated the oscillation without requiring a separate classification head.

Loss function selection. We evaluated four loss functions for action regression: L1, Huber, MSE, and Binary Cross Entropy (BCE). BCE was tested only on the gripper dimension in isolation, as it is not applicable to continuous joint outputs. L1 and Huber losses showed no significant difference in held-out SR compared to MSE and converged more slowly in early training. MSE provided stable training dynamics across all action dimensions and was used in all subsequent experiments.

4.2. Visual Integration and Sequence Policy Architecture

With the simulation infrastructure and action space established, we incrementally built the visual policy. Each subsection introduces a problem encountered in the current configuration and the architectural change made to address it.

4.2.1. Proprioception Injection and Dimensionality

The first visual policy operated on single-camera SPOT features alone. It suffered from spatial drift: the arm moved but failed to converge reliably on the target. Vision alone cannot correct trajectory errors caused by small prediction mistakes. We injected kinematic joint states as proprioceptive feedback to provide explicit positional awareness.

In the single-camera setup, we used a Slot Transformer (a 2-layer, 8-head self-attention encoder) to model geometric relations among the 7 object slots. We ablated three fusion mechanisms for combining the visual representation with the 7-DoF joint state:

- **Direct Concatenation:** The 7 interacted slots were flattened to a 1792-dimensional vector and concatenated with a 128-dimensional proprioceptive embedding from a 2-layer MLP, forming a 1920-dimensional state vector for the action decoder.
- **Learnable Vector Gating:** A linear bottleneck compressed the visual features. A sigmoid-activated gating vector then element-wise balanced the visual and proprioceptive embeddings.
- **Proprio-Visual Cross-Attention:** The 7-DoF joint state was encoded into a 256-dimensional vector via a 2-layer MLP and used as a query attending to the visual slots as keys and values.

Cross-attention provided marginal improvements over the other two, but all three configurations underperformed. Compressing the visual representation to enable cross-attention disrupted the spatial topology and caused overfitting. More importantly, the primary bottleneck was not the fusion mechanism but the absence of explicit goal information, which we address in Section 4.3.

Simplification via Sequence Policy. These fusion experiments revealed that complex modality-specific fusion modules add architectural overhead without resolving the core information bottleneck. When we later adopted the Autoregressive Multimodal Sequence Policy (Section 3.2.2), the proprioceptive state was projected by a 3-layer MLP to the 256-dimensional hidden space and appended as a single token. This straightforward approach yielded stable training dynamics, confirming that causal self-attention handles cross-modal alignment without a dedicated fusion module.

State Truncation. Expanding the proprioceptive input to the full 9-DoF `qpos` array (7 arm joints + 2 gripper finger positions) reduced the SR to 0%. The gripper fingers had very low variance ($\sigma \approx 0.0108$). Applying global Z-score normalization across all 9 dimensions amplified this by approximately $100\times$, corrupting the attention distribution. Truncating the input to the 7 arm joints (`qpos[:7]`) restored training stability. The finger positions are effectively static proxies for the binary open/close state in `PickCube-v1` and carry negligible dynamic information.

4.2.2. Multi-View Integration and Fusion Interventions

With proprioception stabilising the arm trajectory, a new problem emerged: single-camera setups suffer from Z-axis depth ambiguity during the final descent. The arm frequently approached from the wrong angle, particularly when the target was partially occluded by the gripper. We added a second fixed camera to provide complementary geometric coverage.

Camera Configuration Selection. We first tested whether relocating the primary camera to a lower-angle side view could alone resolve the occlusion problem. To quantify representation quality under different viewpoints, we defined a **Temporal Embedding Diversity** metric: we extracted SPOT embeddings for each frame of a complete expert trajectory and computed the mean pairwise L2 distance across all timestep embeddings. A higher value indicates that the encoder responds to changes in the scene; a near-zero value indicates representation collapse.

Moving to the low-angle side view caused this metric to drop by approximately 50% (from 0.549 to 0.279). The frozen SPOT encoder was pre-trained on COCO—natural, human-perspective photographs—and could not discriminate scene states from an unconventional

viewpoint. This showed that the Base Camera must be retained and a second camera added rather than replacing the first.

We evaluated four fixed auxiliary camera positions. Figure 4.1 shows all four configurations and the selection outcome.

We also evaluated an eye-in-hand wrist camera using `stack_cube_v1` as a diagnostic environment. During the final approach, the target cube frequently exited the wrist camera’s narrow Field of View (FOV), causing the SPOT encoder to lose slot tracking immediately. Frozen object-centric encoders require stable, fixed camera positions; the wrist camera was discarded.

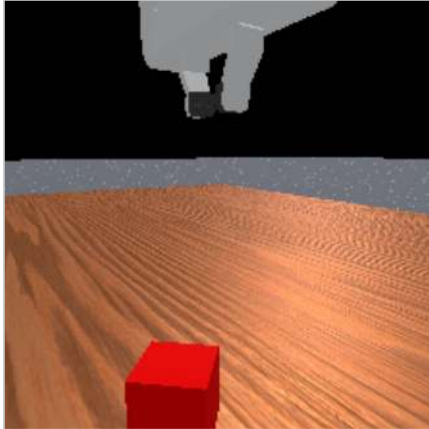
We implemented the `PickCubeDualCam-v1` environment, retaining the default Base Camera and adding the Diagonal Side Camera. To generate synchronised dual-camera training data, we replayed the existing expert demonstrations with `--use-env-states`. This transition introduced two discrepancies between offline training and online evaluation that required resolution before meaningful experiments could proceed. First, the training data used the environment’s internal sensor cameras (90° FOV, grey background), while the online evaluator defaulted to the human render camera (57.3° FOV, black background). The 32.7° discrepancy altered visual target coordinates. Second, the 3D goal marker was placed in a hidden rendering layer during evaluation, making it visible during training but invisible at test time. Aligning the evaluation camera parameters with the training sensors and forcing the goal marker to render raised the training SR from approximately 10% to above 30%.

With the dual-camera setup verified, we investigated how to fuse the 14 resulting slots. This proceeded through three phases.

Phase I: Aggregation Bottleneck. Two cameras produced 14 discrete object slots. We attempted to compress these into a single latent vector using a 2-layer Transformer Encoder ($d_{\text{model}} = 256$, $d_{\text{ff}} = 1024$, 4 heads) to first compute cross-slot interactions, followed by three aggregation strategies:

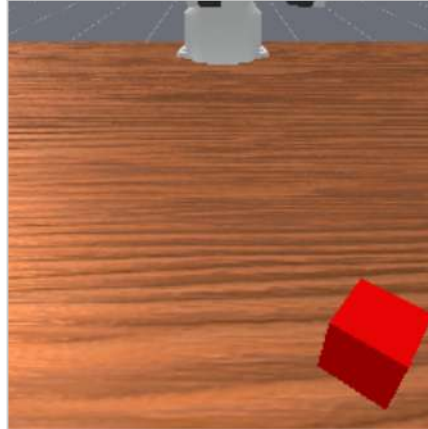
- **MLP Compression:** The 14 interacted slots were flattened into a 3584-dimensional vector and compressed via a 3-layer MLP ($3584 \rightarrow 1024 \rightarrow 512 \rightarrow 256$).
- **Mean Pooling:** Uniform mean pooling across the sequence dimension, compressing the 14-slot output to a single 256-dimensional vector.
- **Query-Based Cross-Attention:** A single learnable query token (1×256) attended to the Transformer output via multi-head cross-attention to aggregate task-relevant information.

Table-level side view



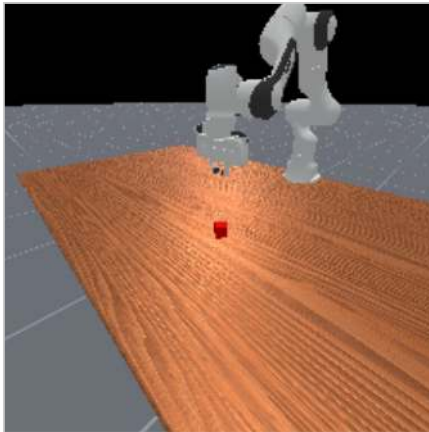
Eliminated: excessive dark background

Front-low view



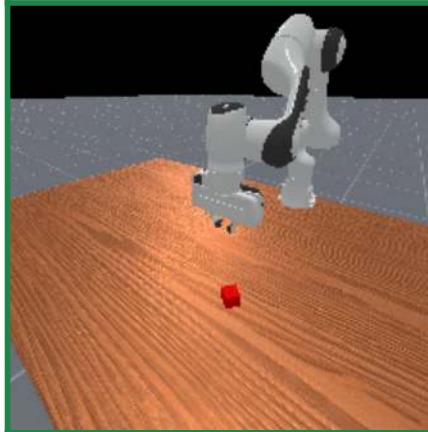
Eliminated: high overlap with base camera

Wide-angle side view



Eliminated: objects too small in frame

Diagonal side view ✓



Selected: complementary Z-axis coverage
eye=[0.4,0.3,0.5] target=[0,0,0.2]

Figure 4.1.: Four auxiliary camera configurations evaluated for the dual-camera setup. Top-left: table-level side view, eliminated due to excessive dark background. Top-right: front-low view, eliminated due to overlap with the Base Camera. Bottom-left: wide-angle side view (eye = $[0.5, 0.5, 0.4]$), eliminated because objects appeared too small. Bottom-right: diagonal side view (eye = $[0.4, 0.3, 0.5]$, target = $[0, 0, 0.2]$), selected for its complementary Z-axis coverage.

All three strategies failed to improve performance. The fundamental problem was not the choice of aggregation method but the compression step itself: forcing 14 independent permutation-invariant slots into a single vector disrupted the set topology of Slot Attention and discarded multi-view spatial resolution.

Phase II: Object-Centric Interventions. To break the performance ceiling without discarding the aggregation approach, we tested three targeted modifications:

1. **Camera Embeddings:** Learnable camera-specific embeddings (`cam_embed_base`, `cam_embed_side`, initialised as $\mathcal{N}(0, 1) \times 0.01$) were added to the slots to inform the model of each slot’s visual origin. Even at this minimal scale, the artificial embeddings disrupted the pre-trained slot distributions.
2. **Task-Aware Slot Filtering:** A learnable query scored the 14 multi-view slots and retained only the top-4 most task-relevant, discarding background noise before compression. Without explicit language conditioning, the filter could not bootstrap meaningful slot selection from manipulation demonstrations alone.
3. **Temporal GRU:** A GRU was applied over the aggregated slots to address dynamic occlusions across frames. Combining multi-head attention with recurrent processing caused gradient instability, producing NaN losses.

These results confirmed that the aggregation paradigm itself was the bottleneck.

Phase III: Sequence-Based Fusion. Recognising that the compression step itself was the problem, we abandoned aggregation and adopted the Autoregressive Multimodal Sequence Policy. The 14 slots are maintained as an uncompressed token sequence, and causal self-attention routes spatial information across views without fixed aggregation. This resolved the aggregation bottleneck and stabilised training.

However, evaluating the policy on both seen and unseen scene initializations revealed a large generalization gap. The Bidirectional Cross-Attention fusion variant achieved approximately 60% SR on training-distribution seeds but only 31% on held-out seeds—a 29 percentage point gap. The simple concatenation baseline also converged at 31% on unseen seeds, indicating that without an explicit spatial anchor, more complex fusion merely overfits to the training distribution. This finding directly motivated the spatial grounding work in Section 4.3.

4.2.3. Autoregressive Multimodal Policy Refinement

With the sequence policy architecture established, we refined two remaining design choices: the action head and the temporal context length.

Action Head. We compared a bounded Tanh head (3-layer MLP with final Tanh activation) against an unbounded ReLU head (3-layer MLP with linear outputs). The Tanh head consistently failed to predict correct action magnitudes. The offline action denormalization pipeline occasionally produced target values slightly outside $[-1, 1]$, causing saturation and vanishing gradients at the boundaries. The unbounded ReLU head resolved this.

Temporal Context and Action Chunking. To provide short-term object permanence and smooth trajectory execution, we combined a sliding temporal window (T) with Action Chunking [7], [8] ($H = 10$). We ablated the context length across $T \in \{5, 20, 40\}$. Because self-attention scales quadratically ($\mathcal{O}(L^2)$), $T = 40$ raised training time to approximately 6.4 minutes per epoch. Tests with $T = 20$ degraded performance, suggesting that excessive context dilutes the immediate spatial cues required for fine motor control. $T = 5$ provided sufficient short-term memory at acceptable cost and was fixed for all final experiments. Quantitative results for this ablation are reported in Section 5.3.

Architectural Validation. To confirm that the frozen SPOT encoder is computationally necessary, we attempted to replace it with a trainable ResNet18 backbone. ResNet18’s Layer 4 produces a 7×7 spatial grid per camera, yielding 98 visual tokens per timestep from both cameras. With $T = 5$, the total sequence length reached approximately 500 tokens. End-to-end backpropagation through both the 11M-parameter ResNet18 encoders and the 8-layer Transformer decoder triggered immediate Out-of-Memory (OOM) errors on our 11 GB hardware. This confirms that dimensionality reduction via a frozen encoder is a practical requirement for sequence-based control on standard hardware.

4.3. Algorithmic Interventions for Spatial Grounding

Failure taxonomy analysis of early dual-camera evaluations (Section 3.3.2) revealed that the dominant failure mode was the Near Miss: the robot grasped and transported the

cube but could not place it within the required spatial tolerance. The sequence policy had learned to execute smooth approach and grasp trajectories but systematically failed at the final placement. This pointed to a specific missing capability: the frozen 2D object-centric slots lacked absolute 3D coordinate information, and the policy had no reliable way to determine the exact goal location. This section documents our investigation into providing explicit spatial information to resolve this bottleneck.

4.3.1. Spatial Grounding via Goal Injection

We hypothesized that the Near Miss failures were caused by the absence of explicit spatial grounding rather than limitations in the policy architecture. In single-camera experiments, we had provided visual guidance by rendering a 3D green spherical goal marker in the training frames. In the dual-camera setup, this visual method failed. Attention map analysis showed that the encoder frequently merged the goal marker and the grasped cube into a single dominant slot, making the goal representation ambiguous and easily overwhelmed during multi-view fusion.

Explicit 3D Oracle. To isolate the spatial grounding bottleneck, we injected the exact 3D world coordinates $([x, y, z])$ from the simulator (`obs['extra']['goal_pos']`). This 3-dimensional vector was encoded by a linear projection to 256 dimensions and inserted as the 16th token in the sequence, directly before the action token. The 3D Oracle serves as a diagnostic upper bound: it establishes the maximum performance when absolute 3D coordinate information is available. Quantitative results are reported in Section 5.1.

2D Spatial Projection. To use spatial information that is derivable from visual observations rather than directly from simulator states, we projected the 3D target coordinates into the 2D pixel planes of both cameras using the simulator-provided intrinsic and extrinsic matrices, following the projection equations in Section 3.2.1. This produced a 4D positional vector $([u_{\text{base}}, v_{\text{base}}, u_{\text{side}}, v_{\text{side}}])$, which was encoded by a 3-layer MLP to a 256-dimensional goal token and appended to the sequence.

In principle, the per-slot attention masks from SPOT could provide goal coordinates without ground-truth, by computing the 2D centroid of the mask corresponding to the target object. We investigated this offline but found it unreliable. The encoder frequently assigned both the goal marker and the cube to the same slot, placing the computed centroid between the two objects. Background slots covering the table produced centroids

with no meaningful spatial correspondence. The policy would also need to learn which of the seven slot centroids corresponds to the goal, adding a learning burden that explicit projection avoids. For these reasons, we use the 3D oracle as a diagnostic upper bound and the GT-projected 2D coordinates as our primary practical approach. A self-supervised variant using mask-derived centroids is left for future work (Section 7.3).

An early implementation used approximate camera parameters. A subsequent run used exact extrinsic and intrinsic matrices. The performance difference between the two quantifies the sensitivity of the method to calibration accuracy.

4.3.2. Algorithmic Interventions for the “Last Millimeter”

Adding the 2D goal token substantially reduced Near Miss failures, but failure taxonomy on this configuration showed that approximately 45% of remaining failures were still Near Misses. The 3D oracle had nearly eliminated them, indicating that some residual placement error persisted with 2D projected coordinates. We hypothesized that the standard MSE loss—which penalizes all timesteps and action dimensions equally—caused the policy to prioritise the easier approach and grasp phases over the precise final placement. To investigate this, we tested three dynamic loss weighting mechanisms based on trajectory progress:

- **Time-Weighted Scheduling:** A linear scaling factor applied to the MSE loss that increases towards the end of the trajectory, with a warm-up phase at epochs 0–100 using uniform weight to first establish grasping behavior.
- **U-Shaped Temporal Weighting:** Dual Gaussian peaks centred at 35% and 85% of episode progress (height 2.0, width 0.15), emphasizing both the grasp and placement phases while down-weighting the transport phase.
- **Dimension-Decoupled Scheduling:** Separate loss weights for the 7-DoF joints and the 1-DoF gripper, with a phase transition at 40% episode progress. Gripper loss weight is 3.0 early and 1.0 late; joint loss weight is 1.0 early and 3.0 late.

All weighting factors are precomputed offline from the episode progress index and applied only during training; inference uses a standard forward pass. Quantitative results are reported in Section 5.4.

4.4. Baseline Comparisons and Task Generalization

This section formalizes the experimental configurations evaluated in Chapter 5.

4.4.1. The Core Object-Centric Pipeline

We established our core methodology using the PickCube-v1 task with 1,000 expert trajectories. All SPOT configurations use the frozen DINO ViT-B/16 backbone, extracting 7 slots per camera view (14 slots total). The policy operates with $T = 5$ and $H = 10$. Models are trained using AdamW with learning rate 10^{-4} and batch size 32.

Baseline: approximate projection, 128×128 . Initial dual-camera experiments used 128×128 native rendering with approximate camera parameters for the 2D goal projection. This is the starting reference for subsequent comparisons.

Exact spatial projection, 128×128 . The same setup using exact extrinsic and intrinsic matrices, isolating the effect of calibration accuracy.

Exact spatial projection, 224×224 . The simulator renders natively at 224×224 pixels. At 128×128 , a single pixel corresponds to a larger physical area, which may limit precision near the goal. All other settings match the exact projection configuration.

These three configurations form a two-factor comparison. The first factor is calibration accuracy: comparing the approximate and exact projection variants at 128×128 isolates the effect of camera parameter precision on placement performance. The second factor is rendering resolution: comparing the exact projection configurations at 128×128 and 224×224 isolates the effect of spatial quantization. At 128×128 , each pixel in the native rendering corresponds to a larger physical distance; upscaling to 224×224 for the SPOT encoder introduces quantization error that may limit precision during the final placement phase. Rendering natively at 224×224 eliminates this upscaling step. Quantitative results for all three configurations are reported in Section 5.1.

4.4.2. Ablation Study: Visual Representations

To determine whether performance gains stem from the object-centric grouping mechanism rather than the underlying DINO features, we designed three baselines using the frozen `dino_vitb16` encoder with the same policy and 2D goal token. Table 4.3 summarizes the computational properties of all configurations.

Table 4.3.: Visual representation configurations. Visual tokens per timestep counts tokens from both cameras combined. Sequence length is computed over T timesteps, including 3 additional tokens per timestep (proprioception, goal, action). The DINO 14×14 dense configuration uses $T = 1$ and $H = 1$ due to memory constraints.

Configuration	Visual tokens/step	Seq. length	T	H	Batch
DINO Global [CLS]	2	25	5	10	32
DINO 4×4 patches	32	175	5	10	32
DINO 14×14 dense	392	395	1	1	4
SPOT 14 slots (proposed)	14	85	5	10	32

Global Features. We extracted the [CLS] token from each camera view, yielding 2 visual tokens per timestep. The total sequence length over $T = 5$ is $(2 + 3) \times 5 = 25$ tokens. This tests manipulation performance using only global scene semantics with no spatial layout information.

Uniform Spatial Patches. We used the dense DINO patch outputs. The full 14×14 grid (196 tokens per camera) gives $(196 \times 2 + 3) \times 5 = 1,975$ tokens over $T = 5$, triggering OOM. A first reduction to 7×7 (49 tokens per camera) gives $(49 \times 2 + 3) \times 5 = 505$ tokens, which also caused OOM. We therefore applied 2D adaptive average pooling to a 4×4 grid (16 tokens per camera), giving $(16 \times 2 + 3) \times 5 = 175$ tokens. This fits within the memory budget while using a comparable number of visual tokens to SPOT (32 vs. 14 per timestep). This configuration evaluates whether uniform spatial sampling can match object-centric grouping under similar token constraints.

Dense Features and Memory Constraints. To evaluate the full 14×14 DINO spatial resolution, we reduced the temporal window to $T = 1$ and action chunking to $H = 1$.

The sequence length per timestep observation is $196 \times 2 + 3 = 395$ tokens. Memory profiling showed that the 8-layer Transformer’s activations and gradients each consumed approximately 3 GB. To fit within our 11 GB VRAM limit, we reduced the batch size from 32 to 4. This tests whether a 14-fold increase in spatial tokens per camera compensates for the loss of temporal history and trajectory chunking.

4.4.3. Task Generalization: The StackCube Challenge

To evaluate whether the learned representations and policy generalize beyond single-object pick-and-place, we introduced the StackCube-v1 task. This requires the robot to grasp a red cube and place it stably on top of a green cube—a two-stage task that demands sequential reasoning about two objects.

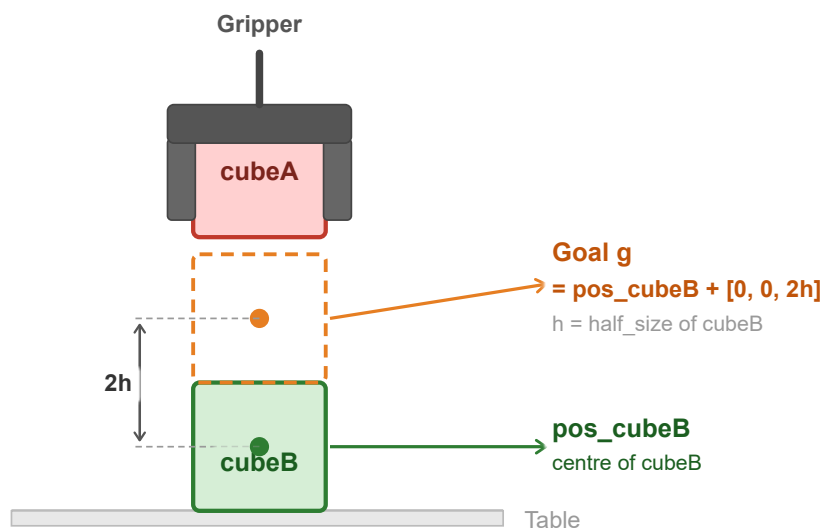


Figure 4.2.: Goal definition for the StackCube-v1 task. The 3D goal coordinate is defined as the top-center of the green cube: $\mathbf{g} = \text{pos}_{\text{cubeB}} + [0, 0, 2 \times \text{half_size}]$. This is the target position that the centre of the red cubeA must reach for the task to succeed.

Native Setup Baseline. The default environment uses a fixed Base Camera and a dynamic hand camera mounted on the wrist. We deployed the Autoregressive Multimodal Sequence Policy ($T = 5$, $H = 10$) without explicit goal conditioning to establish a lower-bound reference. Preliminary evaluation confirmed the FOV limitation of the wrist camera identified in Section 4.2.2: the target cube frequently exited the frame during the downward placement phase, causing the encoder to lose slot tracking.

Aligned Configuration with 2D Goal. To resolve this limitation and enable a fair comparison with the `PickCube-v1` results, we replaced the wrist camera with the same Diagonal Side Camera, creating a `StackCubeDualCam-v1` environment. We defined the 3D goal coordinate as the target placement centre of the red cube: $\text{pos}_{\text{cubeB}} + [0, 0, 2 \times \text{half_size}]$. This was projected to a 2D goal token using approximate camera parameters, with precise extrinsic calibration left for future work.

This aligned configuration establishes a standardised baseline for multi-stage compositional tasks. We conduct a detailed failure mode analysis tracking the distinct rates of *No Grasp*, *Drop*, and *Near Miss* to evaluate whether stable geometries and explicit goal conditioning facilitate task generalization. Figure 4.2 illustrates the goal definition geometrically.

5. Results and Ablation Studies

This chapter presents the quantitative results for the experiments defined in Chapter 4. We first evaluate the impact of explicit spatial grounding on placement precision and report the effect of camera calibration accuracy and rendering resolution (Section 5.1). We then compare object-centric slot representations against global and dense DINO baselines (Section 5.2). Architectural ablations on slot processing and temporal context follow (Section 5.3). We then report the results of loss weighting interventions (Section 5.4), and conclude with task generalization experiments on StackCube-v1 (Section 5.5).

5.1. The Impact of Explicit Spatial Grounding

Hypothesis: The per-slot attention masks of SPOT encode approximate spatial positions of scene entities. However, we hypothesize that this implicit positional encoding is insufficient for millimeter-level placement, and that injecting explicit goal coordinates as a dedicated input token more reliably resolves the Near Miss bottleneck.

5.1.1. Quantitative Success Rate Comparison

We evaluated three dual-camera sequence policies ($T = 5$, $H = 10$) that are identical except for the goal information provided:

- **Pure Visual:** No goal coordinates. The policy relies entirely on the visual slots.
- **2D-Spatial-Proj:** The 3D goal coordinate is projected onto both camera planes using the camera intrinsic and extrinsic matrices, yielding a 4D pixel-space vector injected as a goal token.

- **Explicit-3D-GT:** The ground-truth 3D world coordinate of the goal is injected directly as a goal token. This serves as a diagnostic upper bound.

Table 5.1.: Comparison of spatial grounding configurations on PickCube-v1. All three policies use the same frozen SPOT encoder, dual-camera setup (128×128 rendering), and Autoregressive Multimodal Sequence Policy ($T = 5, H = 10$). SR and mean episode length are reported as mean \pm standard deviation over three independent evaluation runs of 300 held-out episodes each.

Method	Held-Out SR (%)	Mean Episode Length (steps)
Pure Visual	31.0 ± 2.8	313 ± 6
2D-Spatial-Proj	55.0 ± 2.9	247 ± 6
Explicit-3D-GT	71.7 ± 4.1	182 ± 9

Figure 5.4 illustrates representative Near Miss episodes from this configuration. As shown in Table 5.1, adding the 2D projected goal token substantially increased SR over the pure visual baseline. The 3D oracle established the performance upper bound at $71.7 \pm 4.1\%$. Figure 5.1 shows that both goal-conditioned policies converged faster and reached higher rewards during training. The improvement in mean episode length (Table 5.1 and Figure 5.2) shows that explicit goal information also reduces the number of exploratory steps near the target: the 3D oracle policy averaged 182 steps per episode, compared to 313 for the pure visual baseline.

5.1.2. Failure Taxonomy

We applied the automated failure taxonomy (Section 3.3.2) to 200 evaluation episodes per configuration to characterize how explicit grounding changes the failure distribution.

Success counts in Table 5.2 reflect single failure analysis runs of 200 episodes and may differ slightly from the stability test SR values in Table 5.1, which are averaged over three independent runs. As shown in Table 5.2, the Near Miss rate dropped from 31.0% in the pure visual baseline to 2.5% with the 3D oracle. The No Grasp rate remained stable across all three configurations, confirming that spatial grounding specifically addresses the placement bottleneck and does not affect the grasping phase.

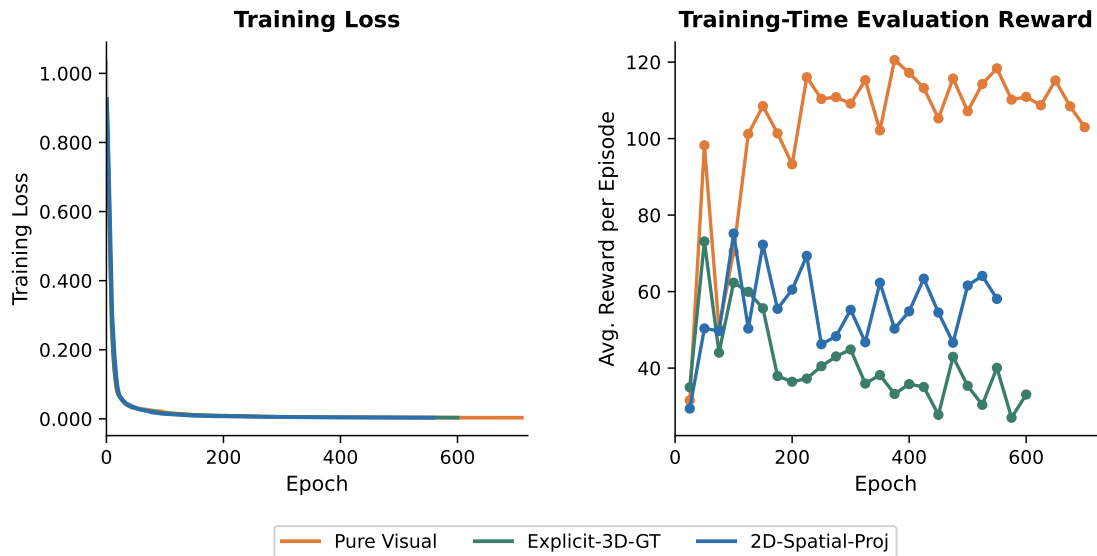


Figure 5.1.: Training dynamics for the three spatial grounding configurations. Left: training loss over epochs, smoothed with a 10-epoch rolling average. Right: average reward per episode during training-time evaluation. Policies with explicit goal conditioning converge to lower loss and higher reward. Held-out SR for each configuration is reported separately in Table 5.1.

Table 5.2.: Failure mode distribution for the three spatial grounding configurations, evaluated over 200 held-out episodes each. Near Miss denotes episodes where the robot grasped and transported the cube to within 0.1 m of the goal but failed the placement tolerance. Drop denotes episodes where the cube was grasped but released before reaching the goal. No Grasp denotes episodes where the cube was never lifted. Unknown denotes episodes with ambiguous kinematic profiles.

Method	Success	Near Miss	Drop	No Grasp	Unknown	Total
Pure Visual	73 (36.5%)	62 (31.0%)	50 (25.0%)	15 (7.5%)	0 (0.0%)	200
2D-Spatial-Proj	107 (53.5%)	42 (21.0%)	33 (16.5%)	15 (7.5%)	3 (1.5%)	200
Explicit-3D-GT	150 (75.0%)	5 (2.5%)	20 (10.0%)	24 (12.0%)	1 (0.5%)	200

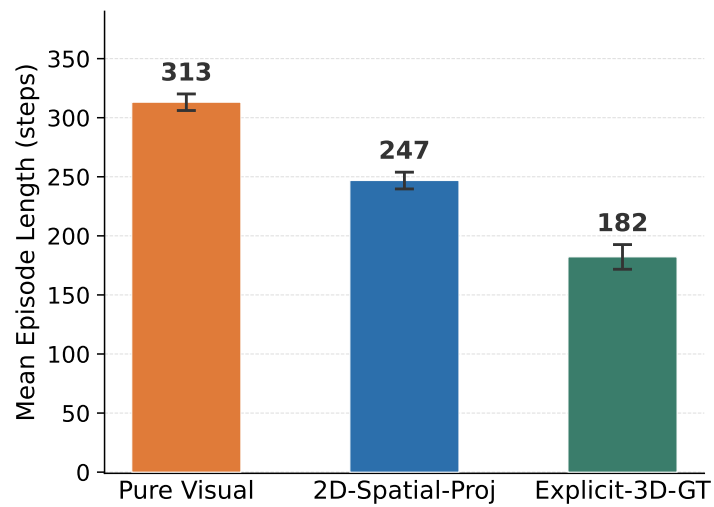


Figure 5.2.: Mean episode length across all evaluation episodes for the three spatial grounding configurations. Error bars show standard deviation over three independent evaluation runs of 300 episodes each. The 3D oracle reduces mean episode length by 42% compared to the pure visual baseline (182 ± 9 vs. 313 ± 6 steps), reflecting fewer exploratory adjustments during final placement.

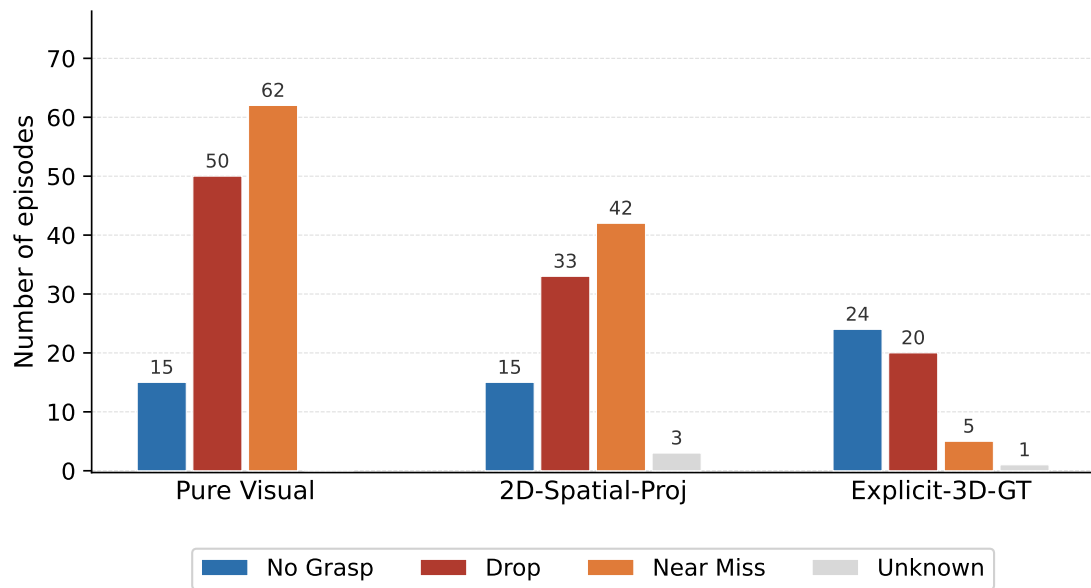


Figure 5.3.: Failure mode counts for the three spatial grounding configurations (200 episodes each, failures only). Near Miss episodes decrease sharply from 62 in the pure visual baseline to 5 with the 3D oracle, confirming that this failure mode is primarily caused by missing spatial information. No Grasp counts remain stable across all three configurations, indicating that spatial grounding specifically resolves the placement bottleneck without affecting the grasping phase. Success counts in this figure reflect single failure analysis runs and may differ slightly from the stability test results in Table 5.1.

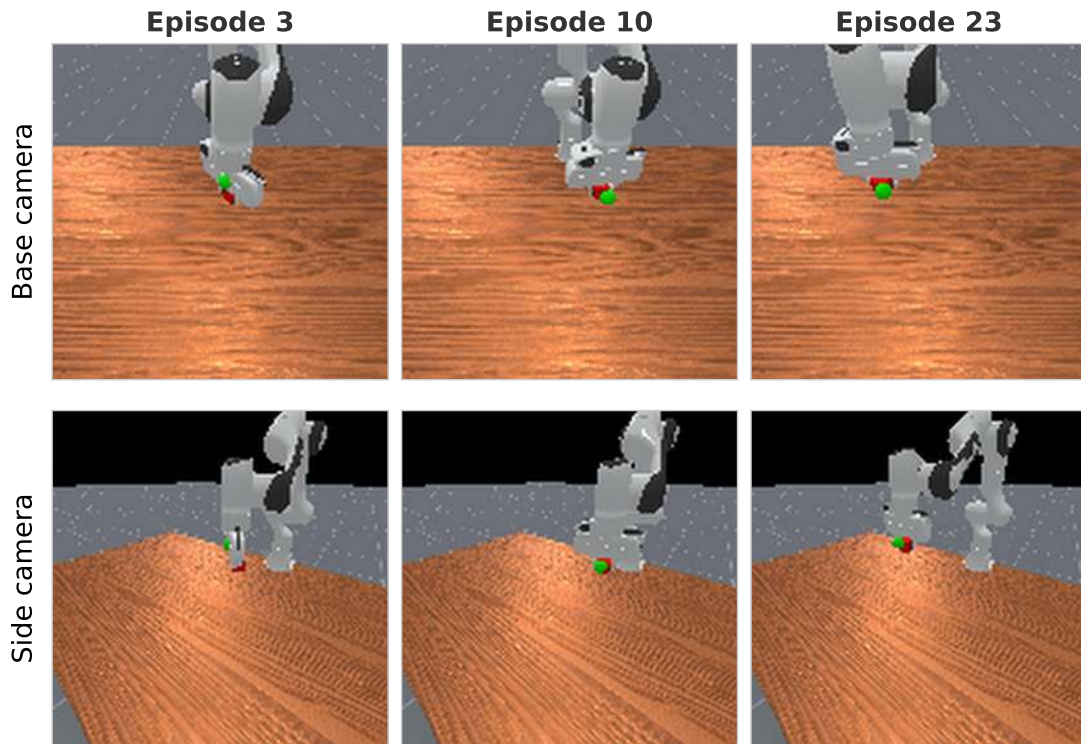


Figure 5.4.: Three representative Near Miss episodes from the 2D-Spatial-Proj configuration (top: Base Camera, bottom: Side Camera). In all three cases, the robot successfully grasps and transports the red cube to the goal vicinity but fails to achieve the required placement tolerance. The green sphere marks the 3D goal position. Near Miss accounts for 21.0% of all episodes in this configuration (Table 5.2).

5.1.3. Effect of Camera Calibration and Rendering Resolution

We evaluated three SPOT configurations that vary in camera parameter accuracy and rendering resolution, holding the policy architecture constant.

Table 5.3.: Effect of camera calibration accuracy and rendering resolution on PickCube-v1 SR. SPOT-Approx-128 uses approximate camera parameters for the 2D goal projection at 128×128 native rendering. SPOT-Exact-128 replaces these with exact extrinsic and intrinsic matrices at the same resolution. SPOT-Exact-224 uses exact parameters and renders natively at 224×224 , eliminating the upscaling step. All other settings are identical ($T = 5$, $H = 10$, 1000 demonstrations). SR is reported as mean \pm standard deviation over three independent evaluation runs of 300 held-out episodes each.

Experiment	Camera params	Resolution	Held-Out SR (%)
SPOT-Approx-128	Approximate	128×128	55.0 ± 2.9
SPOT-Exact-128	Exact	128×128	58.7 ± 3.4
SPOT-Exact-224	Exact	224×224	68.7 ± 4.2

As detailed in Table 5.3, the comparison between SPOT-Approx-128 and SPOT-Exact-128 isolates the effect of calibration accuracy (55.0% vs. 58.7%), a modest but consistent improvement. The further gain from SPOT-Exact-128 to SPOT-Exact-224 (58.7% to 68.7%) suggests that eliminating the $128 \rightarrow 224$ upscaling step substantially improves spatial precision during the final placement phase, where individual pixels correspond to smaller physical distances at higher native resolution. SPOT-Exact-224 achieves $68.7 \pm 4.2\%$ held-out SR, the best result in this thesis and a 37.7 percentage point improvement over the pure visual dual-camera baseline (31.0%, Table 5.1). Note that the representation ablation in Section 5.2 uses SPOT-Approx-128 as the SPOT reference point to ensure a fair comparison with the DINO baselines, which also use 128×128 rendering and approximate camera parameters.

5.2. Baseline Comparison: Object-Centric vs. Global and Dense Features

Hypothesis: Global and dense DINO features mix task-relevant and irrelevant signals across the scene. We hypothesize that object-centric slot representations, by grouping features into discrete entity vectors, provide a cleaner input for the sequence policy and achieve higher manipulation SR under the same computational budget.

5.2.1. Quantitative Comparison

We compared SPOT-Approx-128 against three DINO baselines (Section 4.4.2), using the same 128×128 rendering resolution and approximate camera parameters across all configurations. All configurations use the same Autoregressive Multimodal Sequence Policy and 2D goal token.

Table 5.4.: Comparison of visual representation approaches on PickCube-v1. All configurations use the frozen `dino_vitb16` encoder or SPOT (which also uses a frozen DINO ViT-B/16 backbone), the same Autoregressive Multimodal Sequence Policy, and 2D goal projection. Visual tokens per timestep counts both cameras combined. The DINO 14×14 configuration uses $T = 1$ and $H = 1$ due to memory constraints. Training time per epoch is measured on an RTX 2080 Ti (11 GB); the DINO 14×14 configuration uses a reduced batch size of 4 due to memory constraints. SR is reported as mean \pm std over three evaluation runs of 300 episodes each, except DINO 14×14 which reports the sweep peak (116 epochs trained in 48 hours).

Method	Visual tokens	T	H	min/epoch (batch)	Held-Out SR (%)
DINO Global [CLS]	2	5	10	1:22 (32)	32.6 ± 1.5
DINO 4×4 patches	32	5	10	6:07 (32)	31.7 ± 3.0
DINO 14×14 dense	392	1	1	21:53 (4)	1.0 (peak)
SPOT-Approx-128 (ours)	14	5	10	2:23 (32)	55.0 ± 2.9

As shown in Figure 5.5, the DINO Global baseline removed all spatial layout information, retaining only global scene semantics per camera. The DINO 4×4 baseline preserved a coarse spatial structure with 16 patches per camera, comparable in token count to SPOT’s 7 slots. Despite this similarity, uniform spatial sampling did not match object-centric

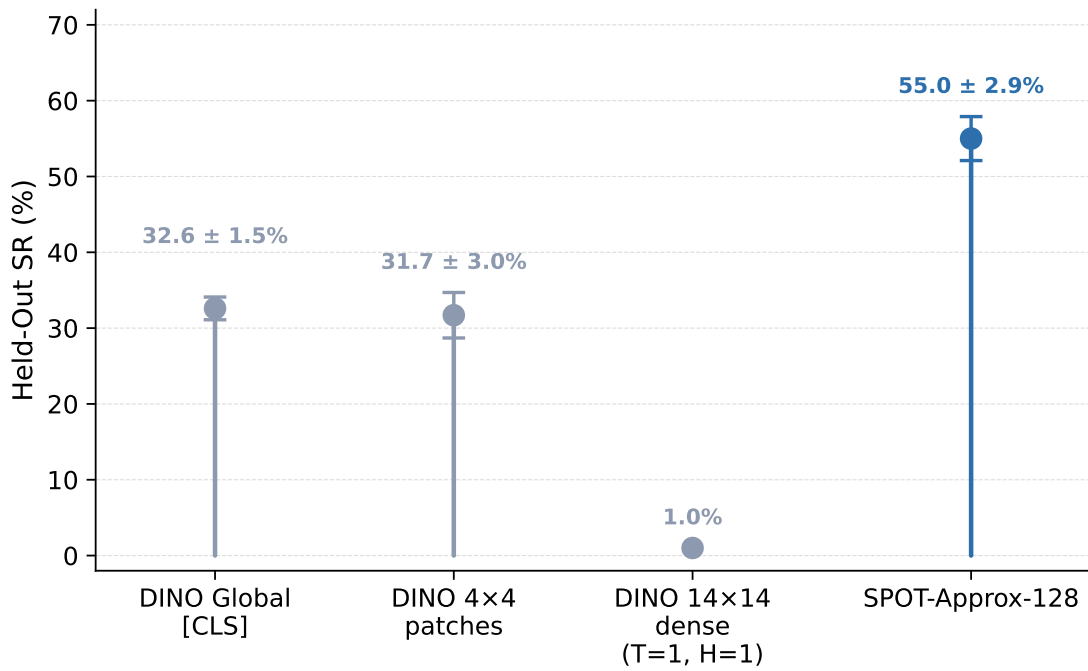


Figure 5.5.: Held-out SR for object-centric (SPOT) and holistic DINO representations on PickCube-v1. All configurations use the same policy and 2D goal conditioning. Error bars show standard deviation over three evaluation runs of 300 episodes each; the DINO 14×14 bar reports the sweep peak only (116 epochs trained in 48 hours).

grouping. The DINO 14×14 dense baseline used the full patch resolution but was limited to single-frame, single-step prediction due to memory constraints. This severely degraded performance, demonstrating that maximizing spatial tokens cannot compensate for the loss of temporal history and action chunking.

5.3. Architecture Ablations

5.3.1. Single-Camera Slot Processing

Hypothesis: Processing object slots with a Transformer encoder that models inter-slot geometric relations outperforms naive aggregation methods such as flattening or symmetric pooling, which either destroy set topology or dilute object-specific information.

These ablations were conducted during the single-camera development phase (Section 4.2.1) to select the optimal slot processing strategy before transitioning to the dual-camera sequence policy.

Table 5.5.: Comparison of single-camera slot processing architectures on PickCube-v1. All configurations use 7 SPOT slots from a single Base Camera, no explicit goal conditioning, and 7-DoF proprioception. Parameter count refers to the slot processing and action decoding modules only, excluding the frozen SPOT encoder. SR is recorded from training-time evaluation runs of 30 episodes and is intended for qualitative comparison only.

Architecture	Param count	Training-time SR (%)
Naive Flattening	~920K	10.0
DeepSets	~250K	6.7
Per-Slot Weighted	~300K	10.0
Slot Transformer	~800K	13.3

As summarized in Table 5.5, the DeepSets approach applied symmetric sum-pooling, treating all slots with equal weight. This caused information dilution, as background slots contributed equally to the aggregated representation as the task-relevant foreground slots. Naive flattening preserved all dimensions but did not model inter-slot relationships. The Slot Transformer applied multi-head self-attention across the 7 slots, allowing it to compute geometric relations between objects before decoding. This achieved the

highest single-camera performance and was used as the basis for subsequent multi-view experiments.

5.3.2. Multi-View Fusion and the Generalization Gap

Hypothesis: Without explicit spatial goal information, complex multi-view fusion mechanisms overfit to the spatial layout of the training distribution and fail to generalize to unseen scene initializations.

Table 5.6.: Generalization gap for multi-view fusion strategies on PickCube-v1, evaluated *without* explicit goal conditioning. Training SR is measured on seen scene initializations during training. Held-out SR is measured over 300 episodes with held-out seeds (index $\geq 10,000$). Training SR values are approximate, recorded during training-time evaluation. Both strategies converge to the same held-out SR, confirming that architectural complexity does not improve generalization without an explicit spatial anchor.

Fusion strategy	Training SR (%)	Held-Out SR (%)	Gap (pp)
Naive concatenation	~ 38.6	31.0	-7.6
Bidirectional cross-attention	~ 60.0	31.0	-29.0

Table 5.6 quantifies this generalization gap: the cross-attention fusion policy appeared better during training, reaching approximately 60% on seen initializations. Under the held-out seed protocol, both strategies converged to 31%, yielding a 29 percentage point generalization gap for cross-attention. Without an explicit spatial anchor to guide the attention queries, the more complex fusion module learned to memorize the training distribution rather than generalize. This result justifies the use of simple token concatenation in the final architecture, and confirms that architectural complexity without spatial grounding does not improve true generalization.

5.3.3. Temporal Context Window (T)

Hypothesis: A short temporal history of $T = 5$ frames is sufficient for short-term object permanence during manipulation. Longer windows increase computational cost without proportional performance gains.

Table 5.7.: Effect of temporal context window size on PickCube-v1, evaluated without explicit goal conditioning. All configurations use SPOT slots and the dual-camera setup. Sequence length is $17 \times T$ tokens. Training time per epoch is measured on an RTX 2080 Ti (11 GB). All runs trained for approximately 24 hours. Peak epoch and reward are recorded at the checkpoint with highest held-out SR. The identical SR across all three configurations indicates that temporal window size does not determine the performance ceiling without spatial goal information.

T	Seq. len.	min/epoch	Epochs	Reward at peak	SR (%)
5	85	1:42	~680	70	6.7
20	340	4:20	~300	45	6.7
40	680	6:04	~150	36	6.7

As shown in Table 5.7, all three configurations reached the same held-out SR of 6.7%, but differed substantially in convergence speed and reward quality. With $T = 5$, the policy reached its peak reward of 70 at epoch 475 after 680 total epochs of training. With $T = 20$, the peak reward dropped to 45 at epoch 225, with only 300 epochs completed in the same 24-hour budget. With $T = 40$, the peak reward fell further to 36 at epoch 120, with only 150 epochs completed.

The identical SR across all three window sizes suggests that temporal context length is not the limiting factor in this setting. Without explicit goal conditioning, all configurations are constrained by the same spatial information deficit, and additional temporal history cannot compensate for the missing goal signal. The reward gap, however, confirms that $T = 5$ makes better use of the available compute budget: it trains approximately $4.5\times$ more epochs than $T = 40$ within the same wall-clock time and achieves higher intermediate reward, indicating better trajectory quality even when the binary success condition is rarely met.

For these reasons, $T = 5$ was selected for all goal-conditioned experiments. The performance gains reported in Section 5.1 therefore reflect the addition of spatial goal information, not the choice of temporal window.

5.3.4. Proprioceptive State Dimensionality

Hypothesis: Including gripper finger positions in the proprioceptive state introduces a variance mismatch that destabilizes normalization and degrades policy performance.

Table 5.8.: Effect of proprioceptive state composition on PickCube-v1. The 9-DoF configuration appends the two gripper finger joint positions ($q_{pos}[7:9]$, range 0.018–0.040 m) to the 7 arm joint angles ($q_{pos}[1:7]$). The finger positions take only two values in practice (open/closed) and have near-zero variance ($\sigma \approx 0.0108$ m). Two normalization strategies were tested for the 9-DoF input. All configurations use the dual-camera Autoregressive Multimodal Sequence Policy without explicit goal conditioning. SR is the mean over three evaluation runs of 300 held-out episodes each, except where noted.

Configuration	Normalization	Held-Out SR (%)
9-DoF (joints + fingers)	Global Z-score (all 9 dims)	0.0
9-DoF (joints + fingers)	Grouped Z-score (joints / fingers)	6.7 (single run)
7-DoF (joints only)	Z-score (7 dims)	31.0 ± 2.8

Table 5.8 demonstrates the impact of state dimensionality: using the full 9-DoF state with global Z-score normalization produced 0.0% SR. The gripper remained permanently closed in all episodes. The finger widths have near-zero variance ($\sigma \approx 0.0108$ m). Global Z-score normalization amplified this by approximately $100\times$, producing large gradient updates that caused the policy to predict closed-gripper actions unconditionally.

Applying grouped Z-score normalization reduced this effect but still yielded only 6.7% SR. The finger positions appear to carry no useful dynamic information for this task regardless of normalization strategy. Truncating the input to the 7 arm joints restored stable training and raised SR to $31.0 \pm 2.8\%$.

5.4. Algorithmic Interventions for the “Last Millimeter”

Hypothesis: The residual Near Miss failures in the 2D goal configuration stem partly from uniform temporal penalization in the MSE loss. Dynamic loss weighting that emphasizes the placement phase will reduce Near Miss rates.

5.4.1. Performance and Stability

We evaluated three heuristic loss weighting mechanisms (Section 4.3.2) against the 2D-Spatial-Proj baseline.

Table 5.9.: Success rates and evaluation variance for heuristic loss weighting interventions on PickCube-v1. All configurations use SPOT slots, dual-camera setup (128×128), approximate camera parameters, and 2D goal conditioning. The baseline is the SPOT-Approx-128 configuration with standard MSE loss. SR is reported as mean \pm standard deviation over three independent evaluation runs of 300 held-out episodes each.

Method	Held-Out SR (%)
Baseline (uniform MSE)	55.0 ± 2.9
Time-Weighted Scheduling	38.9 ± 0.8
U-Shaped Temporal Weighting	34.8 ± 0.6
Dim-Decoupled Scheduling	49.8 ± 3.4

As reported in Table 5.9, all three loss weighting interventions performed below the uniform MSE baseline ($55.0 \pm 2.9\%$). The purely temporal interventions (Time-Weighted and U-Shaped) did not improve overall SR and in some cases shifted failures between categories without net reduction. The Dim-Decoupled approach achieved the closest result ($49.8 \pm 3.4\%$) but with higher variance than the baseline ($\sigma = 3.4\%$ vs. 2.9%), indicating disrupted training stability.

5.4.2. Failure Mode Shifts

As shown in Figure 5.6, all three interventions shifted failures between categories rather than reducing the total. U-Shaped scheduling increased Drop from 33 to 54. Penalizing errors at both the grasp and placement phases appears to disrupt the grasping sub-policy before it is stable. Dim-Decoupled scheduling reduced Near Miss from 42 to 30 but increased No Grasp from 15 to 26, indicating a trade-off between placement precision and grasping reliability.

The results suggest that progress-based loss weights are too coarse for this task. Applying a scalar multiplier to the entire action vector at each phase does not account for the different learning dynamics of individual dimensions. Fully decoupling gripper control at

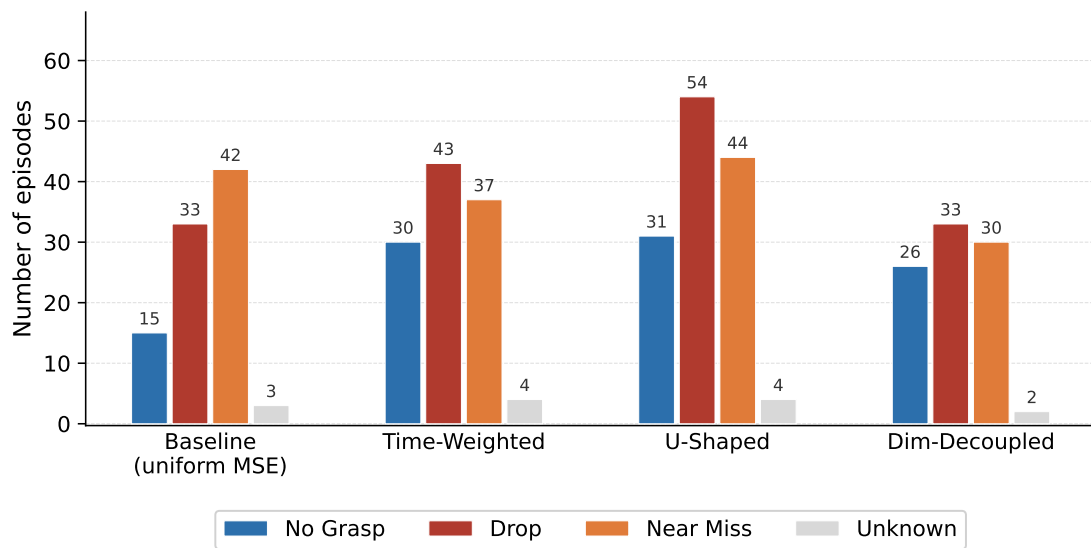


Figure 5.6.: Failure mode counts for loss weighting interventions (200 episodes each, failures only). U-Shaped weighting increases Drop failures from 33 to 54, indicating that over-emphasizing the placement phase destabilizes the grasping sub-policy. Dim-Decoupled scheduling reduces Near Miss from 42 to 30 but does not eliminate it, and increases No Grasp from 15 to 26.

the loss level would be a more targeted fix: replacing MSE with binary cross-entropy for the gripper dimension would treat gripper timing as a classification problem rather than regression, which better matches its discrete physical nature. This is discussed further in Section 7.3.

5.5. Task Generalization: The StackCube Challenge

Hypothesis: Replacing the dynamic wrist camera with a fixed side camera, combined with explicit 2D goal conditioning, resolves the FOV occlusion failures observed in the native setup and improves success rates on the two-object stacking task.

5.5.1. Native Setup vs. Aligned Configuration

Table 5.10.: Comparison of StackCube-v1 configurations. The native setup uses the default Base Camera and dynamic wrist-mounted hand camera without goal conditioning. The aligned setup replaces the wrist camera with the fixed Diagonal Side Camera used in the PickCube-v1 experiments and adds a 2D goal token (target position defined as the top-centre of the green cube). Both configurations use the same Autoregressive Multimodal Sequence Policy ($T = 5, H = 10$) and 1000 expert demonstrations. SR for Stack-Native is reported over a single run of 200 held-out episodes. SR for Stack-Aligned is reported as mean \pm standard deviation over three independent runs of 300 held-out episodes each.

Configuration	Camera setup	Goal conditioning	Held-Out SR (%)
Stack-Native	Base + Hand (dynamic)	None	1.5
Stack-Aligned	Base + Side (fixed)	2D Proj	5.3 \pm 2.2

Stack-Native achieved 1.5% SR. Visual inspection confirmed that the wrist camera frequently lost track of the target cube during the downward placement phase due to FOV occlusion, consistent with the observations in Section 4.2.2. Switching to the fixed side camera and adding 2D goal conditioning raised the SR to 5.3 \pm 2.2%.

5.5.2. Failure Mode Analysis

Table 5.11 and Figure 5.7 detail the failure distribution: the primary bottleneck was grasping: 144 out of 200 episodes (72.0%) failed before the cube was lifted. In *StackCube-v1*, the green target cube occupies part of the scene and may partially occlude the red cube during the approach, which appears to disrupt the grasping sub-policy learned on single-object demonstrations. The 10.0% Near Miss rate mirrors the spatial precision bottleneck observed in *PickCube-v1* (Section 5.1), confirming that this limitation persists in multi-object scenarios.

Table 5.11.: Failure mode distribution for *Stack-Aligned*, evaluated over 200 held-out episodes. No Grasp denotes episodes where the red cube was never lifted. Drop denotes episodes where the cube was grasped but released before reaching the target. Grasp No Stack denotes episodes where the robot lifted the red cube but never transported it to the vicinity of the green cube; this category is specific to the two-object stacking task. Near Miss denotes episodes where the cube reached within 0.1 m of the stacking target but failed the placement condition.

Failure mode	Count	Percentage (%)
Success	12	6.0
Idle	0	0.0
No Grasp	144	72.0
Drop	23	11.5
Grasp No Stack	1	0.5
Near Miss	20	10.0
Unknown	0	0.0
Total	200	100.0

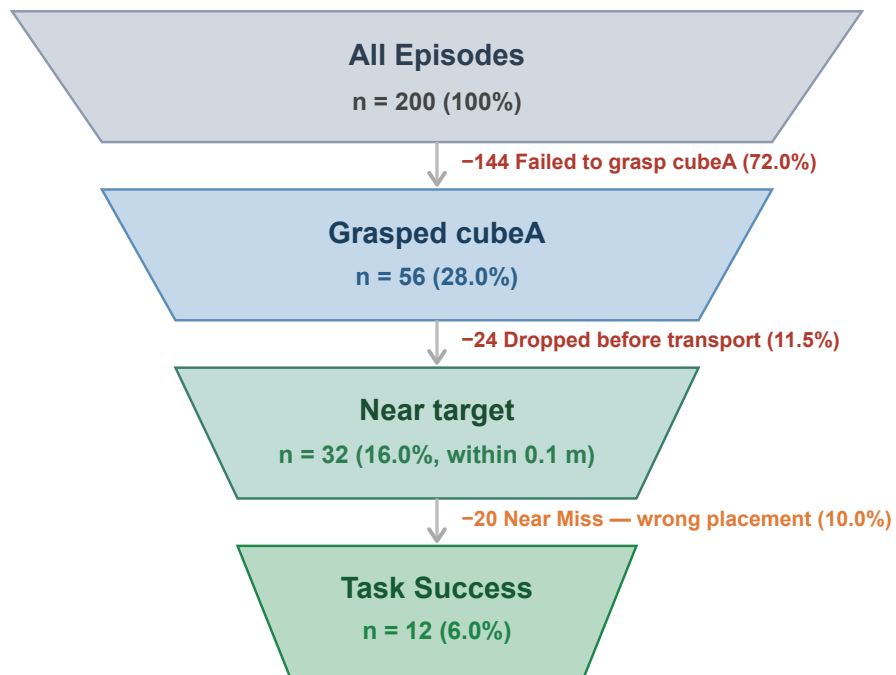


Figure 5.7.: Sub-task conversion funnel for Stack-Aligned (200 held-out episodes). Each stage reports the number of episodes that reached that sub-task milestone. The largest attrition occurs at the grasping stage (200 \rightarrow 56), indicating that multi-object scene complexity disrupts the grasping sub-policy more severely than in the single-object PickCube-v1 task. The dominant failure mode is No Grasp (72% of all episodes).

6. Discussion

The experiments in Chapter 5 establish a clear boundary for unsupervised object-centric representations in continuous control. While Slot Attention excels at isolating semantic entities, it lacks absolute 3D spatial awareness. Explicit spatial grounding bridges this gap. This chapter discusses the mechanisms behind these findings, including the trade-offs between spatial resolution and temporal modeling, the challenges of multi-view slot integration, and the limits of loss-based interventions.

6.1. Representation Limits: Object-Centric Grouping vs. Dense Features

Our ablation studies compared object-centric slots against various configurations of standard VFMs [3], [5], [13]. The results highlight trade-offs between spatial resolution, token efficiency, and temporal modeling.

Semantic isolation over token quantity. Standard dense models extract features at a uniform patch level. The DINO 4×4 patches baseline used 16 uniform spatial patches per view, a token count comparable to our 14-slot setup. Despite this parity, the object-centric policy performed better. Uniform grid sampling assigns tokens to fixed image regions regardless of object boundaries. Slot Attention [2], [6] routes features competitively to isolate semantic entities, filtering out background noise and reducing feature overlap between the end-effector and target object when they physically overlap in the image plane.

The spatial-temporal trade-off. Using full dense spatial grids (196 tokens per view) introduces memory bottlenecks. The DINO 14×14 dense baseline required reducing the temporal window to $T = 1$ and the action chunking horizon to $H = 1$. The resulting low success rate shows that maximizing spatial tokens cannot compensate for the loss of temporal history. Temporal modeling provides motion context, and action chunking ensures trajectory smoothness. Efficient imitation learning requires a representational bottleneck that preserves spatial topology while allowing temporal sequence modeling.

Implicit multimodal alignment. The sequence policy treats four fundamentally different modalities as a unified token stream: object-centric visual slots (semantic), proprioceptive joint angles (kinematic), projected goal coordinates (geometric), and action targets (motor). Projecting each modality to a shared 256-dimensional space before causal self-attention constitutes an implicit multimodal alignment step. The policy learns to route information across these modalities—for example, attending to geometric goal tokens during the placement phase while relying more heavily on visual slots during approach. This differs from language-vision alignment in that all modalities are physically grounded in continuous geometries or low-level motor commands, and the downstream task is motor execution rather than semantic prediction.

6.2. Challenges in Multi-View Slot Integration

Integrating multi-view object slots is more complex than fusing dense visual features. Dense features rely on fixed spatial grids, making concatenation or pooling straightforward. Object slots are permutation-invariant sets with no fixed spatial ordering.

Overfitting in cross-attention fusion. Applying bidirectional cross-attention as a fusion module caused overfitting when explicit spatial goal coordinates were absent. The policy memorized spatial correlations from the training data, resulting in a 29 percentage point generalization gap on unseen seeds. Expert demonstration data lacks hard occlusion edge cases, so the policy relied on dominant viewpoint correlations rather than learning view-invariant representations. Without explicit spatial anchors, more complex fusion architectures do not improve generalization to unseen initializations.

Slot correspondence across views. Object slots from two cameras are independently computed permutation-invariant sets. There is no explicit correspondence between slot i from the base camera and slot j from the side camera. The causal self-attention in the sequence policy must implicitly learn this matching from training data alone. Without an explicit correspondence mechanism, the policy has no guarantee that the same physical object is represented by consistently positioned slots across views.

Data imbalance and single-view reliance. In the expert demonstration data, the target cube is visible from both cameras in the vast majority of frames. Hard occlusion cases—where the cube is only visible from one camera—are rare. As a result, the policy tends to rely on the dominant base camera view and does not learn robust cross-view fallback behavior. When a transient occlusion occurs during evaluation, the policy degrades rather than compensating with the complementary view.

Directions for improved multi-view fusion. Several directions could address these limitations. Explicit cross-view slot matching, using geometric constraints from known camera poses, would provide stable object correspondences across views. Temporally consistent slot tracking, as explored in SAVi++ [18] and temporal contrastive approaches [20], would maintain object identity across frames even under occlusion. Augmenting the training data with single-camera occlusion scenarios would additionally force the policy to learn view-independent representations rather than relying on the dominant viewpoint.

6.3. The “Last Millimeter” Bottleneck and Spatial Grounding

Our failure taxonomy identified the Near Miss as the primary bottleneck for pure visual policies. The model mastered the approach and grasping phases but failed at precise placement. Unsupervised semantic slots isolate entities but do not encode absolute 3D coordinates.

Slot masks as a self-supervised spatial anchor. SPOT produces per-slot attention masks alongside the slot vectors. In principle, the 2D centroid of the mask corresponding to the target object could serve as a goal coordinate without ground-truth input. We investigated this approach offline but found it unreliable. The frozen encoder frequently assigned both the goal marker and the cube to the same slot, placing the computed centroid

between the two objects. Background slots covering the table produced centroids with no spatial meaning. The policy would also need to identify which of the seven centroids corresponds to the goal, adding a matching problem that explicit projection avoids. These limitations motivated the use of GT-projected 2D coordinates as the primary approach and the 3D oracle as a diagnostic upper bound. A self-supervised variant that combines slot segmentation masks with a lightweight 2D object detector to identify the goal slot remains an open direction (Section 7). Appendix E provides representative visualizations of these failure cases.

The insufficiency of loss heuristics. Dynamic loss weighting interventions (time-weighted scheduling, U-shaped weighting, and dimension-decoupled scheduling) did not resolve the placement bottleneck. Re-weighting shifted failures between categories without net reduction, and some configurations introduced training instability. Dynamic loss weighting cannot compensate for missing spatial information in the visual representation.

Spatial quantization error. The improvement from 128×128 to 224×224 native rendering confirms the effect of spatial quantization. At lower resolution, a single pixel spans a larger physical distance. Upscaling does not recover missing geometric detail. As illustrated in Figure 6.1, native high-resolution rendering provides the encoder with finer spatial detail near the placement target.

Robustness to systematic projection error. Early 2D goal injection experiments used approximate camera parameters, introducing a fixed pixel offset into the projected goal coordinates. Despite this fixed offset, the policy still achieved substantial improvement over the pure visual baseline. This indicates that exact coordinate accuracy is not required; a consistent spatial anchor is sufficient.

6.4. Generalization and Temporal Logic in Multi-Stage Tasks

Testing on StackCube-v1 required transitioning from a single-object task to a two-stage sequential task: grasp cube A, place it on cube B.

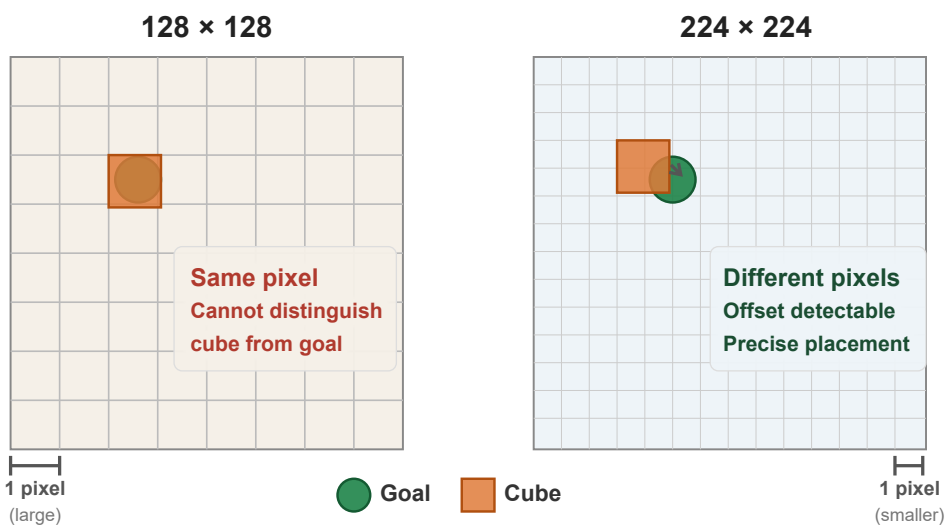


Figure 6.1.: Spatial quantization at 128×128 (left) vs. 224×224 (right) native rendering. Each grid cell represents one pixel. At 128×128 , the cube and goal occupy the same pixel cell, making the spatial offset imperceptible to the encoder. At 224×224 , finer pixel boundaries allow the offset to be detected, enabling more precise placement.

Visual tracking and dynamic occlusion. The eye-in-hand wrist camera caused the target to exit the field of view during the final descent. Because the frozen encoder processes frames independently, the encoder lost tracking of the target slot immediately. Fixed multi-camera setups or recurrent tracking mechanisms are needed to maintain object permanence under dynamic viewpoints.

Sub-task analysis in the aligned dual-camera configuration revealed two bottlenecks. First, the grasp failure rate increased compared to `PickCube-v1`, confirming that visual interference from the second cube complicates target acquisition. Second, the Near Miss failure in `StackCube-v1` manifested differently from the `PickCube` case: the gripper reached the target destination above the green cube but did not release cube A. The policy learned the spatial trajectory but failed to execute the gripper release at the correct sequence step. Multi-stage tasks require precise temporal execution that remains a challenge for autoregressive visual policies.

6.5. Evaluating Representations: Process Metrics vs. Binary Success

Standard imitation learning benchmarks use a binary success metric [9]. Our evaluations showed that this metric obscures the partial capabilities of visual representations.

In the pure visual baseline, the policy frequently achieved high episodic rewards despite a low binary success rate. Kinematic logs confirmed that the policy executed the approach, grasp, and transport phases successfully. The failure occurred only during the final placement. A binary metric treats a complete failure (no grasp) and a precision failure (near miss) identically, hiding this distinction.

Evaluating visual policies therefore requires decoupled, multi-stage metrics. Separating grasp rate from final placement distance accurately isolates where a representation model succeeds and where it fails. This connects to ongoing work in reward-based policy learning. Model-based approaches such as COBRA [35] and SOLD [32] learn object-centric world models that predict the effect of actions on individual slots, enabling dense reward signals that reflect sub-task progress rather than binary task completion. Integrating structured reward models with a sequence-based policy is a natural direction to close the gap between semantic understanding and geometric precision.

This observation also motivates decoupling the gripper at the loss level: replacing MSE with binary cross-entropy for the gripper dimension would better match its discrete physical semantics and provide a more informative training signal for the grasping sub-task.

7. Conclusion and Future Work

7.1. Conclusion

This thesis evaluated the integration of pre-trained object-centric representations into continuous robotic manipulation policies. Rather than proposing a novel visual architecture, we focused on bridging discrete semantic visual features and physical kinematic execution. Based on evaluations in the ManiSkill3 [9] environment, we draw the following conclusions.

Object-centric representations outperform global and dense features. Our ablation study showed that the SPOT encoder, which groups visual features into 7 discrete entity slots per camera view, outperformed a DINO global [CLS] baseline ($32.6 \pm 1.5\%$ SR) and a spatially-pooled 4×4 patch baseline ($31.7 \pm 3.0\%$) under matched token counts and identical policy architecture. SPOT-Approx-128 achieved $55.0 \pm 2.9\%$ SR under the same conditions. The dense DINO 14×14 configuration confirmed that maximizing spatial token count cannot compensate for the loss of temporal history and action chunking when memory constraints force a reduction to $T = 1$ and $H = 1$.

Explicit spatial grounding resolves the Near Miss bottleneck. Injecting 2D-projected goal coordinates increased the SR from $31.0 \pm 2.8\%$ (pure visual baseline) to $55.0 \pm 2.9\%$ and reduced the Near Miss failure rate from 31.0% to 21.0%. The 3D oracle established the performance upper bound at $71.7 \pm 4.1\%$ and nearly eliminated Near Miss failures (2.5%). Heuristic loss weighting interventions all performed below the uniform MSE baseline, confirming that dynamic loss reweighting cannot substitute for missing spatial information.

Native resolution rendering improves placement precision. Native 224×224 rendering improved SR from $55.0 \pm 2.9\%$ (SPOT-Approx-128) to $68.7 \pm 4.2\%$ (SPOT-Exact-224), a 13.7 percentage point gain. Exact camera calibration provided a consistent but modest additional improvement ($55.0 \pm 2.9\%$ to $58.7 \pm 3.4\%$). SPOT-Exact-224 represents the best result achieved in this thesis, a 37.7 percentage point improvement over the pure visual dual-camera baseline (31.0%), obtained on a single RTX 2080 Ti without end-to-end visual fine-tuning.

Simple fusion generalizes better than cross-attention. Cross-attention fusion achieved approximately 60% SR on seen initializations but only 31% on held-out seeds, a 29 percentage point generalization gap. Simple token concatenation matched this held-out performance without overfitting. More complex fusion did not improve generalization without an explicit spatial anchor.

Heterogeneous action dimensions require decoupled treatment. Including gripper finger positions in the proprioceptive state reduced SR to 0.0% due to variance amplification under Z-score normalization. Truncating to 7-DoF arm joints and applying a hard step function to the gripper output restored stable training.

Structured failure analysis separates independent bottlenecks. The kinematic failure taxonomy revealed a clear separation between grasping failures and placement precision failures. Spatial grounding resolved the placement deficit without affecting the grasping phase. This provided a clearer picture of where each representation succeeded and where it failed than binary success rates alone.

7.2. Limitations

Several limitations restrict the scope of these findings.

Our evaluations used only two manipulation tasks in simulation, both limited to basic geometric shapes (cubic blocks) on a flat tabletop. Whether the conclusions generalize to tasks involving different object geometries, cluttered scenes, or articulated objects remains untested.

The policy did not generalize well to the `StackCube-v1` task. The primary bottleneck was grasping, with 72.0% of episodes failing to lift the red cube. Visual inspection indicated that during the downward approach, the robotic arm or gripper occluded the target cube from the fixed side camera, causing the frozen encoder to lose slot tracking. This is consistent with the single-camera occlusion vulnerability identified in Section 4.2.2: without a view that maintains unobstructed coverage during close-range manipulation, the policy cannot recover from momentary slot collapse.

Spatial grounding relied on ground-truth 3D goal coordinates extracted from the simulator. In the 2D projection variant, these are transformed into pixel coordinates via exact camera matrices, which still constitutes privileged information. Real deployments would require replacing this with a visual goal detection method.

Hardware memory constraints (11 GB VRAM) forced the sequence policy to use a short temporal window ($T = 5$, 0.25 s at 20 Hz) without recurrent hidden states. This limits the policy’s ability to handle longer occlusions or multi-stage tasks requiring extended temporal reasoning.

All experiments were conducted in simulation. Real-world deployment introduces sensor noise, motion blur, lighting variation, and camera calibration errors that are absent from our dataset. The generalization of these findings to physical robots has not been validated. This includes the open question of whether the performance gains from native 224×224 rendering translate to real camera setups, where spatial quantization and sensor resolution are determined by hardware rather than simulation parameters.

7.3. Future Directions

Based on our empirical findings and failure analyses, we outline six directions for future work.

- **Strict gripper decoupling.** Our architecture bypassed Z-score normalization for the gripper dimension and applied a hard step function at inference. However, the gripper action was still subject to temporal smoothing via Action Chunking and penalized with a unified MSE loss during training. Smoothing a discrete binary action produces hesitant, half-open gripper states that cause drops. A dedicated classification head trained with binary cross-entropy would treat gripper control as a discrete decision rather than a regression target, and decouple it entirely from continuous joint optimization.

-
- **Temporal consistency in the visual encoder.** The current pipeline relies on the policy’s $T = 5$ temporal window to smooth noisy observations. When the frozen encoder fails to assign a slot to the target object in a given frame, the policy receives corrupted input with no recovery mechanism. Recurrent slot updating, as explored in SAVi++ [18] and temporal contrastive approaches [20], would allow the encoder to maintain object identity across frames and handle transient occlusions at the representation level rather than relying on the policy to average over them.
 - **Multi-view slot correspondence.** Object slots from two cameras are independently computed permutation-invariant sets. The sequence policy must implicitly learn which base-camera slot corresponds to which side-camera slot from training data alone. Explicit correspondence mechanisms using known relative camera poses—such as Hungarian matching or optimal transport assignment over slot features—would provide stable cross-view object alignment without requiring pixel-level correspondences.
 - **Self-supervised spatial grounding via slot masks.** SPOT generates per-slot attention masks alongside the slot vectors. In principle, the 2D centroid of the mask corresponding to the target object could provide goal coordinates without ground-truth input. Our offline analysis showed this is unreliable with the current frozen encoder, because the goal marker and cube are frequently assigned to the same slot. A lightweight 2D object detector trained to identify the goal slot from the mask output would resolve this matching ambiguity and replace the GT-projected coordinates with a fully self-supervised spatial anchor.
 - **Slot count and configuration.** The number of slots per camera ($K = 7$) was fixed throughout all experiments. Ablating slot count and exploring task-adaptive slot selection remains future work. The sequence policy treats slots as tokens and supports arbitrary K within memory constraints, so this is a natural next step.
 - **Sim-to-real transfer and broader task evaluation.** The ultimate validation of this architecture lies in physical deployment. Future work should transfer the pipeline to a physical Franka Panda manipulator, addressing real-world sensor noise and inference latency constraints. Evaluation on a broader set of tasks—including different object geometries, multi-object scenes, and visual distribution shifts such as lighting and texture changes—would establish whether the advantages of object-centric representations observed in simulation extend to the generalization conditions described in the related literature [1].

Bibliography

- [1] A. Chapin, B. Machado, E. Dellandréa, and L. Chen, *Spotlighting task-relevant features: Object-centric representations for better generalization in robotic manipulation*, 2026. arXiv: 2601.21416.
- [2] F. Locatello et al., “Object-centric learning with slot attention”, in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 11 525–11 538.
- [3] M. Caron et al., “Emerging properties in self-supervised vision transformers”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 9650–9660.
- [4] M. Seitzer et al., “Bridging the gap to real-world object-centric learning”, in *International Conference on Learning Representations*, 2023.
- [5] M. Oquab et al., “DINOv2: Learning robust visual features without supervision”, *Transactions on Machine Learning Research*, 2024, ISSN: 2835-8856.
- [6] I. Kakogeorgiou, S. Gidaris, K. Karantzalos, and N. Komodakis, “SPOT: Self-training with patch-order permutation for object-centric learning with autoregressive transformers”, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2024, pp. 22 776–22 786. DOI: 10.1109/CVPR52733.2024.02149
- [7] C. Chi et al., “Diffusion policy: Visuomotor policy learning via action diffusion”, in *Robotics: Science and Systems XIX*, Robotics: Science and Systems Foundation, 2023. DOI: 10.15607/RSS.2023.XIX.026
- [8] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, “Learning fine-grained bimanual manipulation with low-cost hardware”, in *Robotics: Science and Systems XIX*, Robotics: Science and Systems Foundation, 2023. DOI: 10.15607/RSS.2023.XIX.016
- [9] S. Tao et al., *ManiSkill3: GPU parallelized robotics simulation and rendering for generalizable embodied AI*, 2025. arXiv: 2410.00425.

-
-
- [10] S. Nair, A. Rajeswaran, V. Kumar, C. Finn, and A. Gupta, “R3M: A universal visual representation for robot manipulation”, in *Conference on Robot Learning (CoRL)*, 2022.
- [11] Y. J. Ma, S. Sodhani, D. Jayaraman, O. Bastani, V. Kumar, and A. Zhang, “VIP: Towards universal visual reward and representation via value-implicit pre-training”, in *International Conference on Learning Representations*, 2023.
- [12] N. Di Palo and E. Johns, “DINOBot: Robot manipulation via retrieval and alignment with vision foundation models”, in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2024, pp. 2798–2805. DOI: 10.1109/ICRA57147.2024.10610923
- [13] J. Shang et al., “Theia: Distilling diverse vision foundation models for robot learning”, in *Conference on Robot Learning (CoRL)*, 2024.
- [14] A. Majumdar et al., *Where are we in the search for an artificial visual cortex for embodied intelligence?*, 2024. arXiv: 2303.18240.
- [15] A. Brohan et al., “RT-2: Vision-language-action models transfer web knowledge to robotic control”, in *Conference on Robot Learning (CoRL)*, 2023.
- [16] M. J. Kim et al., *OpenVLA: An open-source vision-language-action model*, 2024. arXiv: 2406.09246.
- [17] T. Kipf et al., “Conditional object-centric learning from video”, in *International Conference on Learning Representations*, 2022.
- [18] G. F. Elsayed, A. Mahendran, S. van Steenkiste, K. Greff, M. C. Mozer, and T. Kipf, “SAVi++: Towards end-to-end object-centric learning from real-world videos”, in *Advances in Neural Information Processing Systems*, vol. 35, 2022.
- [19] Z. Wu, J. Hu, W. Lu, I. Gilitschenski, and A. Garg, “SlotDiffusion: Object-centric generative modeling with diffusion models”, presented at the Thirty-seventh Conference on Neural Information Processing Systems, Nov. 2, 2023. [Online]. Available: <https://openreview.net/forum?id=ETk6cfs3vk>
- [20] A. Manasyan, M. Seitzer, F. Radovic, G. Martius, and A. Zadaianchuk, *Temporally consistent object-centric learning by contrasting slots*, 2025. arXiv: 2412.14295.
- [21] A. Kori, F. Toni, and B. Glocker, “Multi-view object-centric learning with identifiable representations”, Oct. 4, 2024. [Online]. Available: <https://openreview.net/forum?id=rqBc4WnvUP>
- [22] J. Wang, M. Chen, N. Karaev, A. Vedaldi, C. Rupprecht, and D. Novotny, *VGGT: Visual geometry grounded transformer*, 2025. arXiv: 2503.11651.

-
-
- [23] L. Yang et al., *Depth anything v2*, 2024. arXiv: 2406.09414.
- [24] S. Haldar, Z. Peng, and L. Pinto, *BAKU: An efficient transformer for multi-task policy learning*, 2024. arXiv: 2406.07539.
- [25] A. Kirillov et al., “Segment anything”, in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.
- [26] T. Yu et al., “Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning”, in *Conference on Robot Learning (CoRL)*, 2019.
- [27] B. Liu et al., “LIBERO: Benchmarking knowledge transfer for lifelong robot learning”, in *Advances in Neural Information Processing Systems*, vol. 36, 2023.
- [28] G. Jiang, Y. Sun, T. Huang, H. Li, Y. Liang, and H. Xu, *Robots pre-train robots: Manipulation-centric robotic representation from large-scale robot datasets*, 2024. arXiv: 2410.22325.
- [29] N. Heravi et al., *Visuomotor control in multi-object scenes using object-aware representations*, 2023. arXiv: 2205.06333.
- [30] D. Haramati, T. Daniel, and A. Tamar, *Entity-centric reinforcement learning for object manipulation from pixels*, 2024. arXiv: 2404.01220.
- [31] T. Hanyu et al., *SlotVLA: Towards modeling of object-relation representations in robotic manipulation*, 2025. arXiv: 2511.06754.
- [32] M. Mosbach, J. N. Ewertz, A. Villar-Corrales, and S. Behnke, *SOLD: Slot object-centric latent dynamics models for relational manipulation learning from pixels*, 2025. arXiv: 2410.08822.
- [33] M. S. M. Sajjadi et al., *Object scene representation transformer*, 2022. arXiv: 2206.06922.
- [34] I. Radosavovic, T. Xiao, S. James, P. Abbeel, J. Malik, and T. Darrell, *Real-world robot learning with masked visual pre-training*, 2022. arXiv: 2210.03109.
- [35] N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess, and A. Lerchner, *COBRA: Data-efficient model-based RL through unsupervised object discovery and curiosity-driven exploration*, 2019. arXiv: 1905.09275.

A. Dual-Camera Configuration and Projection Parameters

Both cameras use a 90° horizontal FOV and share the same intrinsic matrix format. The focal length is $f = W/2$, where W is the image width. The principal point is at $(c_x, c_y) = (W/2, W/2)$.

Table A.1.: Camera intrinsic parameters for the two rendering resolutions used in this thesis. All experiments use square images ($W = H$).

Resolution	$f_x = f_y$	$c_x = c_y$	Used in
128×128	64.0	64.0	SPOT-Approx-128
224×224	112.0	112.0	SPOT-Exact-224

The camera positions are defined in the ManiSkill3 scene using `look_at(eye, target)` calls. Both cameras use $FOV = \pi/2$ and near/far clip planes of 0.01 m and 100 m respectively.

Table A.2.: Camera position parameters for the dual-camera setup. `eye` is the camera origin in world coordinates (metres). `target` is the point the camera looks at.

Camera	eye (x, y, z)	target (x, y, z)
Base Camera	(0.30, 0.00, 0.60)	(-0.10, 0.00, 0.10)
Side Camera	(0.40, 0.30, 0.50)	(0.00, 0.00, 0.20)

The extrinsic rotation matrices R and translation vectors t (world-to-camera transform) used in the 2D goal projection are:

$$R_{\text{base}} = \begin{bmatrix} 0.0000 & 1.0000 & 0.0000 \\ 0.7809 & 0.0000 & -0.6247 \\ -0.6247 & 0.0000 & -0.7809 \end{bmatrix}, \quad t_{\text{base}} = \begin{bmatrix} 0.0000 \\ 0.1406 \\ 0.6559 \end{bmatrix} \quad (\text{A.1})$$

$$R_{\text{side}} = \begin{bmatrix} -0.6000 & 0.8000 & 0.0000 \\ 0.4116 & 0.3087 & -0.8575 \\ -0.6860 & -0.5145 & -0.5145 \end{bmatrix}, \quad t_{\text{side}} = \begin{bmatrix} 0.0000 \\ 0.1715 \\ 0.6860 \end{bmatrix} \quad (\text{A.2})$$

These matrices are derived from the `look_at` calls at runtime by ManiSkill3 and extracted directly from the simulator at preprocessing time. The 2D projection formula follows Section 3.2.1.

B. Offline Feature Caching Pipeline

B.1. Design Rationale

The frozen SPOT encoder is deterministic: the same input image always produces the same slot embeddings. There is no need to run it during training. We precompute embeddings for all expert demonstration frames once and write them to disk. During training, the policy reads cached embeddings directly. This reduced per-epoch time from 1–2 hours to approximately 1–2 minutes on our hardware (RTX 2080 Ti, 11 GB VRAM).

Random access into a large precomputed file on disk is slow enough to negate the benefit of offline precomputation. We therefore use a chunked layout: embeddings are written to fixed-size `.pt` files during preprocessing. The training dataloader loads one chunk at a time, keeping peak CPU RAM usage under 1 GB.

B.2. Chunked Dataset Format

Each chunk file is a `.pt` archive containing up to 5,000 temporal training samples. For SPOT-Exact-224 (224×224 , 1,000 expert trajectories), the preprocessed dataset spans 13 chunk files, each approximately 345 MB. Total storage is approximately 4.5 GB.

Each sample in a chunk stores:

- **Embeddings:** shape $(T, 2, 7, 256)$ — the $T = 5$ frame window from both cameras.
- **Actions:** shape $(H, 8)$ — the 10-step action chunk, unnormalized.
- **Goal 2D:** shape $(4,)$ — the projected goal pixel coordinates $[u_{\text{base}}, v_{\text{base}}, u_{\text{side}}, v_{\text{side}}]$, normalized to $[0, 1]$.
- **Joint positions:** shape $(T, 7)$ — the 7-DoF proprioceptive state history.

B.3. Loading Strategy

Standard `DataLoader (shuffle=True)` breaks chunk locality: loading a random sample from chunk 7 while chunk 2 is in memory forces an immediate disk read. With 13 chunks, this caused 93 s per batch in early trials.

We replaced this with a chunk-aware iterator (`iter_shuffled_batches`). At the start of each epoch, the chunk order is shuffled. Samples within each chunk are then shuffled internally. This gives epoch-level randomization with only $\mathcal{O}(\text{num_chunks})$ disk reads per epoch. Peak RAM usage remained under 1 GB throughout training.

Algorithm 1 summarizes the full preprocessing procedure.

Algorithm 1 Offline Feature Caching

Require: Expert trajectories \mathcal{D} , frozen encoder f_θ , chunk size $C = 5000$, temporal window $T = 5$, action horizon $H = 10$

Ensure: Chunk files on disk, action normalization statistics

- 1: Compute joint mean μ and std σ over all actions in \mathcal{D} ▷ One pass over raw data
 - 2: Initialize empty buffers for embeddings, actions, goals, joints
 - 3: **for** each trajectory $\tau \in \mathcal{D}$ **do**
 - 4: Encode all frames with f_θ ▷ Batched, no grad
 - 5: Extract 2D goal projection from camera matrices
 - 6: **for** each valid timestep t in τ **do**
 - 7: Append (T -frame window, H -step chunk, goal, qpos) to buffers
 - 8: **if** buffer size = C **then**
 - 9: Save chunk to disk; clear buffers
 - 10: **end if**
 - 11: **end for**
 - 12: **end for**
 - 13: Save remaining buffer as final chunk
 - 14: Save (μ, σ) and total sample count to metadata file
-

C. Action Space Distribution Statistics

Table C.1 lists the per-dimension mean and standard deviation of the 8-dimensional action space, computed over all 77,634 raw frames in the `PickCube-v1` expert dataset. These statistics are used for Z-score normalization of the 7 joint dimensions during training.

Table C.1.: Action space statistics computed over 77,634 expert demonstration frames. Dimensions 0–6 are the 7 joint position targets (radians). Dimension 7 is the single gripper command, which outputs only two values: -1.0 (open) and $+1.0$ (closed). Its standard deviation ($\sigma = 0.9958$) is high because it is a binary signal, not a continuous one. Note that this is the *action output* dimension; the variance mismatch problem in Section 5.3.4 arises from the *proprioceptive input*, specifically the two gripper finger joint positions (`qpos[7:9]`), which have $\sigma \approx 0.0108$ m – far smaller than the joint angle targets.

Dim	Description	Mean	Std
0	Joint 1	0.0026	0.0910
1	Joint 2	0.5537	0.1722
2	Joint 3	-0.0012	0.0667
3	Joint 4	-1.9407	0.2108
4	Joint 5	0.0013	0.0614
5	Joint 6	2.4912	0.1524
6	Joint 7	0.7957	0.3754
7	Gripper	0.0931	0.9958

Applying global Z-score normalization across all 8 dimensions would scale dimension 7 by $1/0.9958 \approx 1.004\times$, which is harmless. The variance mismatch problem described in Section 5.3.4 arises from the *proprioceptive* input, not the action output: including gripper finger joint positions (`qpos[7:9]`, physical range 0.018–0.040 m, $\sigma \approx 0.0108$ m)

alongside arm joint angles ($\sigma \approx 0.06\text{--}0.38$ rad) in a single Z-score normalization amplifies the finger noise by approximately $100\times$. The action statistics here confirm that the joint action targets themselves are well-conditioned for regression.

D. Failure Taxonomy: Classification Rules

Section 3.3.2 describes the five failure categories in qualitative terms. This appendix gives the exact decision rules used by the classification script.

Each episode produces three logs: the sequence of actions $\{a_t\}$, the `is_grasped` boolean flag $\{g_t\}$, and the TCP position sequence $\{p_t\}$ with corresponding goal positions $\{\hat{p}_t\}$. Algorithm 2 lists the classification procedure in priority order. Categories are mutually exclusive: the first matching rule is applied.

Table D.1.: Numerical thresholds used in the failure classification script. All thresholds are fixed across all experiments.

Category	Condition	Threshold
Idle	Mean action variance $\bar{\sigma}_a^2$	< 0.001
Near Miss	Minimum TCP-to-goal distance d_{\min}	< 0.10 m

The script processes kinematic logs only. It does not access visual observations or encoder outputs, so it applies uniformly across all visual representation configurations.

Algorithm 2 Episode Failure Classification

Require: Action log $\{a_t\}$, grasp log $\{g_t\}$, TCP log $\{p_t\}$, goal log $\{\hat{p}_t\}$

Ensure: Category $\in \{\text{Success, Idle, No Grasp, Drop, Near Miss, Unknown}\}$

```
1: if episode terminated with success = True then
2:   return Success
3: end if
4:  $\bar{\sigma}_a^2 \leftarrow \text{mean}(\text{var}(\{a_t\}, \text{axis} = 0))$ 
5: if  $\bar{\sigma}_a^2 < 0.001$  then
6:   return Idle ▷ Robot barely moved
7: end if
8: ever_grasped  $\leftarrow \exists t : g_t = \text{True}$ 
9: if  $\neg$  ever_grasped then
10:  return No Grasp
11: end if
12: if ever_grasped and  $g_{\text{last}} = \text{False}$  then
13:  return Drop
14: end if
15:  $d_{\text{min}} \leftarrow \min_t \|p_t - \hat{p}_t\|_2$ 
16: if  $d_{\text{min}} < 0.10$  m then
17:  return Near Miss ▷ Reached goal vicinity, failed placement tolerance
18: end if
19: return Unknown
```

E. SPOT Slot Attention Visualizations

The following figures show per-slot attention maps from the frozen SPOT encoder on representative `PickCube-v1` expert demonstration frames. Each figure shows one timestep: two rows for the Base Camera (front-top view) and two rows for the Side Camera (diagonal view). The attention sum for each slot is shown in parentheses. These cases support the analysis in Section 6.3.

Case 1: Clean Separation. Figure E.1 shows a frame where the cube and goal marker are spatially separated and assigned to different slots. This is the best-case scenario for mask-derived goal coordinates. Even here the assignment is not perfectly clean across all slots, but the goal and cube are distinguishable.

Case 2: Cube and Goal Merged into One Slot. Figure E.2 shows a frame where the two objects are spatially close and the encoder assigns them to the same slot. A centroid computed from any of the mixed slots falls between the objects rather than on the goal position.

Case 3: Goal Slot Collapse Under Arm Occlusion. Figure E.3 shows four frames from a single trajectory. The goal marker is visible in the first two frames. From Step 40 onward, the descending gripper occludes it and no slot in either camera view shows a goal response. The encoder processes each frame independently and has no mechanism to infer the goal position from prior frames once it is blocked.

Case 4: Goal Merged with Cube from the Start. Figure E.4 shows a trajectory where the goal marker is never assigned a dedicated slot. It appears merged with the cube from the first frame, before any occlusion occurs, making clean goal localization impossible throughout the episode.

Case 5: Slot Assignment Instability Across Frames. Figure E.5 shows four frames from a single trajectory. The slot index corresponding to the goal marker and cube shifts between timesteps. Because slot attention is permutation-invariant and applied independently per frame, there is no consistency constraint across time. A policy relying on mask centroids would need to solve a slot-to-object matching problem at every timestep.

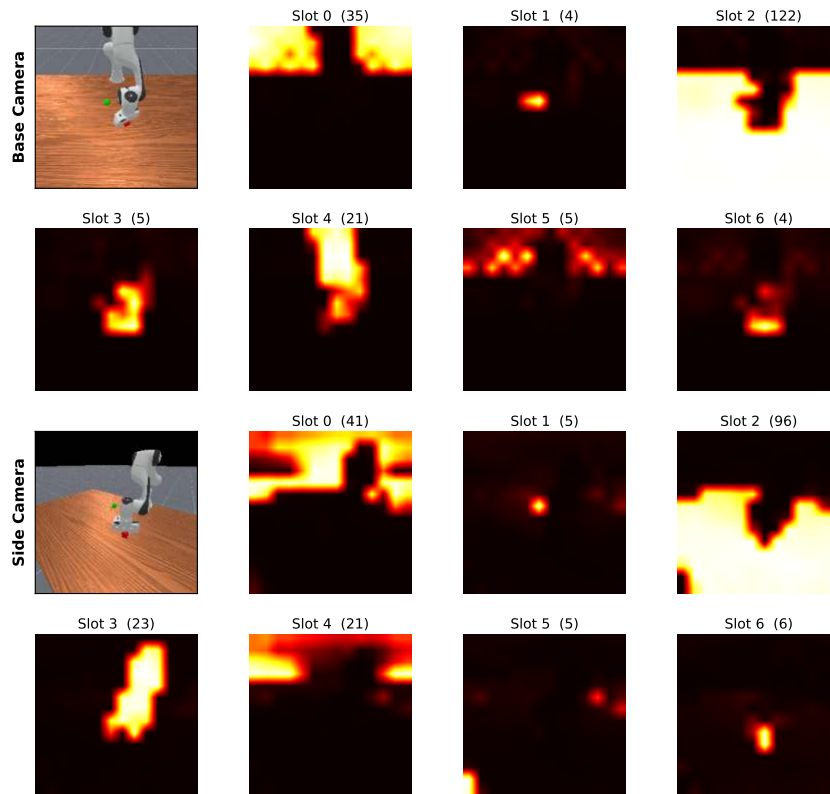


Figure E.1.: Case 1 (Clean Separation): Trajectory 988, Step 40. The cube and goal marker occupy different slots. Base Camera Slot 1 captures the goal marker; Slot 6 captures the cube. Slot 2 dominates (≈ 120), covering the table surface.

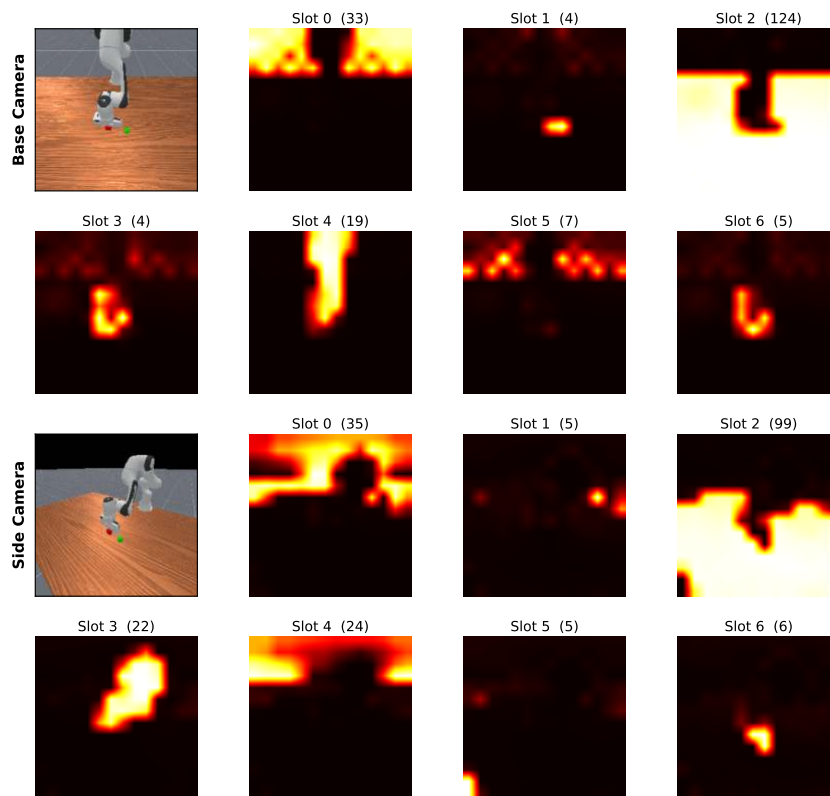
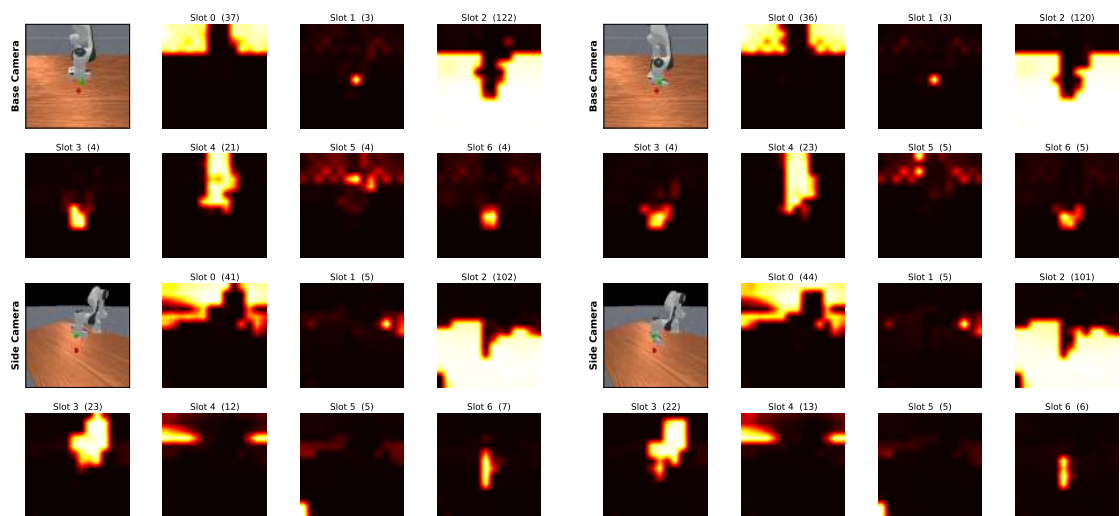
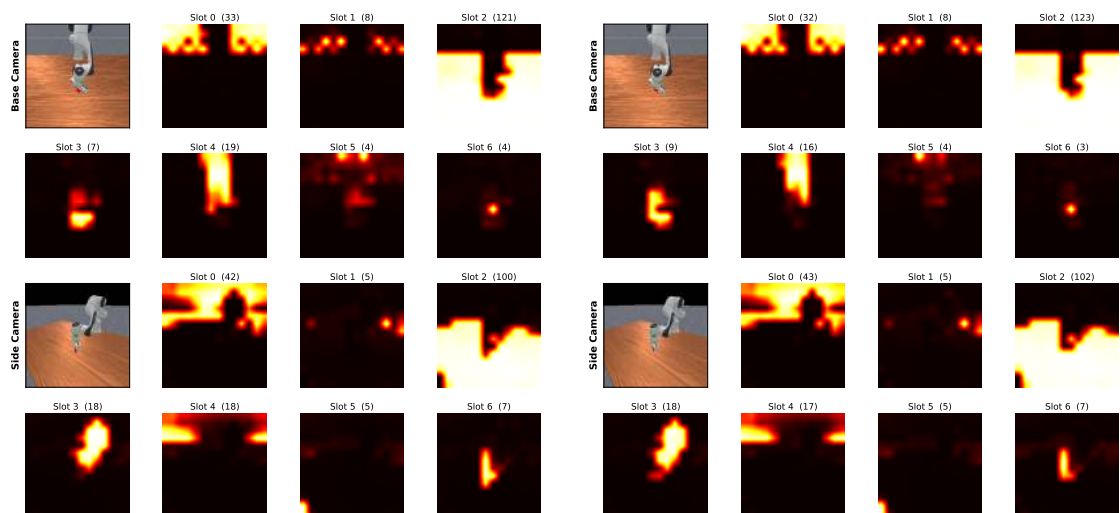


Figure E.2.: Case 2 (Merged Slot): Trajectory 20, Step 40. The cube and goal marker are close. Base Camera Slots 1 and 6 and Side Camera Slot 3 all show mixed attention covering both objects. Side Camera Slot 1 shows no goal response at this step.



(a) Step 5. Goal visible in Base Camera Slot 1.

(b) Step 20. Goal still visible. Arm not yet descending.



(c) Step 40. Gripper occludes goal. Goal response absent from all slots.

(d) Step 60. Goal remains absent. No recovery.

Figure E.3.: Case 3 (Goal Collapse Under Occlusion): Trajectory 259, Steps 5–60. Goal slot response is lost from Step 40 onward and does not recover for the remainder of the episode.

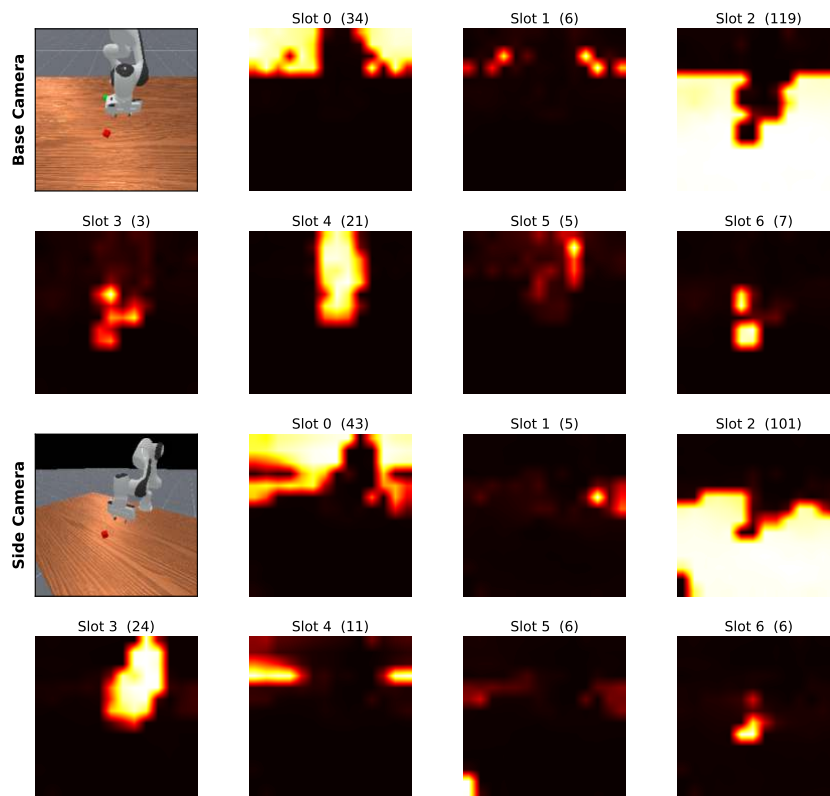


Figure E.4.: Case 4 (Merged from Start): Trajectory 5, Step 5. The goal marker is visible only from the Base Camera. It appears merged with the cube in Base Camera Slot 6. No slot in either view isolates the goal alone at the start of the episode.

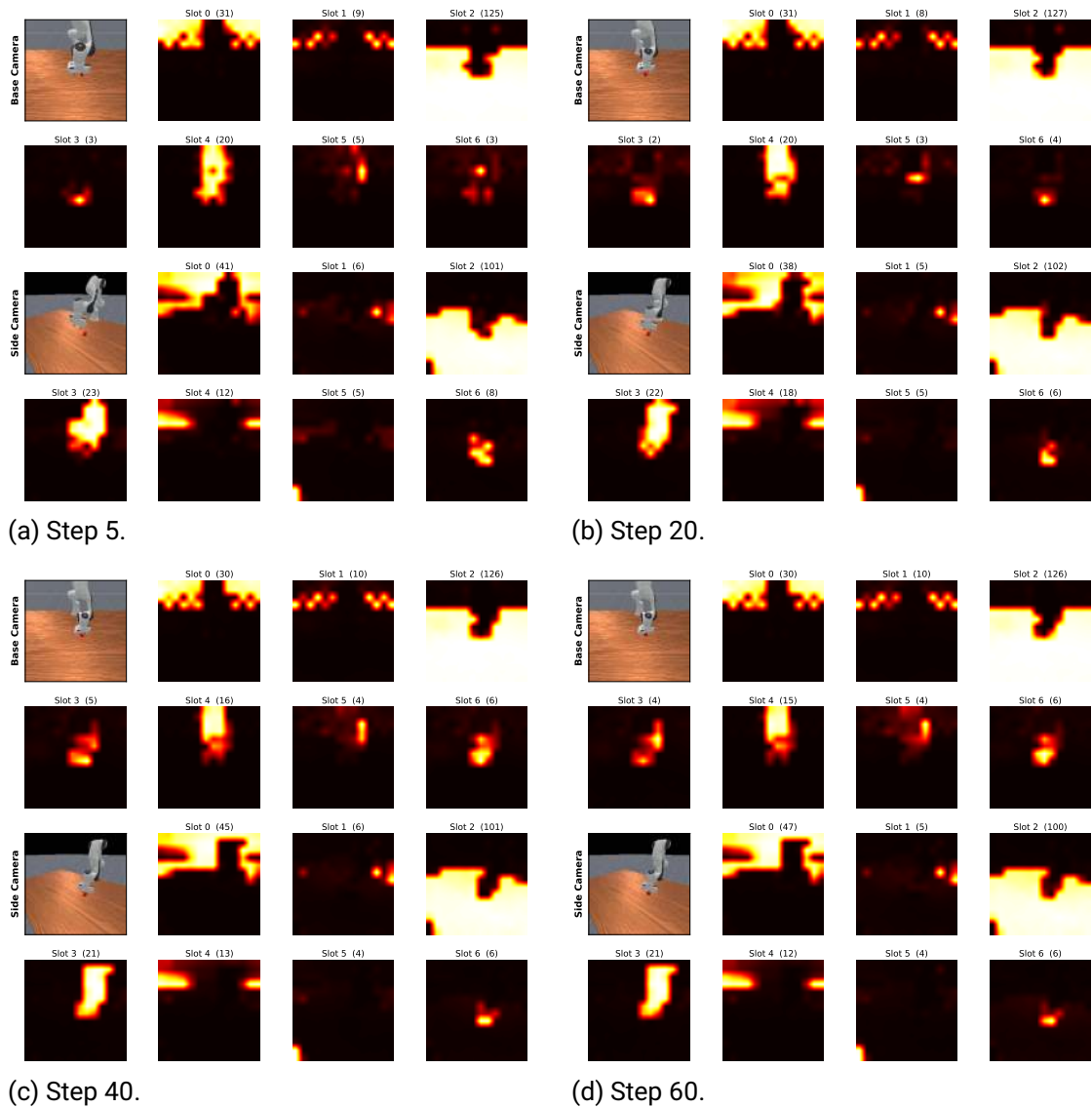


Figure E.5.: Case 5 (Slot Instability): Trajectory 786, Steps 5–60. The slot index for the same physical object shifts across frames. The encoder applies slot attention independently per frame with no temporal consistency constraint.