Proceedings of Machine Learning Research vol 284:1–27, 2026 20th Conference on Neurosymbolic Learning and Reasoning

Learning to Assemble Novel Structures with Unfamiliar Parts under Semantic Constraints

Jonghyuk Park JAY.JH.PARK@ED.AC.UK
Alex Lascarides ALEX@INF.ED.AC.UK
Subramanian Ramamoorthy S.RAMAMOORTHY@ED.AC.UK
School of Informatics, University of Edinburgh
10 Crichton Street, Edinburgh EH8 9AB, UK

Editors: Alessandra Mileo, Andrea Passerini and Cogan Shimizu

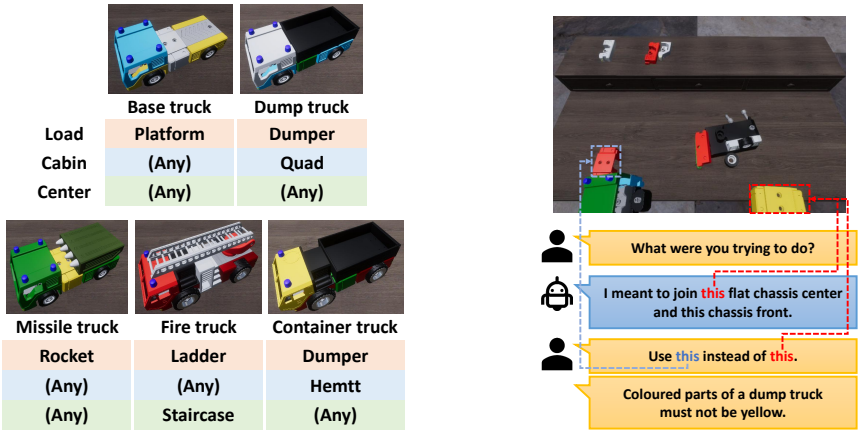
Abstract

This paper describes a neurosymbolic architecture for learning to assemble novel structures using evidence from embodied conversations and task demonstrations. We focus on scenarios where an agent encounters, after deployment, *semantic constraints* on structures—in other words, constraints as to which part types and features make valid structures—that were not available during training, and where it is *initially unaware* of the relevant structure and component part concepts. The agent must acquire and exploit such knowledge through user interactions *while* attempting assembly. We study this setting in a simulated toy truck assembly domain, learning from symbolic evidence encoded in natural language and from dense visual observations. Our experiments show that communicating semantic constraints through natural language (e.g., "dump trucks have a dumper") yields more data-efficient online adaptation than relying only on task demonstrations and/or only naming the parts through natural language.

1. Introduction

Robotic assembly in open-ended environments requires agents to cope with task knowledge that may not be available before deployment (Jiang et al., 2022). Prior work commonly distinguishes assembly knowledge into i) geometry of parts and joins, ii) assembly orders, and iii) low-level control skills (Lee et al., 2024). These aspects comprise largely physical constraints and are well suited to learning from task demonstrations (Zhu and Hu, 2018).

However, successful assembly may also depend on *semantic constraints*: arbitrary domain conventions under which it is physically feasible to join two parts, but doing so creates an invalid structure. In the example shown in Fig. 1(*b*), the join attempt is corrected despite geometric compatibility because it violates a semantic rule: namely that "Coloured parts of a dump truck must not be yellow". Demonstrations alone are insufficient as a corrective signal here, since multiple rules may be consistent with them: the constraint in Fig. 1(*b*) might have been, for example, "Dump trucks must have a red chassis center". The scope of the rule is also latent, as it may apply to a specific subtype of truck or to all trucks. The challenge becomes even greater when the agent is initially unaware of the concepts in which the constraints are expressed; e.g., if it never encountered a flat chassis center nor its label during training. The agent cannot logically infer a rule referring to a concept that it doesn't know exists.



(*a*) Truck types defined by parts. (*b*) User-agent dialogue.

Figure 1: Illustration of an assembly domain, and an example user-agent interaction.

We address this setting with a neurosymbolic agent architecture for online adaptation from embodied natural language (NL) interaction and task demonstrations. The central challenge is that domain knowledge may be introduced piecemeal during deployment, and the agent must change its behaviour on the timescale of the current interaction. This motivates separating dense perceptual grounding from explicit symbolic memory and planning: visual classifiers can be updated via new exemplars; the lexicon and knowledge base can be extended with new concepts and constraints; and the planner can immediately use the updated knowledge to revise object selection and action sequences. The neurosymbolic coupling is therefore primarily perception-to-reasoning: neural perception supplies graded candidate groundings, and newly acquired symbolic knowledge selects among these groundings and revises action plans, without directly revising the visual feature representation. This design targets the online adaptation problem studied here, while leaving tighter symbolic-to-neural feedback as a compatible extension for settings that require representation-level revision.

We evaluate the proposed framework in a simulated toy truck assembly domain. Truck types are defined by constraints over part subtypes and attributes (see Fig. 1(*a*)). We compare an agent that can interpret generic NL statements expressing semantic constraints—such as *dump trucks have a dumper*—against ablative baselines that receive demonstrations, part labels, and/or non-generic correction signals. The results show that communicating semantic constraints through NL generics improves data efficiency for online learning in this assembly domain, beyond the benefits of naming novel part concepts alone.

We publicly release the codebase implementing our agent architecture and the simulation environment for experiments at `https://github.com/jpstyle/semantic-assembler`.

2. Semantic Assembly: Task Setting and Domain

2.1. Task Domain, Goal and Input Formulation

In our formulation, a semantic assembly domain defines two sets of concepts, Π and Σ:

- Π is a set of *atomic part concepts*, represented as unary predicates over primitive objects; their instances are minimal building blocks of assembly structures: e.g., dumper, quad_cabin, cabin.

- Σ is a set of *subassembly concepts*, represented as unary predicates over composite structures; their instances consist of more than one atomic parts assembled together: e.g., truck, fire_truck, truck_front.

Hyper/hyponymy, or *is-a*, relations are represented as subclass implications between unary predicates in Π and in Σ: for instance, quad_cabin is a subtype of cabin; and fire_truck is a subtype of truck. Holo/meronymy, or *has-a*, relations are represented using the binary predicate $have(x, y)$ between Π and Σ (e.g., A truck has a cabin) or within Σ (e.g., A truck has a truck_front). Together, these predicates and implication/relation rules constitute the domain theory Ω.

Each instance of a semantic assembly task is characterised by: (a) two sets of objects C and D laid out on the tabletop; and (b) an assembly goal $\gamma \in \Sigma$. C contains exactly the 'ground-truth' atomic parts required to build an instance of γ, while D is a (possibly empty) set of 'distractor' parts, which are geometrically feasible for assembly but will violate a semantic constraint if used. Given the subtype rules within Π, an object in C or D may satisfy multiple part predicates. Parts also have colour attributes, and so may be referred to as 'a red quad cabin' in NL, for example.

Each goal concept $\gamma \in \Sigma$ is associated with two formal specifications G_γ and S_γ, which together define the range of valid assembly structures admissible as an instance of γ:

- $G_\gamma = (V_\gamma, E_\gamma)$ is an intensional structural specification ('assembly topology graph') of γ that specifies the necessary and sufficient part joins in a valid instance of γ. Each node in V_γ specifies a type requirement from Π or Σ. The edges E_γ specify required joins and their relative poses.

- S_γ is a set of quantified first-order logic (FOL) rules or integrity constraints relevant to γ (see Fig. 1(*a*)), which must hold in all instances of γ. For instance, "Dump trucks have a dumper" is encoded as $\forall x \exists y. dumpTruck(x) \rightarrow dumper(y) \wedge have(x, y)$.

The task objective is to plan and execute a sequence of actions that constructs a structure satisfying G_γ while observing all constraints in S_γ, using only parts from C and not D. The agent has a single-view RGB image $\mathcal{I} \in [0, 1]^{3\times H\times W}$ of the scene, where H and W are the sensor height and width respectively. During assembly, the agent engages in an NL dialogue with a teacher who supervises the task execution; see §2.4 for details.

2.2. Scope and Assumptions

In this study, we make several simplifying assumptions to isolate deployment-time acquisition of semantic assembly knowledge from low-level robotics and open-domain language-understanding challenges. Experiments are conducted in a simulated toy-truck domain and we evaluate online adaptation within this domain, rather than cross-domain transfer or real-world robustness. The agent receives oracle object masks and 6D poses from the simulator. Demonstrations are assumed to be segmented into parametrised primitive actions

which are executed by an oracle controller. Teacher interactions follow a controlled dialogue protocol, so the experiments test the effect of receiving generic semantic content rather than open-domain NL understanding. The visual feature backbone is fixed: online perceptual adaptation updates exemplar memories and concept classifiers, not visual features. Colour predicates are grounded by the same exemplar-based procedure as part concepts, but their classifiers are initialised before the main assembly episodes from a small set of positive and negative exemplars. Finally, instances of the same atomic part concept are assumed to share a 3D geometry, enabling type-level point-cloud and join-pose representations.

2.3. Initial Agent Knowledge and Task Abstraction

We consider a setting in which the agent possesses domain-neutral skills but lacks domain-specific assembly knowledge. The domain-neutral skills include primitive action interfaces and a high-level symbolic planning faculty that deploys them. We adopt *Answer Set Programming* (ASP; Lifschitz 2019), a declarative programming approach based on normal logic programs. We use ASP because it allows seamless integration of learned semantic constraints in planning, also encoded as ASP program fragments, thanks to its ability to model indirect action effects (Tran et al., 2023). App. A describes how the assembly planning problem is implemented as ASP programs.

At the start, the agent is ignorant of domain-specific knowledge: the concept sets Π and Σ (i.e., the hypothesis space of possible parts and structures), the domain theory Ω, the 2D-visual and 3D-geometric features of the atomic parts in Π, the assembly topology graphs G_γ and the semantic constraints S_γ for each $\gamma \in \Sigma$ are empty. The agent thus lacks the hypothesis space of possibilities and must acquire it after deployment. Our main hypothesis is that generic NL statements improve online learning by communicating reusable semantic constraints, beyond what can be obtained from demonstrations and part labels alone.

2.4. Agent-Teacher Interactions

Task demonstrations and NL dialogue provide complementary evidence. Demonstrations convey continuous information, in particular the relative poses required to join parts. Within our controlled dialogue protocol, each episode begins with a teacher-specified goal: "Build a γ". If the agent lacks the target concept or required part concepts, the teacher provides either a demonstration, a verbal definition, or labelled exemplars. During planning and execution, agent failures expose what kind of knowledge is missing: an "Is there a dumper?" query calls for a labelled exemplar; a mistaken "I meant to join this quad cabin" response lets the teacher correct the part type; and a geometrically feasible but invalid join calls for a semantic constraint. Teacher feedback targets the exposed gap, updating visual exemplars, the lexicon, or symbolic memory before replanning. A fuller account of the possible interaction flows, along with a visual flowchart depiction, is provided in App. B.

3. The Neurosymbolic Agent Architecture

This section describes the neurosymbolic architecture used to implement the interaction protocol above. Before describing individual modules, Alg. 1 summarises the episode-level control loop: perception produces candidate symbolic groundings, dialogue updates memory

Algorithm 1 Interactive semantic assembly episode

Require: goal utterance u, scene image $\mathcal{I}$, visual exemplar base XB, symbolic knowledge base $KB = (\Omega, \{G_\sigma, S_\sigma\}_{\sigma \in \Sigma})$, lexicon L

Map u to target concept γ using lexicon L.
if γ or required concepts are unknown **then**
  Obtain teacher definition, labelled exemplars, or demonstration.
  Update XB, KB, L as applicable.
end if
while task not accepted by teacher **do**
  Perceive scene from $\mathcal{I}$ and produce candidate part groundings with scores.
  Solve goal-selection ASP to choose an object-filled goal structure under current G_γ, S_γ, and grounding.
  if a required part cannot be grounded **then**
    Ask teacher for an instance and update XB.
    continue
  end if
  Solve join-sequence ASP using the selected goal structure, then execute the resulting sequence.
  if teacher interrupts **then**
    Explain intended action.
    Update XB and/or KB according to the diagnosed error and active strategy.
  end if
end while

when knowledge gaps are exposed, symbolic reasoning selects goals and plans joins, and actuation executes the selected actions. Fig. 2 illustrates the overall architecture.

3.1. Vision Processing Module

The vision processing module maps dense visual inputs to structured symbolic representations consumed by the planning and learning components. Given a single-view RGB image of the scene, the module outputs, for each detected object instance: 1) a set of candidate atomic part concepts from Π with associated confidence scores; and 2) a 3D geometric representation suitable for specifying join relations. Confidence scores produced by the vision module are propagated to the symbolic layer, where they influence planning decisions. Implementation details of feature extraction, classification, geometry extraction, and pose handling are given in App. C.

3.2. Dialogue Management Module

The dialogue management module interprets teacher utterances and generates agent responses in the controlled interaction protocol of §2.4. It handles goal utterances, clarification requests, corrective feedback, and generic statements that express semantic constraints. Our implementation deploys an off-the-shelf large-coverage semantic parser (Copestake and Flickinger, 2000), appended with a heuristic postprocessing pipeline. Here, generic statements that express semantic constraints take two forms:

- Universal constraints dictate that a part type of a subassembly must all have certain qualities. For example, "All fenders of fire trucks are red" is:

$$\forall x \forall y. fireTruck(x) \wedge fender(y) \wedge have(x, y) \rightarrow red(y)$$

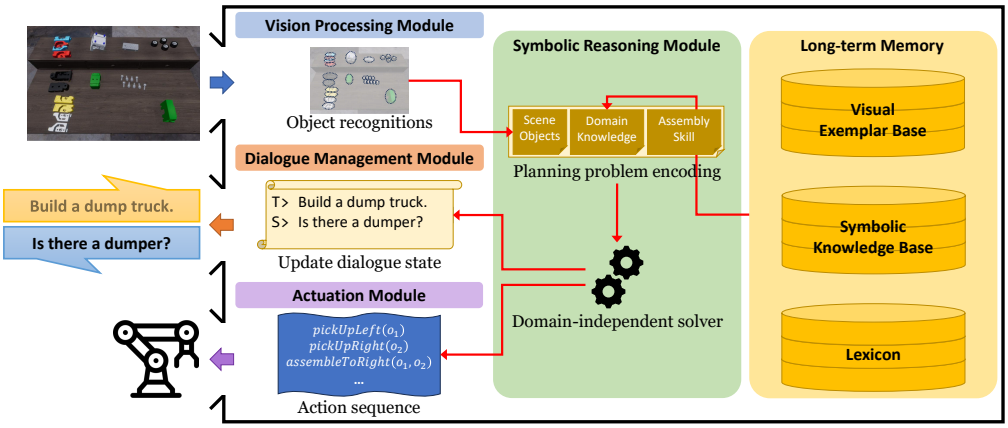


Figure 2: Overview of the neurosymbolic agent architecture.

- Existential constraints, that a certain subassembly must have at least one part with certain qualities. For example, "Missile trucks have a green fender.":

$$\forall x \exists y. missileTruck(x) \rightarrow fender(y) \wedge green(y) \wedge have(x, y)$$

These FOL statements undergo an additional automated translation step into ASP program clauses, to be integrated with ASP planning problems. The above examples become the following, where c_i is the index of the violated constraint:

$$violated(c_i) \leftarrow fireTruck(X), fender(Y), have(X, Y), \text{ not } red(Y).$$

$$violated(c_i) \leftarrow missileTruck(X), \#count\{Y : fender(Y), green(Y), have(X, Y)\} = 0\,.$$

3.3. Actuation Module

The actuation module executes the primitive actions selected by the symbolic planner using the oracle simulator controller described in §2.2.

3.4. Long-term Memory Module

The long-term memory module stores the knowledge updated through Alg. 1: the visual exemplar base (XB), the symbolic knowledge base (KB), and the lexicon. The visual XB stores positive and negative visual exemplars for each atomic part concept $\pi \in \Pi$, respectively χ_π^+ and χ_π^-. Each exemplar is represented as a feature vector generated by the vision processing module. Whenever either χ_π^+ or χ_π^- is updated, a new binary classifier for π is induced. New entries are added when the teacher says an object is (not) an example of π.

The symbolic KB stores the domain theory Ω, the assembly topology graphs G_γ, and the semantic constraint sets S_γ for each $\gamma \in \Sigma$. As explained in §2.4 and §3.2, these structures are populated from teacher definitions, demonstrations, and generic utterances.

The lexicon maps concepts in Π, Σ to their NL labels. We do not directly use NL labels as the internal indices $\pi \in \Pi$; instead, the lexicon tracks associations between internal concept

indices and teacher-provided labels. This separation allows us to compare strategies with and without shared part vocabularies.

3.5. Symbolic Reasoning Module

The symbolic reasoning module implements the two ASP planning subproblems in Alg. 1: goal selection and join-sequence planning. Long-horizon planning for assembly under semantic constraints has a combinatorially large search space, so we decompose the planning problem into the two subproblems. In the goal selection phase, the agent selects the best set of recognised scene objects that can be used to assemble the goal concept γ by building a structure instantiating the assembly topology G_γ. Currently known constraints in S_γ are translated into ASP clauses (see above) and added to the ASP encoding of the goal selection problem, so that the agent avoids knowingly violating the constraints.

Each possible goal structure is scored by the visual compatibility of part recognition, as measured by the sum of corresponding confidence scores, and the count of any violated constraints in S_γ. The result of the goal selection process is that the agent makes decisions as to which objects are instances of which (currently known) atomic part concepts. Our implementation uses the multi-shot solving feature of the ASP solver Clingo (Gebser et al., 2019) for incremental optimisation of the score. In case the agent fails to visually recognise the full set of objects needed for building γ, our ASP encoding encourages partial planning towards fulfilling G_γ, after which the agent reports planning failure and requests feedback from the teacher (see the "grounding failure" link in Fig. 9). Our encoding also accommodates re-planning during execution after planning failures or teacher corrections.

The output of the goal selection phase is passed to the join sequence planning phase, in which the agent plans a collision-free order of joins. Clingo's capability to solve ASP programs modulo theories (Gebser et al., 2016) allows us to check whether there exists a collision-free joining path between two subassemblies by integration of any arbitrary motion planner. Any infeasible joins are discarded and remembered *during* plan search. We record in App. G unique motion-planner calls as an auxiliary metric, since expensive collision checks can affect performance. The planned action sequence is finally passed to the actuation module for execution.

4. Interactive Learning Procedures

4.1. Agent-Teacher Interaction Strategies

We compare four interaction strategies that share the same architecture, task abstraction, visual backbone, planner, and oracle execution assumptions, but differ in what kinds of teacher feedback they can use to update memory.

NoLabels+CaseMemory: The agent does not share a NL vocabulary for the part concepts in Π. It cannot interpret concept labels or generic statements, but it can still store correction cases over its own internal concept symbols when teacher interventions indicate that one attempted choice should be replaced by another. This serves as a diagnostic lower bound for learning without shared part labels.

LabelsOnly: The agent can use shared vocabularies obtained from NL dialogue for part concept labelling, but does not store semantic constraints from teacher corrections.

Labels+CaseMemory: The agent can use shared part concept labels and also stores non-generic correction cases of the form "Use $this_1$ instead of $this_2$", but cannot interpret generic statements that encode constraints in S_γ.

Labels+TeacherRules: This is our full approach. The agent can use concept labels and teacher-provided generic statements that are translated into symbolic constraints.

4.2. Learning Visuals and Geometries of Parts

The three label-aware strategies acquire labelled exemplars for part concepts directly through dialogue and narrated demonstrations, enabling targeted updates to visual grounding. By contrast, NoLabels+CaseMemory lacks a shared part vocabulary and relies on indirect signals, including post-hoc analysis of completed assemblies and teacher interruptions, which introduces additional noise. The extraction of 3D geometries and join poses follows the same procedure for all strategies; label-aware agents can immediately identify novel parts via linguistic cues, whereas NoLabels+CaseMemory relies on recognition uncertainty. The detailed procedures for visual grounding and 3D geometry extraction are provided in App. D.

4.3. Learning Semantic Constraints

Labels+TeacherRules learns semantic constraints directly from generic teacher statements, as shown in Fig. 9. When the agent violates a semantic constraint, the teacher states the violated rule, and the agent translates it into FOL and ASP clauses added to S_γ. LabelsOnly performs no such semantic-memory update.

The +CaseMemory strategies store non-generic correction memories derived from teacher interventions. When a violation is signalled by "Use $this_1$ instead of $this_2$", the agent stores a most-specific correction case that discourages using $this_2$ (as recognised) when an unused $this_1$ (as recognised) is available. For example, suppose the agent is tasked to build a dump truck and is notified to use a red hemtt cabin instead of a blue quad cabin. The agent would then add a new ASP constraint as follows; informally, 'when building a dump truck, do not use a blue quad cabin when a red hemtt cabin is available':

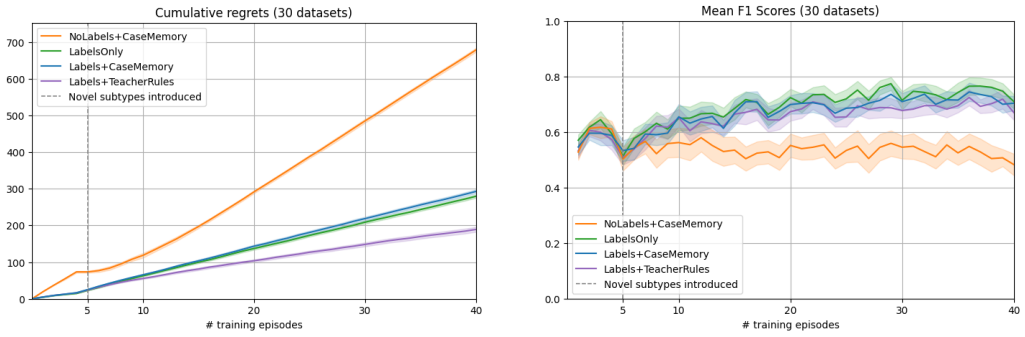
$$violated(c_i) \leftarrow dumpTruck(X), quadCabin(Y), blue(Y), have(X,Y), \\ hemttCabin(Z), red(Z), \text{ not } have(X,Z).$$

This case-memory mechanism does not abstract over corrections or discard irrelevant literals; inducing general constraints from cases, e.g. via ILP or anti-unification, is a natural direction for future work.

5. Experiments

5.1. Experimental Design and Evaluation

We test the interaction strategies on 30 datasets from the simulated toy-truck domain described in App. E, each comprising 40 randomly sampled online assembly episodes. The domain consists of 22 atomic part types in Π and 6 truck types in Σ. The first 5 problems



(*a*) Cumulative regret curves (*b*) Mean F1 curves

Figure 3: Cumulative regret and mean F1 score curves with 95% confidence intervals.

in each dataset are a 'warm-up', where the agent is tasked to assemble a generic truck without any distractors ($D = \varnothing$). After this, unforeseen subtypes of trucks and parts are constantly introduced along with randomly sampled distractors ($D \neq \varnothing$). There are no distinct 'training' and 'testing' splits—learning happens *during* task execution and the interaction with the teacher. The primary comparison is among the three label-aware strategies; NoLabels+CaseMemory is retained as a diagnostic lower bound for learning without shared part labels.

We use cumulative regret as the primary evaluation metric: i.e., the accumulated counts of errors made across a sequence of 40 planning problems. We track five types of errors in total: 1) joining structurally incompatible pairs; 2) joining at incorrect pose; 3) using a distractor part; 4) failures to ground a needed part (addressed by "Is there a X?" questions); and 5) ASP planner timeouts due to excessive grounding uncertainty. These five error types are weighted uniformly. We also monitor how the agents' visual grounding evolves over time, measured as mean F1 score curves obtained across each dataset. We report the cumulative regret and mean F1 curves averaged over the 30 datasets with 95% confidence intervals.

5.2. Results and Discussion

Fig. 3(*a*) presents the averaged cumulative regret curves across 40 episodes for each interaction strategy. NoLabels+CaseMemory incurs substantially higher regret than the label-aware strategies, reflecting the difficulty of learning novel part concepts without a shared vocabulary. Among the label-aware strategies, LabelsOnly and Labels+CaseMemory exhibit very similar regret curves, with overlapping confidence intervals and no reliable advantage from storing most-specific correction cases. This suggests that simple case memory is not an effective substitute for reusable semantic constraints in this domain. By contrast, Labels+TeacherRules consistently achieves lower cumulative regret, showing that teacher-provided generic constraints improve online assembly learning beyond part labels and non-generic correction memories. App. F provides a detailed breakdown of the five error types for all interaction strategies.

Fig. 3(*b*) shows the evolution of mean F1 scores for visual grounding. The three label-aware strategies obtain comparable grounding performance, whereas NoLabels+CaseMemory

lags behind because novel part types are introduced without explicit labelling signals. Crucially, the lower regret of Labels+TeacherRules is not explained by better visual grounding: its mean F1 is comparable to LabelsOnly and Labels+CaseMemory. The gain therefore comes from how generic constraints are used by the symbolic planner to avoid semantically invalid assemblies, not merely from improved part classification. Together, these results isolate the value of generic teacher statements as a direct channel for communicating reusable symbolic constraints during deployment.

6. Related Work

Our work relates to concept-based and explanatory interactive learning, where concept-level corrections, prototype interactions, and self-explanations revise neural or neurosymbolic models (Stammer et al., 2021, 2022, 2024). These approaches demonstrate the value of concept-level feedback, but typically focus on revising learned models with a largely fixed task vocabulary. By contrast, our agent overcomes its unawareness of part predicates, subassembly concepts, topology, and semantic constraints during deployment, with the objective of immediate replanning rather than representation-level model debugging.

Recent VLM-based neurosymbolic systems map perception and language into symbolic structures such as predicates, visual programs, or planning constraints (Athalye et al., 2026; Wüst et al., 2026; Kumar et al., 2026). Human-guided systems such as LARA likewise integrate communication, learning, reasoning, and planning (Kokel et al., 2022). Our work studies a complementary regime: a human teacher communicates generic semantic constraints as verified symbolic knowledge during task execution, and an ASP planner immediately uses them for object-role assignment and join planning. Closest to our own work are Park et al. (2023, 2025), which use generic language for visually grounded concept and domain-theory learning; we extend this line to long-horizon assembly planning under quantified semantic constraints.

7. Conclusion and Future Directions

We presented a neurosymbolic agent that learns semantic assembly tasks through controlled natural language interaction and demonstrations. The study focuses on deployment-time adaptation: the agent begins without the relevant part concepts, subassembly concepts, topology, or semantic constraints, and updates visual exemplars, symbolic memory, and (re)plans during task execution. Our empirical findings show that teacher-provided generic constraints improve online assembly learning in this benchmark beyond part labels and non-generic correction memories alone, with gains arising from symbolic planning rather than visual grounding alone. Together, these results illustrate a form of neurosymbolic adaptation in which explicit symbolic knowledge acquired during interaction immediately reshapes action selection over uncertain perceptual groundings. The study remains limited to a controlled simulated domain and a modular, primarily perception-to-reasoning form of neurosymbolic coupling. Future work should address more robust perception and language interfaces, cross-domain assembly benchmarks, abstraction from correction cases, and tighter symbolic-to-neural feedback where representation-level revision is needed.

Acknowledgments

This work was supported by Informatics Global PhD Scholarships, funded by the School of Informatics at The University of Edinburgh. Ramamoorthy is supported by a UKRI Turing AI World Leading Researcher Fellowship on AI for Person-Centred and Teachable Autonomy (grant EP/Z534833/1). We thank the anonymous reviewers for their feedback on an earlier draft of this paper, and Rimvydas Rubavicius and Gautier Dagan for continued feedback over the course of this research. Naver Labs Europe, the current employer of the attending author, sponsored registration and travel for in-person attendance to the conference venue.

References

Ashay Athalye, Nishanth Kumar, Tom Silver, Yichao Liang, Jiuguang Wang, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. From pixels to predicates: Learning symbolic world models via pretrained vlms. *IEEE Robotics and Automation Letters*, 2026.

Ann A Copestake and Dan Flickinger. An open source grammar development environment and broad-coverage english grammar using hpsg. In *LREC*, pages 591–600. Athens, 2000.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, Max Ostrowski, Torsten Schaub, and Philipp Wanko. Theory solving made easy with clingo 5. In *Technical Communications of the 32nd International Conference on Logic Programming (ICLP 2016)*, pages 2–1. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming*, 19(1):27–82, 2019.

Yuze Jiang, Zhouzhou Huang, Bin Yang, and Wenyu Yang. A review of robotic assembly strategies for the full operation procedure: planning, execution and evaluation. *Robotics and Computer-Integrated Manufacturing*, 78:102366, 2022.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026, 2023.

Harsha Kokel, Mayukh Das, Rakibul Islam, Julia Bonn, Jon Cai, Soham Dan, Narayan-Chen Anjali, Prashant Jayannavar, Doppa Janardhan Rao, Julia Hockenmaier, Sriraam Natarajan, Martha Palmer, and Dan Roth. Lara – human-guided collaborative problem solver: Effective integration of learning, reasoning and communication. *The Tenth Annual Conference on Advances in Cognitive Systems (ACS)*, 2022.

Nishanth Kumar, William Shen, Fabio Ramos, Dieter Fox, Tomás Lozano-Pérez, Leslie Pack Kaelbling, and Caelan Reed Garrett. Open-world task and motion planning via vision-language model generated constraints. *IEEE Robotics and Automation Letters*, 2026.

Regina Kyung-Jin Lee, Hao Zheng, and Yuqian Lu. Human-robot shared assembly taxonomy: A step toward seamless human-robot knowledge transfer. *Robotics and Computer-Integrated Manufacturing*, 86:102686, 2024.

Vladimir Lifschitz. *Answer set programming*, volume 3. Springer Cham, 2019.

Yuan Liu, Yilin Wen, Sida Peng, Cheng Lin, Xiaoxiao Long, Taku Komura, and Wenping Wang. Gen6d: Generalizable model-free 6-dof object pose estimation from rgb images. In *European Conference on Computer Vision*, pages 298–315. Springer, 2022.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy V Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2023.

Evin Pınar Örnek, Yann Labbé, Bugra Tekin, Lingni Ma, Cem Keskin, Christian Forster, and Tomas Hodan. Foundpose: Unseen object pose estimation with foundation features. In *European Conference on Computer Vision*, pages 163–182. Springer, 2024.

Jonghyuk Park, Alex Lascarides, and Subramanian Ramamoorthy. Interactive acquisition of fine-grained visual concepts by exploiting semantics of generic characterizations in discourse. In *Proceedings of the 15th International Conference on Computational Semantics*, pages 318–331, 2023.

Jonghyuk Park, Alex Lascarides, and Subramanian Ramamoorthy. Learning visually grounded domain ontologies via embodied conversation and explanation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 14361–14368, 2025.

Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

Wolfgang Stammer, Patrick Schramowski, and Kristian Kersting. Right for the right concept: Revising neuro-symbolic concepts by interacting with their explanations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3619–3629, 2021.

Wolfgang Stammer, Marius Memmel, Patrick Schramowski, and Kristian Kersting. Interactive disentanglement: Learning concepts by interacting with their prototype representations. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 10317–10328, 2022.

Wolfgang Stammer, Felix Friedrich, David Steinmann, Manuel Brack, Hikaru Shindo, and Kristian Kersting. Learning by self-explaining. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=bpjU7rLjJ7.

Son Cao Tran, Enrico Pontelli, Marcello Balduccini, and Torsten Schaub. Answer set planning: a survey. *Theory and Practice of Logic Programming*, 23(1):226–298, 2023.

Antonia Wüst, Wolfgang Stammer, Hikaru Shindo, Lukas Helff, Devendra Singh Dhami, and Kristian Kersting. Synthesizing visual concepts as vision-language programs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17346–17356, 2026.

Zuyuan Zhu and Huosheng Hu. Robot learning from demonstration in robotic assembly: A survey. *Robotics*, 7(2):17, 2018.

Appendix A. ASP Encodings of Planning Subproblems

The ASP program given in Fig. 4, 5 and 6 encodes the first planning subproblem described in the paper, i.e., goal selection. Note that this ASP encoding assumes a solver that supports multi-shot solving feature like Clingo.

The ASP program given in Fig. 7 and 8 encodes the second planning subproblem, i.e., join sequence planning. Note that this ASP encoding also assumes a solver that supports multi-shot solving feature, in addition to ASP-modulo-theory solving feature, again like Clingo.

Appendix B. Detailed Account of Agent-Teacher Interaction Flow

This section expands the compact description of the controlled interaction protocol in §2.4. Fig. 9 depicts the full set of teacher-learner interaction branches used in the experiments.

Each task begins with the teacher specifying the assembly goal. If the agent is unaware of the assembly target, it reports its ignorance and the teacher responds either through a full demonstration or through a verbal definition, depending on context. Demonstrations (accompanied with NL narrations) are preferred when new geometric relations or join poses must be learned. Verbal definitions suffice if the agent already possesses the relevant intermediary concepts (e.g., "A dump truck is a truck with a dumper").

If the agent believes it has sufficient knowledge, it generates a plan and begins execution. During planning, the agent may fail to visually recognise a required atomic part (which is guaranteed to exist in C). In such cases, the agent requests clarification and the teacher responds by pointing to the correct object.

If the agent makes an error during execution, the teacher interrupts and asks the agent to explain its intended action. The agent's response reveals its current beliefs about the involved objects, and exposes two possible knowledge gaps. First, the agent may have made a grounding error. The teacher corrects this by negating the incorrect grounding and asserting the correct type. Second, the grounding may be correct but the attempted action may violate a semantic constraint. The teacher then points to a valid object and expresses the violated constraint as a generic NL statement. After corrective feedback, the agent updates its knowledge and replans.

Appendix C. Vision Processing Module: Implementation details

The vision processing module parses the agent's visual inputs from its RGB sensor into structured representations that other modules can further process. The module deals with

```
% Make choice for the top node (0), which represents the build target subassembly
1{ node_sa_template(0,S,I) : possible_root_template(S,I) }1.
% Determine possible template options for the root node, from the specified
% build target type and available supertype-subtype relations
possible_root_template(S,I) :- build_target(S), template_option(S,I).
possible_root_template(S2,I) :-
    build_target(S1), subtype_of(S1,S2), template_option(S2,I).

% Supertype-subtype relation is transitive
subtype_of(T1,T3) :- subtype_of(T1,T2), subtype_of(T2,T3).

% Annotating each child of a subassembly node N with its required atomic part
% concept type
node_atomic(n(N,NS),P) :- node_sa_template(N,S,I), req_atomic(S,I,NS,P).

% Meanwhile, a subassembly template may be directly specified for a node
node_sa_template(n(N,NS),S2,I2) :-
    node_sa_template(N,S1,I1), req_template(S1,I1,NS,S2,I2).

% Project to specify subassembly type of subassembly nodes
node_sa(N,S) :- node_sa_template(N,S,I).
node_sa(0,S) :- build_target(S).    % Also account for build target subtype
% Supertype-subtype info applied for substructures
node_sa(N,S2) :- node_sa(N,S1), subtype_of(S1,S2).

% Ancestor-descendant relations among nodes
component_node(n(N,NS),N) :- node_atomic(n(N,NS),_).
component_node(n(N,NS),N) :- node_sa(n(N,NS),_).
component_node(N1,N3) :- component_node(N1,N2), component_node(N2,N3).  % Transitive

% An object that is likely to be an instance of a type may fill an atomic node
% with matching type. Each object can fill up to one atomic node, and each
% atomic node can be filled by up to one object.
can_fill(O,N,P) :- node_atomic(N,P), type_likely(O,P,PR).
can_fill(O,N,P1) :- node_atomic(N,P2), subtype_of(P1,P2), type_likely(O,P1,PR).
{ fill_node(O,N) } :- can_fill(O,N,_), not must_unify(_,N).
:- type_likely(O,_,_), #count { N : fill_node(O,N) } > 1.
:- node_atomic(N,_), #count { O : fill_node(O,N) } > 1.

% Once determined an object to fill a specific atomic node, select exactly one
% part (sub)type to commit to among the visually licensed options
1{ type_committed(O,P) : can_fill(O,N,P) }1 :- fill_node(O,N), can_fill(O,N,_).
% Supertype-subtype info percolates upwards
type_committed(O,P2) :- type_committed(O,P1), subtype_of(P1,P2).

% Always use objects labeled by user in response to agent's "Is there a ~?" queries
:- certified_label(O), not fill_node(O,_).
```

Figure 4: ASP encoding of the goal selection subproblem: 1 of 3.

```
% Identifying all objects selected to fill all descendant atomic nodes of a
% subassembly node
component_obj(O,N) :- node_sa(N,S), component_node(ND,N), fill_node(O,ND).

% For tracking which nodes will have to be connected at which contact points
to_connect(D1,D2,CP1,CP2) :-
    node_sa_template(N,S,I),
    fits_signature(D1,N,NS1,SG1), fits_signature(D2,N,NS2,SG2),
    connection_signature(S,I,NS1,NS2,SG1,SG2,CP1,CP2).
fits_signature(n(N,NS),N,NS,NS) :- node_atomic(n(N,NS),_).
fits_signature(D,N,NS,c(NS,SG)) :-
    fits_signature(D,n(N,NS),_,SG), node_sa_template(n(N,NS),_,_).

% Handling additional constraints due to already assembled parts. Atomic
% part type of existing object may or may not be specified, represented
% by arity of ext_obj predicate (ext_obj/2 vs. ext_obj/1). Similarly,
% contact sites between two joined existing parts may or may not be known
% (ext_conn/4 vs. ext_conn/2).
fresh_obj(O) :- component_obj(O,_), not ext_obj(O), not ext_obj(O,_).
    % 'Fresh' if not included in some subassembly

% If an existing object with known part type has exactly one atomic node with
% the matching type, unify
must_unify(O,N) :- ext_obj(O,P), 1{ node_atomic(_,P) }1, node_atomic(N,P).
% If a neighbor of a uniquely unified object connects with another by known
% contact site, unify (assumption here is that contact sites of each atomic part
% are all uniquely distinguishable)
must_unify(O1,N1) :- must_unify(O2,N2), ext_conn(O1,O2,CP1,CP2),
    to_connect(N1,N2,CP1,CP2).
% If a neighbor of a uniquely unified object has known type, and there exist
% exactly one node with matching type in the neighborhood of the uniquely unified
% node, unify
must_unify(O1,N1) :- must_unify(O2,N2), ext_obj(O1,P1), ext_conn(O1,O2),
    1{ node_atomic(N,P1) : to_connect(N,N2,_,_) }1,
    node_atomic(N1,P1), to_connect(N1,N2,_,_).
% If a neighbor of a uniquely unified object has known type, and there exist
% more than one nodes with matching type in the neighborhood of the uniquely
% unified node, *MAY* unify with one of them
may_unify(O1,N1) :- must_unify(O2,N2), ext_obj(O1,P1), ext_conn(O1,O2),
    2{ node_atomic(N,P1) : to_connect(N,N2,_,_) },
    node_atomic(N1,P1), to_connect(N1,N2,_,_).
% If one side of existing object connection uniquely unifies to an atomic node
% while the other's type is not specified, the latter *MAY* be unified with one
% of other atomic nodes connected to the unified node
may_unify(O1,N1) :- must_unify(O2,N2), ext_obj(O1), ext_conn(O1,O2),
    to_connect(N1,N2,_,_).
```

Figure 5: ASP encoding of the goal selection subproblem: 2 of 3.

```
% Observe must_unify relations
fill_node(O,N) :- must_unify(O,N).
% Different objects cannot be forced to unify with the same node
:- must_unify(O1,N1), must_unify(O2,N2), O1 != O2, N1 = N2.
% Observe ext_conn relations
:- ext_conn(O1,O2,_,_), fill_node(O1,N1), fill_node(O2,N2), not to_connect(N1,N2,_,_).
:- ext_conn(O1,O2), fill_node(O1,N1), fill_node(O2,N2), not to_connect(N1,N2,_,_).

% Supertype-subtype relations for ext_objs
ext_obj(O,P2) :- ext_obj(O,P1), subtype_of(P1,P2).

% Edge symmetricity
to_connect(N2,N1,CP2,CP1) :- to_connect(N1,N2,CP1,CP2).
ext_conn(O2,O1,CP2,CP1) :- ext_conn(O1,O2,CP1,CP2).
ext_conn(O2,O1) :- ext_conn(O1,O2).

% Compute average part compatibility score by taking average across all atomic
% nodes filled by recognized objects
node_score(N,PR) :-
    node_atomic(N,_), fill_node(O,N), type_committed(O,P), type_likely(O,P,PR).
avg_score(TS/NN) :- TS = #sum { NS,N : node_score(N,NS) },
    NN = #count { N : node_atomic(N,_) }, NN != 0.

% Prevent any violation of universally quantified constraints. Implemented as
% hard constraint, as violation can be evaded by deciding to not fill corresponding
% nodes with recognized objects.
:- forall_violation(_).

% Penalize any violation of existentially quantified constraints. Implemented
% as soft constraint, as plans violating these can still be admitted, where
% absence of required of parts will be handled by queries to user. In contrast,
% if implemented as hard constraints, many sane partial plans will be eliminated.
total_penalty(TP) :- TP = #sum { 30,EC : exists_violation(EC) }.

% Final goal configuration score
final_score(AS-TP) :- avg_score(AS), total_penalty(TP).

#program check(c).
#external query(c).
:- final_score(FS), query(c), FS <= c.
```

Figure 6: ASP encoding of the goal selection subproblem: 3 of 3.

```
%% Rules common to answer set planning
#program base.
holds(F,0) :- init(F).

#program step(t).
holds(F,t+1) :- holds(F,t), not -holds(F,t+1).
-holds(F,t+1) :- -holds(F,t), not holds(F,t+1).
1{ occ(A,t) : possible(A,t) }1.

#program check(t).
#external query(t).
:- query(t), not goal(t).

%% Rules common to (our way of encoding) assembly domains - Static laws
#program base.
% Part connection is a symmetric relation
to_connect(O2,O1) :- to_connect(O1,O2).

% Track every occasion where a subassembly formed in the previous timestep
% doesn't get used in the immediate next timestep (identifiable by index);
% total count of such occurrences will be minimized
#program step(t).
penalize(t) :-
    occ(join(S1,O1,S2,O2),t), holds(max_sa_index(S),t), S1 != S, S2 != S.

%% Rules common to assembly domains - Dynamic laws
#program step(t).
% join/4: Assembling two subassemblies at specified object; abstracts
% aligning & tightening.
% (Note: This implementation assumes we always assemble from right to left,
% without loss of generality)
possible(join(S1,O1,S2,O2),t) :-
    S1 != S2, O1 != O2,
    holds(part_of(O1,S1),t), holds(part_of(O2,S2),t),
    to_connect(O1,O2), not holds(connected(O1,O2),t).
holds(connected(O1,O2),t+1) :- occ(join(S1,O1,S2,O2),t).
holds(connected(O2,O1),t+1) :- occ(join(S1,O1,S2,O2),t).    % Symmetric
holds(max_sa_index(S+1),t+1) :- occ(join(S1,O1,S2,O2),t), holds(max_sa_index(S),t).
-holds(max_sa_index(S),t+1) :- occ(join(S1,O1,S2,O2),t), holds(max_sa_index(S),t).
-holds(part_of(O,S1),t+1) :- occ(join(S1,_,S2,_),t), holds(part_of(O,S1),t).
-holds(part_of(O,S2),t+1) :- occ(join(S1,_,S2,_),t), holds(part_of(O,S2),t).
holds(part_of(O,S+1),t+1) :-
    occ(join(S1,_,S2,_),t), holds(max_sa_index(S),t), holds(part_of(O,S1),t).
holds(part_of(O,S+1),t+1) :-
    occ(join(S1,_,S2,_),t), holds(max_sa_index(S),t), holds(part_of(O,S2),t).
```

Figure 7: ASP encoding of the join sequence planning subproblem: 1 of 2.

```
% Note: We don't consider 'disassemble' operation here; would need to treat
% subassembly predicate as fluent if we were to do this

% Minimize number of times occasions where agent does not immediately
% re-use the subassembly just assembled in the previous timestep for the
% next join action, thus increasing the number of primitive actions needed.
% It is important that this optimization statement is placed here, as part
% of the program fragment step(t).
#minimize { 1,T : penalize(T) }.

%% Goal condition; all object pairs to be connected have been 'connected'
% at time step t, either truly or speculatively
#program check(t).
goal(t) :- holds(connected(O1,O2),t) : to_connect(O1,O2).

%% Problem init; common to all problems
#program base.
% Initial configs
init(max_sa_index(-1)).
```

Figure 8: ASP encoding of the join sequence planning subproblem: 2 of 2.

three visual subtasks: object detection (including predictions of its types); point cloud extraction; and pose estimation.

The goal of object detection is to localise and to classify scene objects. We use binary segmentation masks to represent localised object instances instead of bounding boxes since they allow more accurate visual feature extraction for classification. Our primary focus is on learning to recognise classes of objects rather than their locations, since this directly interfaces with the goal of identifying novel parts. While high-quality, off-the-shelf segmentation models such as SAM (Kirillov et al., 2023) could be used for object localization, it would introduce uncontrolled confounders for testing our main hypotheses. As stated in §2.2, experiments use ground-truth masks obtained from the simulated environment.

We implement few-shot open-set classification by keeping a collection of lightweight binary classifiers for each atomic part concept that the learner is currently aware of. (This subset of 'aware' part concepts Π keeps expanding as interaction with the teacher proceeds.) The binary classifiers are induced from positive and negative exemplars stored in the agent's long-term memory and gained from teacher interactions (see §2.4). Our implementation uses the DINOv2-base model (Oquab et al., 2023) as a feature extractor backbone, which takes a scene image and object masks as input, and outputs are fed into RBF-kernel SVMs to yield per-concept predictions.

Atomic part concepts need to have proper representations of their 3D structures for accurate manipulation during assembly. Under the type-level geometry assumption stated in §2.2, each atomic part concept is represented by a point cloud. When the learner encounters an atomic part type for the first time, and notices that it does not correspond to any previously known types, the learner registers a new atomic part concept in Π and associates it with a point cloud representation extracted from the instance. Given that visual inputs are single-view RGB images, the learner picks up the novel part concept instance and collects a

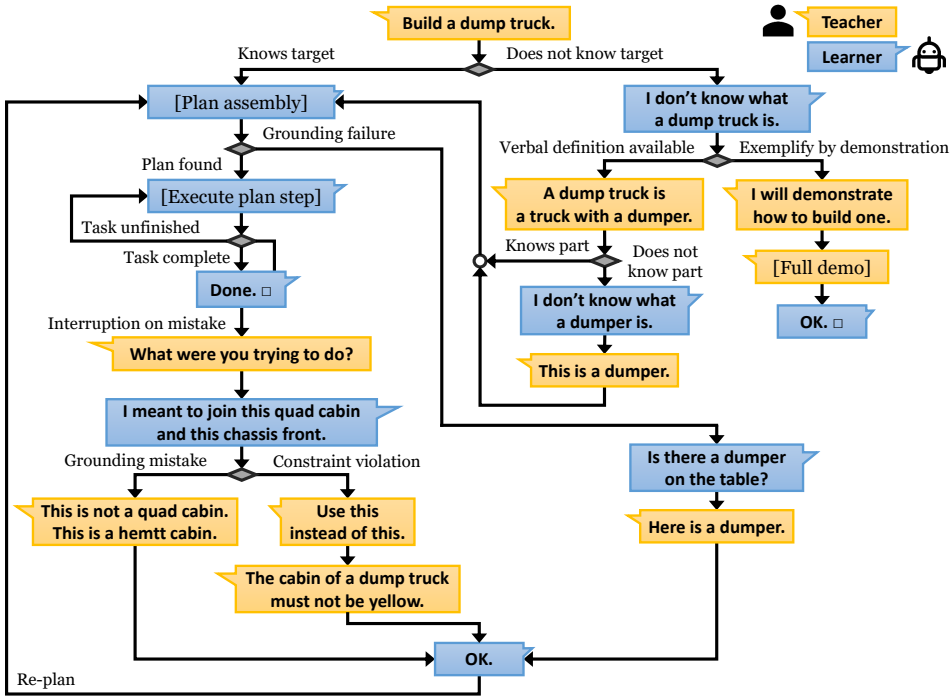


Figure 9: Possible teacher-learner interactions. □ marks termination of an episode.

set of images of the object observed from different viewpoints with known poses. The feature extractor processes each image to obtain per-pixel embeddings at a lower resolution, which can be used as point descriptors to be fed into a photogrammetric point cloud extraction algorithm. Our implementation employs COLMAP (Schonberger and Frahm, 2016) for point cloud reconstruction.

Estimation of 6D poses of objects is necessary to achieve accurate joining of parts at specified relative poses. Few-shot or zero-shot object pose estimation with RGB images has witnessed significant research progress with the advent of powerful vision foundation models (Liu et al., 2022; Örnek et al., 2024). As stated in §2.2, experiments use ground-truth object poses from the simulated environment after each manipulation action.

Appendix D. Per-strategy Details on Learning from Visual Data

D.1. Learning Visual Grounding of Parts

The agent's visual grounding performance is determined by the binary classifiers induced from the positive and negative exemplar sets $\chi_\pi^{+/-}$ stored in the visual XB. The progress of learning to visually recognise atomic parts Π largely depends on acquiring labelled concept exemplars through interactions with the teacher. Consequently, the visual grounding learning process differs between NoLabels+CaseMemory and the label-aware strategies, since only the latter have access to NL labels for concepts in Π.

Label-aware agents directly obtain exemplar labelling through concept labelling statements like "This is a dumper", "This is not a quad_cabin", "Here is a large_wheel". As illustrated in Fig. 9, these labelling statements are given after the agent's utterance that reveals how it has mistaken a part type for another ("I meant to join this X and this Y.") or failed to ground a needed instance ("Is there a X on the table?"). Concept labels are also provided when the teacher provides full demonstrations upon the agent's ignorance of goal subassembly concepts, in which each action step is narrated in natural language. These interactions are possible only when the agent shares a NL vocabulary with the teacher.

In contrast, NoLabels+CaseMemory agents are neither able to understand the teacher's concept labelling statements nor to generate utterances that expose its imperfect grounding capability due to lack of a shared vocabulary. In case of grounding failures, the agent cannot ask "Is there a X on the table?" and has to report "I cannot find a part I need on the table" instead, after which the teacher will simply demonstrate a valid part join. After agent mistakes, the teacher cannot ask the probing question "What were you trying to join?" because agent cannot answer with its current part recognitions, so the agent is simply interrupted; afterwards, the agent will immediately request a demonstration of a valid part join.

Accordingly, the NoLabels+CaseMemory baseline in our experiments has to rely on indirect learning signals to update their exemplar sets: post-hoc analysis of each completed episode, and the teacher's interruptions upon grounding mistakes. Post-hoc analysis of an episode updates positive exemplar sets χ_π^+ by matching the topology of the completed structure against G_γ via graph matching. Negative exemplars can be inferred from teacher interruptions, which can serve as 'pairwise negative labels' of the involved object pairs. Specifically, if the join of objects o_1 and o_2 are interrupted as invalid by the teacher due to incorrect grounding, and the agent had recognised o_1, o_2 as $\pi_1, \pi_2 \in \Pi$ respectively, then it is not the case that o_1 is a π_1 and o_2 is a π_2 at the same time; otherwise, the join would not have been interrupted. From the interruption, if the agent is certain that o_1 is indeed an instance of π_1, it can logically entail o_2 is not an instance of π_2, thereby updating $\chi_{\pi_2}^-$. Note that both label acquisition processes are susceptible to noise.

D.2. Learning 3D Geometries of Novel Parts and Joins

Across the four interaction strategies, the process itself remains the same for learning point cloud representations of atomic parts and joining poses. The practical difference between NoLabels+CaseMemory vs. label-aware agents in this regard has to do with *when* the point cloud extraction procedure is triggered during interactions. As explained in §3.1, the point cloud for an atomic part type is extracted when the agent encounters its instance for the first time in its operation. In other words, the agent must first become aware that it has encountered a novel part type that is not in Π prior to extracting its 3D geometry. This is straightforward for label-aware agents, who can immediately recognise when the teacher introduces a novel part type by use of neologisms.

Conversely, NoLabels+CaseMemory agents have to rely on their own judgements to determine whether an object is an instance of a novel atomic part type. In our implementation, NoLabels+CaseMemory agents register a novel part type when an object used in the teacher's full demonstration obtains low likelihood scores from few-shot visual recogni-

tion across all currently known types in Π. Join poses involving the novel type are then obtained from the teacher's demonstration. Note that NoLabels+CaseMemory agents may mistakenly identify a familiar part type as novel, or conversely, fail to recognise a novel part as unfamiliar, which will likely result in downstream errors during task execution.

Appendix E. Full Description of Toy Truck Assembly Domain

Our simulated toy truck domain comprises 22 atomic part types in Π. See Tab. 1 for the full list of the part types, whether they can have colour attributes, and example visuals. Π also includes their supertypes:

- {cabin, load chassis_center, fender, wheel, fl_fender, fr_fender, bl_fender, br_fender, normal_fender, large_fender}.

See the domain theory Ω for hyper/hyponymy relations that hold among them.
The set of subassembly concepts Σ include goal concepts, which are specified as task goals, and intermediary concepts, which are semantically meaningful substructures of goal concepts.

- Goal subassemblies: {truck, fire_truck, dump_truck, container_truck, missile_truck}.
- Non-goal subeassemblies: {truck_front, truck_back, fl_fw_unit, fr_fw_unit, bl_fw_unit, br_fw_unit, normal_fl_fw_unit, normal_fr_fw_unit, normal_bl_fw_unit, normal_br_fw_unit, large_fl_fw_unit, large_fr_fw_unit, large_bl_fw_unit, large_br_fw_unit}.

Likewise, see the domain theory Ω for hyper/hyponymy relations that hold among them. See the assembly topology graphs G_γ in Fig. 11 for the makeups of the subassembly concepts.

The domain theory Ω features hyper/hyponymy relations and holo/meronymy relations. Fig. 10 depicts all hyper/hyponymy relations among concepts in Π and Σ. Holo/meronymy relations can be induced from the assembly topology graphs G_γ in Fig. 11.

The assembly topology graphs G_γ describe how each subassembly in Σ is composed from atomic parts in Π or other subassemblies in Σ. G_γ for the truck subtypes $\gamma \in$ {fire_truck, dump_truck, container_truck, missile_truck} inherit from G_{truck}; they are further defined with their definitional semantic constraints in S_γ. Fig. 12 illustrates how the 'fully flattened' topology of a truck looks like.

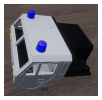
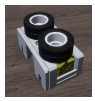
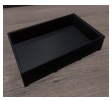
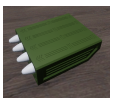
Finally, Tab. 2 provides the full list of the semantic constraints S_γ for $\gamma \in \Sigma$, merged into a single table. Note how many of the constraints, or their more specific variants, cannot be obtained by the most-specific correction memories used by the +CaseMemory strategies.

Appendix F. Breakdown of Cumulative Regrets

Fig. 13 provides a detailed breakdown of the averaged cumulative regret curves reported in the main paper.

Note in particular how Labels+TeacherRules agents make fewer distractor-usage mistakes as learning proceeds. The idiosyncratic behaviours of NoLabels+CaseMemory curves, namely the seemingly increasing error counts for most types after the warm-up problems, are due to the fact that they need full demonstrations for novel truck types, unlike label-aware agents for which verbal definitions suffice; such demonstrations do not count towards

Table 1: Atomic part types in our simulated toy truck domain. Coloured parts may have one of five colours: red, green, blue, yellow, white.

Name	Coloured	Sample image
quad_cabin	True	
hemtt_cabin	True	
chassis_front	False	
chassis_back	False	
flat_chassis_center	True	
spares_chassis_center	True	
staircase_chassis_center	True	
platform	False	
dumper	False	
ladder	False	
rocket_launcher	False	

Name	Coloured	Sample image
normal_fl_fender	True	
normal_fr_fender	True	
normal_bl_fender	True	
normal_br_fender	True	
large_fl_fender	True	
large_fr_fender	True	
large_bl_fender	True	
large_br_fender	True	
normal_wheel	False	
large_wheel	False	
bolt	False	

Table 2: The set of all semantic constraints in our toy truck assembly domain. $\perp$ denotes falsity, which must not be entailed.

NL generic statement	FOL encoding
"A base truck has a platform."	$\forall x \exists y. baseTruck(x) \rightarrow platform(y) \wedge have(x, y)$
"A dump truck has a dumper and a quad cabin."	$\forall x \exists y \exists z. dumpTruck(x) \rightarrow dumper(y) \wedge have(x, y) \wedge quadCabin(z) \wedge have(x, z)$
"A container truck has a dumper and a hemtt cabin."	$\forall x \exists y \exists z. containerTruck(x) \rightarrow dumper(y) \wedge have(x, y) \wedge hemttCabin(z) \wedge have(x, z)$
"A missile truck has a rocket launcher."	$\forall x \exists y. missileTruck(x) \rightarrow rocketLauncher(y) \wedge have(x, y)$
"A fire truck has a ladder and a staircase center."	$\forall x \exists y \exists z. fireTruck(x) \rightarrow ladder(y) \wedge have(x, y) \wedge staircaseCenter(z) \wedge have(x, z)$
"All fender pairs of a truck front must have same color."	$\forall x \forall y \forall z. truckFront(x) \wedge fender(y) \wedge fender(z) \wedge have(x, y), have(x, z) \rightarrow sameColor(y, z)$
"All fender pairs of a truck back must have same color."	$\forall x \forall y \forall z. truckBack(x) \wedge fender(y) \wedge fender(z) \wedge have(x, y), have(x, z) \rightarrow sameColor(y, z)$
"A fw-unit must not have a large fender and a normal wheel."	$\forall x \forall y \forall z. fwUnit(x) \wedge largeFender(y) \wedge normalWheel(z) \wedge have(x, y), have(x, z) \rightarrow \perp$
"A fw-unit must not have a normal fender and a large wheel."	$\forall x \forall y \forall z. fwUnit(x) \wedge normalFender(y) \wedge largeWheel(z) \wedge have(x, y), have(x, z) \rightarrow \perp$
"All fender-center pairs of a truck must not have same color."	$\forall x \forall y \forall z. truck(x) \wedge fender(y) \wedge chassisCenter(z) \wedge have(x, y), have(x, z) \rightarrow \neg sameColor(y, z)$
"A truck must not have a normal wheel and a large wheel."	$\forall x \forall y \forall z. truck(x) \wedge normalWheel(y) \wedge largeWheel(z) \wedge have(x, y), have(x, z) \rightarrow \perp$
"All cabin-center pairs of a base truck must not have same color."	$\forall x \forall y \forall z. baseTruck(x) \wedge cabin(y) \wedge chassisCenter(z) \wedge have(x, y), have(x, z) \rightarrow \neg sameColor(y, z)$
"A base truck must not have a staircase center."	$\forall x \forall y. baseTruck(x) \wedge staircaseCenter(y) \wedge have(x, y) \rightarrow \perp$
"A missile truck has a green fender."	$\forall x \exists y. missileTruck(x) \rightarrow fender(y) \wedge green(y) \wedge have(x, y)$
"All cabin-center pairs of a missile truck must have same color."	$\forall x \forall y \forall z. missileTruck(x) \wedge cabin(y) \wedge chassisCenter(z) \wedge have(x, y), have(x, z) \rightarrow sameColor(y, z)$
"All coloured parts of a dump truck must not be yellow."	$\forall x \forall y. dumpTruck(x) \wedge coloredPart(y) \wedge have(x, y) \rightarrow \neg yellow(y)$
"All wheels of a dump truck must be normal wheels."	$\forall x \forall y. dumpTruck(x) \wedge wheel(y) \wedge have(x, y) \rightarrow normalWheel(y)$
"All coloured parts of a container truck must not be blue."	$\forall x \forall y. containerTruck(x) \wedge coloredPart(y) \wedge have(x, y) \rightarrow \neg blue(y)$
"All wheels of a container truck must be large wheels."	$\forall x \forall y. containerTruck(x) \wedge wheel(y) \wedge have(x, y) \rightarrow largeWheel(y)$
"A fire truck has a white coloured part."	$\forall x \exists y. fireTruck(x) \rightarrow coloredPart(y) \wedge white(y) \wedge have(x, y)$
"All fender of a fire truck must be red."	$\forall x \forall y. fireTruck(x) \wedge fender(y) \wedge have(x, y) \rightarrow red(y)$

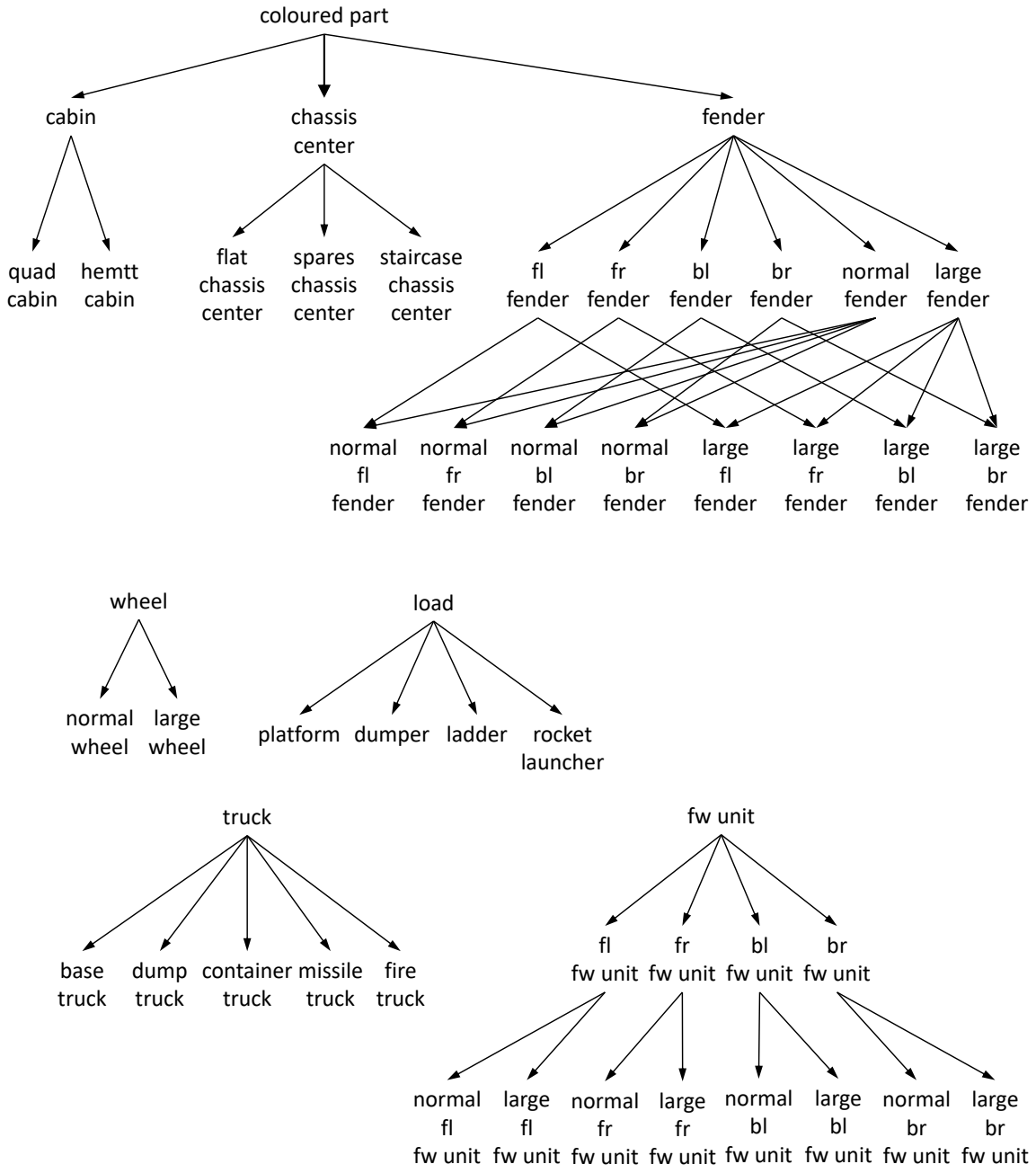


Figure 10: Supertype-subtype relations in our simulated truck domain.

any error type. It is randomly determined at which point novel truck types are introduced in each dataset, so the numbers are averaged out across the datasets, giving the false sense of 'increasing errors' as the learning progresses. Indeed, after sufficient number of episodes, NoLabels+CaseMemory agents start to show learning progress again for most error types.

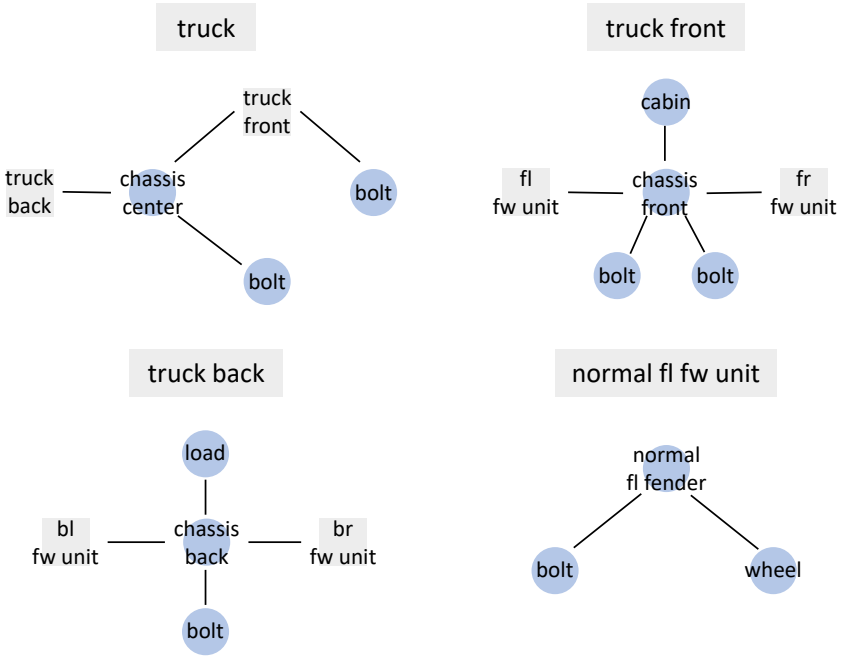


Figure 11: Assembly topology graphs G_γ for each $\gamma \in \Sigma$. Blue circle nodes denote atomic parts, whereas grey square nodes denote subassembly parts. G_γ for $\gamma \in$ {normal_fr_fw_unit, normal_bl_fw_unit, normal_br_fw_unit, large_fl_fw_unit, large_fr_fw_unit, large_bl_fw_unit, large_br_fw_unit} are parallel to the graph for normal_fl_fw_unit except the fender piece, so they are omitted as redundant.

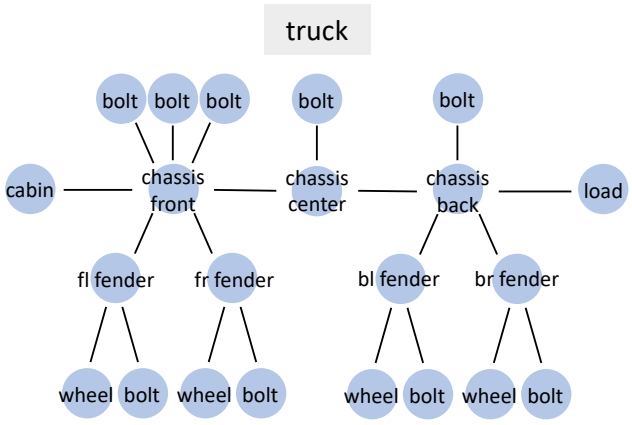
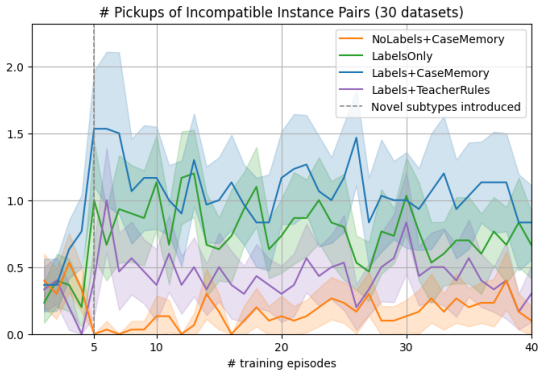
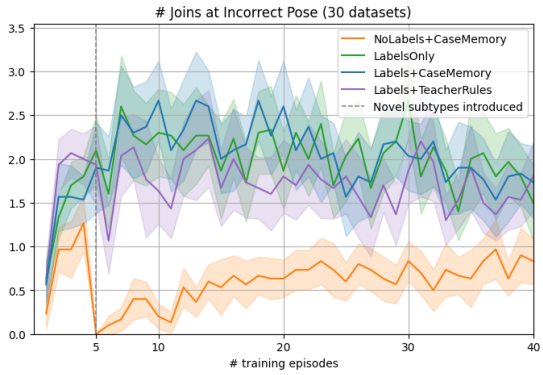


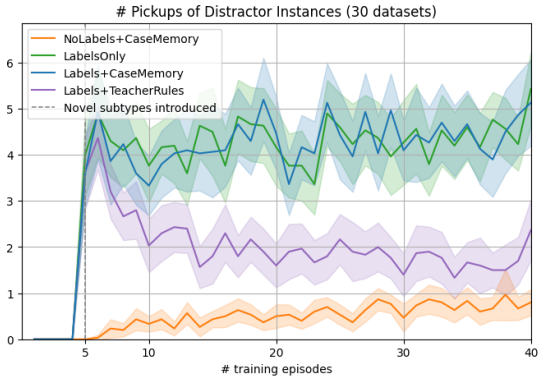
Figure 12: G_{truck}, the assembly topology of truck, 'flattened out' by recursively replacing subassembly nodes with their respective G_γ.



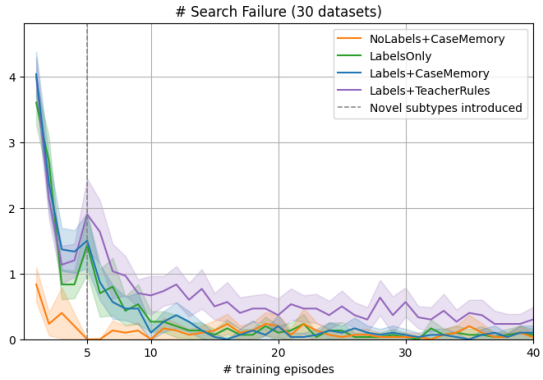
(*a*) Structurally incompatible join attempt



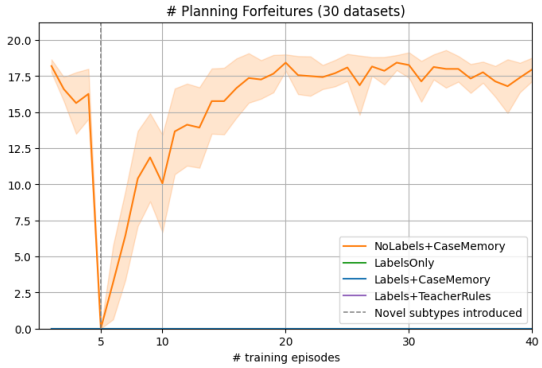
(*b*) Join at incorrect pose



(*c*) Distractor usage



(*d*) Grounding failure



(*e*) Planner timeout

Figure 13: Breakdown of the cumulative regrets into the five agent error types.

Appendix G. Auxiliary Metric: Unique Motion Planner Calls

Fig. 14 provides the accumulated unique calls to the motion planner integrated into ASP planning procedure, invoked for collision-free joining path checks.

NoLabels+CaseMemory makes substantially more motion-planner calls than the label-aware strategies, largely because poorer grounding produces more invalid object choices

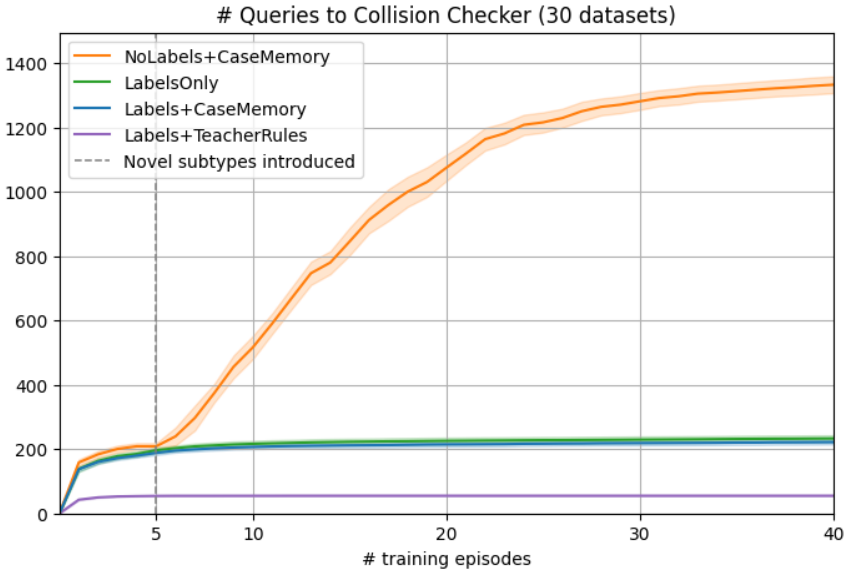


Figure 14: Accumulated counts of unique calls to the integrated motion planner.

and hence more replanning. Among the label-aware strategies, Labels+TeacherRules makes fewer calls because semantic constraints are incorporated already in the goal-selection ASP subproblem: object-role assignments that would violate known constraints are rejected or penalised before the join-sequence planner invokes collision checks. This can reduce calls to the motion planner when semantically invalid distractor choices would otherwise remain geometrically feasible. Since this metric is auxiliary and sensitive to implementation details of the planner and collision checker, we do not use it as primary evidence for the semantic-learning claim.