NESYCHAIN: A NEUROSYMBOLIC FRAMEWORK FOR LIFELONG LEARNING IN EMBODIED AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Embodied agents operate in changing open-world environments and therefore require lifelong learning. Recent work addresses this with neurosymbolic agents, whose combination of foundation models and symbolic tools makes learned knowledge explicit and verifiable. Yet in these agents, both reasoning and learning are tightly coupled to a particular configuration of foundation models, symbolic tools, and knowledge format, and must be rebuilt whenever it changes. We present NESYCHAIN, a modular framework for building neurosymbolic embodied agents whose reasoning and learning are decoupled from the configuration. Foundation models and symbolic tools are loosely coupled through a shared knowledge layer that mediates their interactions, so reasoning flows stay intact when components change. Specifically, for lifelong learning, NESYCHAIN employs two-stage knowledge transformation: acquired data is consolidated into symbolic knowledge through contrastive validation, then compiled into neural memory on the frozen model, reducing tool reliance. NESYCHAIN remains effective across 6 neurosymbolic configurations, spanning 4 foundation models, 9 symbolic tools, and 3 knowledge formats, on 3 benchmarks and in real-world settings, using a common modular structure and learning procedure. On every benchmark, it outperforms the strongest baseline, with 21.8% higher task success and 51.3% lower symbolic tool reliance on average. In real-world deployment, verifying learned knowledge before use raises task success from 33.3% to 96.7%.

1 INTRODUCTION

Embodied agents in changing open-world environments encounter new operating constraints, unexpected execution outcomes, and information from diverse sources after deployment, making lifelong learning an integral design requirement. Moving beyond monolithic policies, embodied agents increasingly combine foundation models, including language models (LMs), vision-language models (VLMs), and vision-language-action (VLA) models, with symbolic tools for planning, verification, and other structured operations. These designs yield diverse neurosymbolic configurations, each defined by its foundation models, symbolic tools, and knowledge format. Recent work tailors lifelong learning to predefined neurosymbolic configurations, providing a promising way to keep learned knowledge explicit and verifiable for reuse in reasoning throughout the agent's lifetime (Liang et al., 2024; Tang et al., 2024; Piriyakulkij et al., 2025).

However, existing approaches typically implement reasoning and learning flows ad hoc for each neurosymbolic configuration, without a common development structure. The reasoning flow requires custom logic specifying what knowledge is exchanged between components and how one component's outputs map to another's inputs. The learning flow likewise requires a custom procedure for transforming acquired data into knowledge for subsequent reasoning. Consequently, reasoning and learning remain tightly coupled to the chosen configuration, requiring both flows to be rebuilt when its components—foundation models, symbolic tools, or knowledge format—change.

To address this limitation, we present NESYCHAIN, a modular framework for building neurosymbolic embodied agents that decouples reasoning and learning from any particular neurosymbolic configuration. Rather than directly coupling foundation models and symbolic tools, NESYCHAIN mediates their interactions through a shared knowledge layer. Reasoning flows are structured around operations on shared knowledge rather than direct output-to-input mappings between components.

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

This design is implemented through three *Modular Interfaces*. The Model and Tool Interfaces standardize foundation model and symbolic tool invocation, respectively, while the Knowledge Interface abstracts differences among knowledge formats through common read and write operations. Component-specific implementation details are confined to these interfaces, allowing reasoning flows to remain intact when components change.

Through the Knowledge Interface, NESYCHAIN implements a two-stage knowledge transformation for reliable lifelong learning, comprising (i) *Knowledge Consolidation* and (ii) *Knowledge Compilation*. Process (i) continually consolidates acquired data into persistent symbolic knowledge through contrastive validation, refining symbolic elements so that the resulting knowledge reproduces valid traces and rejects invalid ones. This symbolic knowledge can be logically verified before use in subsequent tasks, reducing the risk that erroneous updates propagate to future behavior. Process (ii) compiles consolidated knowledge into efficient neural memory with the target foundation model frozen. By making consolidated knowledge directly reusable during inference, neural memory progressively reduces symbolic tool reliance. Together, these processes combine verifiable symbolic learning with direct neural reuse, providing a common learning procedure across configurations.

We evaluate NESYCHAIN on three embodied benchmarks (LIBERO-PRO (Zhou et al., 2025), CaP-Bench (Fu et al., 2026), and CL-ALFWorld) and in three real-world scenarios. Across these settings, we instantiate six neurosymbolic configurations within NESYCHAIN, spanning four foundation models, nine symbolic tools, and three knowledge formats, while retaining a common modular structure and learning procedure. On every benchmark, NESYCHAIN outperforms the strongest baseline, with 21.8% higher task success and 51.3% lower symbolic tool reliance on average (Table 1). Real-world experiments with a Franka Research 3 (FR3) robot further demonstrate reliable lifelong learning from diverse post-deployment data. Post-deployment learning with knowledge verified before use raises task success from 33.3% with prior knowledge alone to 96.7% (Figure 4).

We make the following contributions:

- We present NESYCHAIN, a modular framework for building neurosymbolic embodied agents that decouples reasoning and learning from specific configurations through a shared knowledge layer.
- We develop a common two-stage knowledge transformation for reliable lifelong learning that consolidates acquired data into symbolic knowledge and then compiles it into neural memory.
- We validate NESYCHAIN across benchmarks and real-world settings, showing broad applicability, improved task success, reduced symbolic tool reliance, and reliable post-deployment learning.

2 RELATED WORK

Neurosymbolic AI for embodied agents. Neurosymbolic AI couples neural models with symbolic reasoning (Besold et al., 2021; Yu et al., 2023; Wang et al., 2024). Recent work on neurosymbolic embodied agents studies how foundation models and symbolic tools can be coordinated to solve complex embodied tasks (Hsu et al., 2023; Wang et al., 2023b; Yang et al., 2025). Research on symbolic knowledge acquisition includes state abstraction, which maps perceptual observations to predicates (Li & Silver, 2023; Silver et al., 2023; Liang et al., 2024), and dynamics abstraction, which derives action models from agent experience (Ahmetoglu et al., 2022; Chitnis et al., 2022; Tang et al., 2024). While these studies advance reasoning and learning within particular configurations, NESYCHAIN provides a common development structure that can accommodate such configuration-specific methods across diverse neurosymbolic configurations and support reliable lifelong learning.

Modular frameworks for AI agents. Modular agent frameworks provide composable components and configurable workflows to coordinate foundation models, tools, and agents in multi-step reasoning and task execution (Chase, 2022; Wu et al., 2023; LangChain Team, 2024). Recent robotics frameworks and harnesses similarly encapsulate robot policies and control primitives as modular components for high-level planning and closed-loop execution (Guan et al., 2025; OpenRAL Contributors, 2026; Chen et al., 2026; Zhang et al., 2026). However, modular invocation and coordination interfaces do not by themselves provide a common structure for updating, validating, and compiling knowledge for diverse neurosymbolic configurations. NESYCHAIN complements these capabilities with a shared knowledge layer that mediates interactions between foundation models and symbolic tools across knowledge formats, supporting both reasoning and learning.

Lifelong learning for embodied agents. Lifelong learning enables embodied agents to retain and reuse knowledge acquired after deployment (Kasaei et al., 2021; Zhu et al., 2024; Zheng et al., 2026).

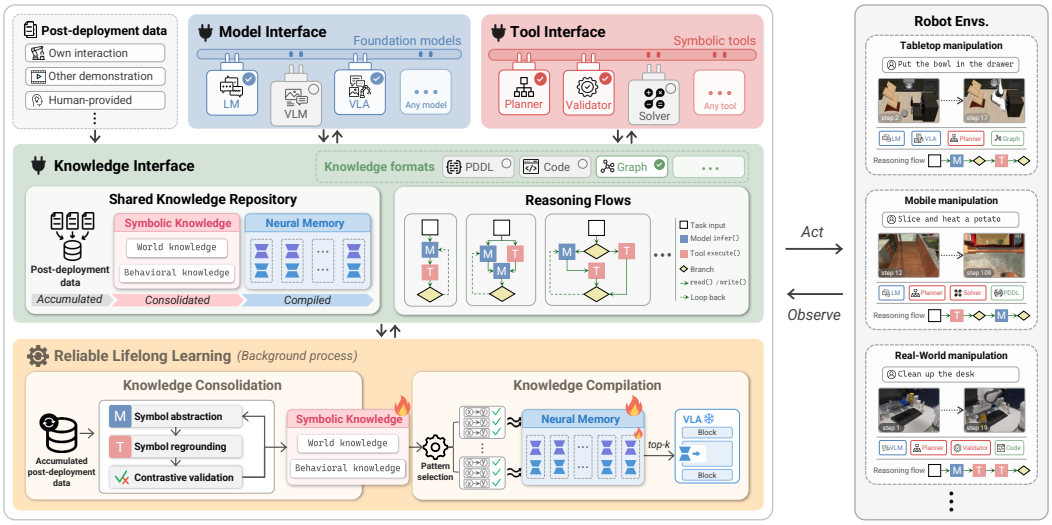


Figure 1: Overview of NESYCHAIN. The left shows *Modular Interfaces* supporting decoupled reasoning and lifelong learning, and the right shows deployment across neurosymbolic configurations.

Existing methods broadly retain acquired knowledge implicitly in learned parameters or explicitly in non-parameterized forms (Zheng et al., 2025; Hou et al., 2026). Parameter-based methods update model or adapter parameters, making acquired knowledge directly reusable during inference (Liu et al., 2023b; Kim et al., 2024; Wang et al., 2026). Non-parameterized methods retain knowledge in explicit forms that remain accessible as knowledge accumulates (Sarch et al., 2023; Wang et al., 2023a; Tziafas & Kasaei, 2024). Rather than choosing either form, NESYCHAIN combines both to balance verifiability and efficient reuse, consolidating acquired data into persistent symbolic knowledge for verification before compiling it into neural memory for direct reuse during inference.

3 NESYCHAIN

As shown in Figure 1, we present NESYCHAIN, a framework for instantiating diverse neurosymbolic configurations through three *Modular Interfaces*: the Model Interface, the Tool Interface, and the intermediate Knowledge Interface. Within reasoning flows, foundation models and symbolic tools interact through shared knowledge accessed via the Knowledge Interface, while accumulated data and knowledge are maintained in a shared knowledge repository. The Knowledge Interface embeds two background processes that together provide a common lifelong learning procedure across configurations. (i) *Knowledge Consolidation* transforms accumulated data into persistent symbolic knowledge through contrastive validation, and (ii) *Knowledge Compilation* compiles the consolidated knowledge into efficient neural memory while keeping the target foundation model frozen.

3.1 MODULAR INTERFACES

NESYCHAIN defines three standardized interfaces that expose common operations while encapsulating component-specific mechanisms.

Model and Tool Interfaces. The Model Interface supports foundation models such as LMs, VLMs, and VLAs by encapsulating model-specific inference behind a standardized `infer()` operation. The Tool Interface supports symbolic tools including planners, validators, and constraint solvers by encapsulating tool-specific execution behind a standardized `execute()` operation.

Knowledge Interface. The Knowledge Interface encapsulates format-specific access and update mechanisms behind common `read()` and `write()` operations, through which reasoning flows access accumulated knowledge and record deployment data and reasoning outcomes. Reasoning flows are thus structured around standardized component operations coordinated through shared knowledge rather than component-specific output-to-input mappings. Component-specific changes are handled within the corresponding interfaces, allowing reasoning flows to remain unchanged.

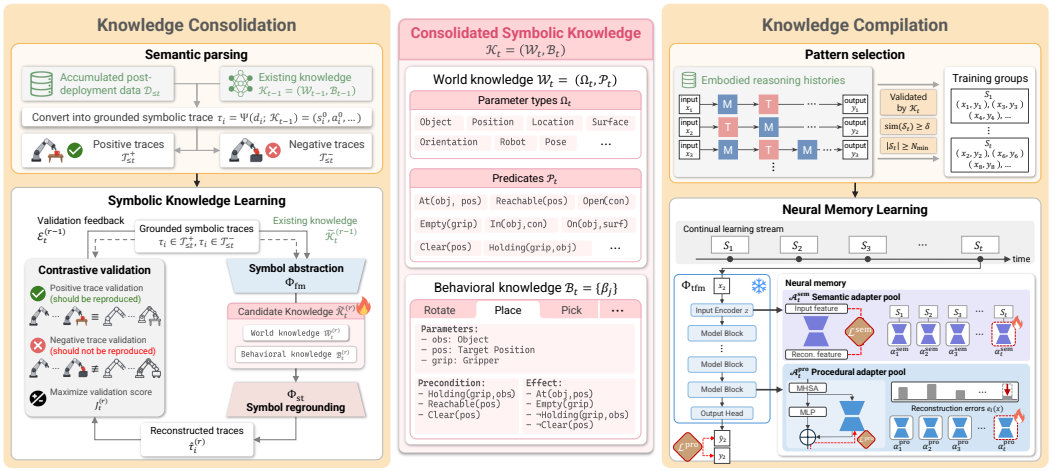


Figure 2: *Knowledge Consolidation* (left) and *Knowledge Compilation* (right) in NESYCHAIN.

3.2 KNOWLEDGE CONSOLIDATION

At the t-th consolidation, NESYCHAIN refines symbolic knowledge $\mathcal{K}_{t-1}$ using accumulated deployment data $\mathcal{D}_{\leq t}$. We describe the result as $\mathcal{K}_t = (\mathcal{W}_t, \mathcal{B}_t)$, regardless of knowledge format. World knowledge $\mathcal{W}_t = (\Omega_t, \mathcal{P}_t)$ comprises parameter types Ω_t and lifted predicates $\mathcal{P}_t$ representing environment properties and relations. Behavioral knowledge $\mathcal{B}_t = \{\beta_j\}$ comprises action schemas $\beta_j = \langle n_j(\mathbf{x}_j), \mathrm{Pre}_j, \mathrm{Eff}_j \rangle$ with typed signatures and lifted preconditions and effects over $\mathcal{P}_t$. A semantic parser Ψ, implemented by a foundation model or symbolic tool, maps each $d_i \in \mathcal{D}_{\leq t}$ into a grounded symbolic trace $\tau_i = \Psi(d_i; \mathcal{K}_{t-1}) = (s_i^0, a_i^0, \ldots, a_i^{L_i-1}, s_i^{L_i})$, where s_i^ℓ is a set of ground facts, a_i^ℓ a ground action, and L_i the number of transitions. Each trace receives a label $y_i \in \{0, 1\}$ from available feedback or trace-based estimation, indicating whether the trace is treated as valid ($y_i = 1$) or invalid ($y_i = 0$) during consolidation. These labels define positive and negative trace sets:

$$\mathcal{T}^+_{\leq t} = \{\tau_i \mid d_i \in \mathcal{D}_{\leq t},\ y_i = 1\}, \qquad \mathcal{T}^-_{\leq t} = \{\tau_i \mid d_i \in \mathcal{D}_{\leq t},\ y_i = 0\}. \tag{1}$$

Symbol abstraction. This step lifts grounded symbolic traces into candidate symbolic knowledge $\widetilde{\mathcal{K}}_t^{(r)}$, where r indexes refinement iterations and $\widetilde{\mathcal{K}}_t^{(0)} = \mathcal{K}_{t-1}$. For $r \geq 1$, a foundation model Φ_{fm} introduces or revises parameter types, predicates, and action schemas guided by its pretrained priors:

$$\widetilde{\mathcal{K}}_t^{(r)} = (\widetilde{\mathcal{W}}_t^{(r)}, \widetilde{\mathcal{B}}_t^{(r)}) = \Phi_{\mathrm{fm}}(\mathcal{T}^+_{\leq t}, \mathcal{T}^-_{\leq t}; \widetilde{\mathcal{K}}_t^{(r-1)}, \mathcal{E}_t^{(r-1)}, \mathcal{D}_{\leq t}). \tag{2}$$

Here, $\mathcal{E}_t^{(r-1)}$ denotes validation feedback from the previous iteration, with $\mathcal{E}_t^{(0)} = \varnothing$. $\mathcal{D}_{\leq t}$ supplements grounded traces with semantic evidence for introducing new symbolic elements.

Symbol regrounding. This step regrounds $\widetilde{\mathcal{K}}_t^{(r)}$ against each trace to reconstruct its transitions. For new predicates in $\widetilde{\mathcal{P}}_t^{(r)}$, Ψ augments each s_i^ℓ with groundings from d_i. A symbolic tool Φ_{st} instantiates a matching action schema with the arguments of a_i^ℓ. If the schema's preconditions hold in s_i^ℓ, Φ_{st} applies the schema's effects to obtain the successor state $\widehat{s}_i^{\ell+1}$:

$$\widehat{s}_i^{\ell+1} = \Phi_{\mathrm{st}}(s_i^\ell, a_i^\ell; \widetilde{\mathcal{W}}_t^{(r)}, \widetilde{\mathcal{B}}_t^{(r)}). \tag{3}$$

If no matching schema applies, the transition is recorded as a failure. The successor states and failures obtained independently per state–action pair form the transition-wise reconstruction $\widehat{\tau}_i^{(r)}$.

Contrastive validation. This step scores $\widetilde{\mathcal{K}}_t^{(r)}$ by rewarding reconstruction of positive traces and rejection of negative traces. The reconstruction score $R(\tau_i, \widehat{\tau}_i^{(r)}) \in [0, 1]$ measures transition-wise agreement between $s_i^{\ell+1}$ and $\widehat{s}_i^{\ell+1}$, counting regrounding failures as mismatches. The contrastive validation score is defined as follows:

$$J_t^{(r)} \equiv J_t(\widetilde{\mathcal{K}}_t^{(r)}) = \frac{1}{|\mathcal{T}^+_{\leq t}|} \sum_{\tau_i \in \mathcal{T}^+_{\leq t}} R(\tau_i, \widehat{\tau}_i^{(r)}) + \lambda \frac{1}{|\mathcal{T}^-_{\leq t}|} \sum_{\tau_i \in \mathcal{T}^-_{\leq t}} \left[1 - R(\tau_i, \widehat{\tau}_i^{(r)})\right]. \tag{4}$$

Here, $\lambda \geq 0$ weights the negative trace term, and for $r \geq 1$, validation provides feedback $\mathcal{E}_t^{(r)}$ for the next abstraction iteration. After N iterations, we select $r^\star \in \operatorname{argmax}_{0 \leq r \leq N} J_t^{(r)}$ and set $\mathcal{K}_t = \widetilde{\mathcal{K}}_t^{(r^\star)}$.

3.3 KNOWLEDGE COMPILATION

As illustrated in Figure 2 (right), consolidated symbolic knowledge $\mathcal{K}_t = (\mathcal{W}_t, \mathcal{B}_t)$ is progressively compiled into neural memory $\mathcal{M}_t = (\mathcal{A}_t^{\text{sem}}, \mathcal{A}_t^{\text{pro}})$, comprising semantic and procedural adapter pools. Compilation targets relatively compact foundation models exposing internal representations and supporting adapter attachment and optimization. At each compilation, adapter pairs are trained and added only for reasoning patterns meeting the compilation criteria, while the target foundation model remains frozen. Knowledge revisions can be handled by replacing affected pairs. During inference, semantic adapters guide sparse activation of paired procedural adapters in the target model.

Pattern selection. During task execution, NESYCHAIN records foundation model and symbolic tool invocation histories as part of $\mathcal{D}_{\leq t}$. From these histories, we group input-output pairs (x, y) by similarity. A group $\mathcal{S}_t$ is selected for compilation if:

$$V((x,y);\mathcal{K}_t) = 1 \quad \forall (x,y) \in \mathcal{S}_t, \qquad \text{sim}(\mathcal{S}_t) \geq \delta, \qquad |\mathcal{S}_t| \geq N_{\min}, \tag{5}$$

where V is a binary validation function against $\mathcal{K}_t$ (e.g., forward checks using Eq. 3), sim measures within-group similarity, and δ and $N_{\min}$ are the similarity and group-size thresholds, respectively.

Neural memory. Given a selected group $\mathcal{S}_t$, we train a semantic adapter α^{sem} and a procedural adapter α^{pro}, adding them to $\mathcal{A}_t^{\text{sem}}$ and $\mathcal{A}_t^{\text{pro}}$, respectively. α^{sem} captures group-specific semantic patterns by reconstructing input embeddings and can be implemented as a lightweight autoencoder. α^{pro} captures output decision patterns by adapting the target foundation model to reproduce corresponding outputs and can use LoRA (Hu et al., 2022) modules on frozen feedforward layers within its model blocks. Let $z(x)$ denote the input embeddings and $\Phi_{\text{tfm}}(\cdot;\theta,\alpha^{\text{pro}})$ the target foundation model with parameters θ and procedural adapter α^{pro}. We use the original training objective of Φ_{tfm} as the output loss ℓ_{tfm} and define the adapter objectives as follows:

$$\mathcal{L}^{\text{sem}} = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{S}_t}\left[\|z(x) - \alpha^{\text{sem}}(z(x))\|_2^2\right], \quad \mathcal{L}^{\text{pro}} = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{S}_t}\left[\ell_{\text{tfm}}(\Phi_{\text{tfm}}(\cdot;\theta,\alpha^{\text{pro}});x,y)\right]. \tag{6}$$

We optimize only the new adapter parameters, keeping θ and existing adapters frozen.

Sparse activation. Given a new input x, we compute errors $e_i(x) = \|z(x) - \alpha_i^{\text{sem}}(z(x))\|_2^2$ for $i \in \{1,\ldots,I_t\}$ and activate the procedural adapters from the k lowest-error pairs:

$$\mathcal{I}_k(x) = \mathop{\arg\text{top-k}}_{i\in\{1,\ldots,I_t\}}\left[-e_i(x)\right], \qquad \hat{y} = \text{Generate}(\Phi_{\text{tfm}}(x;\theta,\{\alpha_i^{\text{pro}} \mid i \in \mathcal{I}_k(x)\})), \tag{7}$$

where I_t is the size of $\mathcal{A}_t^{\text{sem}}$ and $\hat{y}$ the generated output. Generate follows the target foundation model's generation procedure. We use $k=1$ by default (Römer et al., 2026), though routing can be extended to adapter mixtures (Wang et al., 2025).

Algorithm 1 Knowledge Consolidation

Deployment data $\mathcal{D}_{\leq t}$, existing knowledge $\mathcal{K}_{t-1}$

for $d_i \in \mathcal{D}_{\leq t}$ **do**
  $\tau_i \leftarrow \Psi(d_i;\mathcal{K}_{t-1})$
end for
Construct $\mathcal{T}_{\leq t}^+$ and $\mathcal{T}_{\leq t}^-$ by Eq. 1
$r \leftarrow 0$; $\widetilde{\mathcal{K}}_t^{(0)} \leftarrow \mathcal{K}_{t-1}$; $\mathcal{E}_t^{(0)} \leftarrow \varnothing$
$J_t^{(0)} \leftarrow J_t(\widetilde{\mathcal{K}}_t^{(0)})$ using Eq. 3 and Eq. 4
repeat
  $r \leftarrow r+1$
  Generate $\widetilde{\mathcal{K}}_t^{(r)}$ by Eq. 2
  for $\tau_i \in \mathcal{T}_{\leq t}^+ \cup \mathcal{T}_{\leq t}^-$ **do**
    Reconstruct $\widehat{\tau}_i^{(r)}$ under $\widetilde{\mathcal{K}}_t^{(r)}$ using Eq. 3
  end for
  Collect discrepancies in $\mathcal{E}_t^{(r)}$
  $J_t^{(r)} \leftarrow J_t(\widetilde{\mathcal{K}}_t^{(r)})$ by Eq. 4
until $r \geq N$
Select $r^\star \in \mathop{\text{argmax}}_{0\leq r\leq N} J_t^{(r)}$; $\mathcal{K}_t \leftarrow \widetilde{\mathcal{K}}_t^{(r^\star)}$

Algorithm 2 Knowledge Compilation

Deployment data $\mathcal{D}_{\leq t}$, consolidated knowledge $\mathcal{K}_t$, neural memory $\mathcal{M}_{t-1} = (\mathcal{A}_{t-1}^{\text{sem}}, \mathcal{A}_{t-1}^{\text{pro}})$, target foundation model Φ_{tfm} with parameters θ

$\mathcal{A}_t^{\text{sem}} \leftarrow \mathcal{A}_{t-1}^{\text{sem}}$; $\mathcal{A}_t^{\text{pro}} \leftarrow \mathcal{A}_{t-1}^{\text{pro}}$
Form new candidate training groups from $\mathcal{D}_{\leq t}$
for each group $\mathcal{S}_t$ satisfying Eq. 5 **do**
  Initialize α^{sem} and α^{pro}
  repeat
    Update α^{sem} to minimize $\mathcal{L}^{\text{sem}}$ by Eq. 6
  until $\mathcal{L}^{\text{sem}}$ converges
  repeat
    Update α^{pro} to minimize $\mathcal{L}^{\text{pro}}$ with frozen θ by Eq. 6
  until $\mathcal{L}^{\text{pro}}$ converges
  $\mathcal{A}_t^{\text{sem}} \leftarrow \mathcal{A}_t^{\text{sem}} \cup \{\alpha^{\text{sem}}\}$
  $\mathcal{A}_t^{\text{pro}} \leftarrow \mathcal{A}_t^{\text{pro}} \cup \{\alpha^{\text{pro}}\}$
end for
$\mathcal{M}_t \leftarrow (\mathcal{A}_t^{\text{sem}}, \mathcal{A}_t^{\text{pro}})$

4 EVALUATION

4.1 EXPERIMENTAL SETTING

Benchmarks. We evaluate NESYCHAIN on three embodied benchmarks spanning different observation modalities and control interfaces. LIBERO-PRO (Zhou et al., 2025) evaluates vision-based robotic manipulation with continuous control. We organize 160 task instances derived from the LIBERO (Liu et al., 2023a) task suites into a sequential task stream. CaP-Bench (Fu et al., 2026) evaluates visually grounded manipulation through executable code that composes perception and control primitives. Using its LIBERO-PRO integration, we construct a sequential task stream comprising 123 instances. We construct CL-ALFWorld from ALFWorld (Shridhar et al., 2021), which provides textual observations and high-level discrete actions, following the incremental-learning setup of CL-ALFRED (Kim et al., 2024). The resulting sequential stream contains 200 language-guided mobile manipulation tasks, emphasizing multi-step reasoning over object states and relations.

Baselines. We select modular agent frameworks and neurosymbolic approaches for each benchmark according to its observation modalities and control interfaces. The modular agent frameworks include OpenRAL (OpenRAL Contributors, 2026), HarnessVLA (Zhang et al., 2026), and LangChain (Chase, 2022). The neurosymbolic approaches include VisualPredicator (Liang et al., 2024), NeSyC (Choi et al., 2025), and WALL-E 2.0 (Zhou et al., 2026). We additionally consider benchmark-specific reference methods, including $\pi_{0.5}$ (Physical Intelligence et al., 2025) fine-tuned on LIBERO (Liu et al., 2023a) for LIBERO-PRO and CaP-Agent0 (Fu et al., 2026) for CaP-Bench.

Metrics. We divide each task stream into ten sequential blocks and evaluate task performance and symbolic tool reliance throughout lifelong learning. Final success rate (SR) averages block-wise task success rates after learning the full stream. Backward transfer (BWT) measures success rate changes on previously learned blocks after subsequent learning, while forward transfer (FWT) measures changes relative to initial performance on future blocks before they are learned (Lopez-Paz & Ranzato, 2017). Positive BWT and FWT indicate beneficial knowledge transfer. Final tool reliance (TR) averages block-wise fractions of reasoning that rely on symbolic tools after learning the full stream (Xu et al., 2025). Backward reliance reduction (BRR) and forward reliance reduction (FRR) measure reductions in symbolic tool reliance on previously learned and future blocks, respectively. Lower TR and positive BRR and FRR indicate reduced symbolic tool reliance and are interpreted jointly with task success to assess the effectiveness of *Knowledge Compilation* (Qian et al., 2025). We report SR and TR as percentages and the remaining metrics in percentage points (pp).

Implementations. We instantiate NESYCHAIN with benchmark-specific foundation models, symbolic tools, and knowledge formats. Across all benchmarks, we use GPT-5.6-Sol (medium) (OpenAI, 2026) for *Knowledge Consolidation*, while the benchmark-specific foundation models perform inference with compiled neural memory. For LIBERO-PRO, we use $\pi_{0.5}$ with a graph processing module (Hagberg et al., 2008), a task-and-motion planner (Garrett et al., 2020), and a symbolic planner (Helmert, 2006) over graph-based knowledge. For CaP-Bench, we use Qwen3.5-4B (Qwen Team, 2026) with a Python interpreter, a forward BFS planner (Alkhazraji et al., 2020), and an argument binder (Francés et al., 2018) over code-based knowledge. For CL-ALFWorld, we use Qwen3-4B-Instruct-2507 (Qwen Team, 2025) with a symbolic planner (Helmert, 2006) over PDDL-based knowledge (McDermott et al., 1998). Appendix A provides additional experimental details.

4.2 MAIN RESULT

Table 1 evaluates whether NESYCHAIN can instantiate neurosymbolic embodied agents and support lifelong learning across diverse domains and configurations of foundation models, symbolic tools, and knowledge formats. Across the three benchmarks, NESYCHAIN achieves an average SR of 85.09%, compared with 63.34% for the highest SR baseline on each benchmark. Its average TR is 46.23%, versus 97.53% for the highest SR baseline for which TR is defined on each benchmark. These results support broad applicability across domains, with higher task success and lower symbolic tool reliance. On LIBERO-PRO in **Table 1a**, NESYCHAIN achieves 75.00% SR versus 66.04% for HarnessVLA, with BWT higher by 16.43 pp. HarnessVLA uses accumulated memory for LM-based planning and VLA coordination, whereas NESYCHAIN uses learned symbolic knowledge to support planning and verification with symbolic tools. Maintaining this symbolic knowledge across tasks enables constraint checking and refinement, supporting reliable VLA execution. On

Table 1: Overall comparison of task performance and symbolic tool reliance across three embodied benchmarks. Results report mean and standard deviation across three random seeds.

(a) Results on LIBERO-PRO. "-" indicates that the metric is not applicable to the baseline.

LIBERO-PRO	Task Performance			Symbolic Tool Reliance		
Method	SR (↑)	BWT (↑)	FWT (↑)	TR (↓)	BRR (↑)	FRR (↑)
$\pi_{0.5}$ (LIBERO-FT)	47.50±3.12	0.00±0.00	0.00±0.00	-	-	-
OpenRAL	59.17±0.95	0.00±0.00	0.00±0.00	100.00±0.00	0.00±0.00	0.00±0.00
HarnessVLA	66.04±3.97	2.78±2.78	6.94±6.25	-	-	-
VisualPredicator	57.50±1.65	−3.24±6.22	3.01±5.82	100.00±0.00	0.00±0.00	0.00±0.00
NeSyC	51.25±2.17	8.80±1.45	2.78±3.67	100.00±0.00	0.00±0.00	0.00±0.00
NeSyChain (Ours)	**75.00**±3.13	**19.21**±4.24	**7.99**±4.43	**54.33**±9.54	**14.54**±7.98	**30.52**±1.77

(b) Results on CaP-Bench.

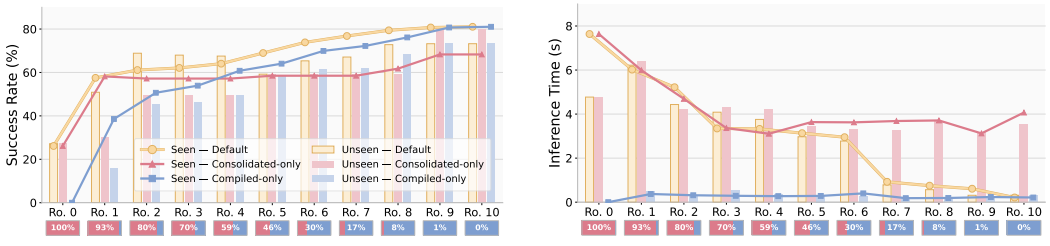
CaP-Bench	Task Performance			Symbolic Tool Reliance		
Method	SR (↑)	BWT (↑)	FWT (↑)	TR (↓)	BRR (↑)	FRR (↑)
CaP-Agent0	60.41±0.38	−2.83±2.88	−4.34±4.46	99.47±0.17	−0.51±1.63	0.70±0.61
LangChain	59.12±3.33	−0.64±3.14	0.28±1.11	100.00±0.00	0.00±0.00	0.00±0.00
NeSyC	57.71±3.27	1.76±1.99	−23.96±3.11	100.00±0.00	0.00±0.00	0.00±0.00
Wall-E 2.0	65.98±3.88	1.33±3.54	4.08±6.83	99.06±0.33	−1.46±2.55	−1.92±2.15
NeSyChain (Ours)	**88.95**±1.23	**5.06**±0.49	**7.31**±2.85	**17.14**±4.51	**5.90**±2.05	**64.54**±1.75

(c) Results on CL-ALFWorld.

CL-ALFWorld	Task Performance			Symbolic Tool Reliance		
Method	SR (↑)	BWT (↑)	FWT (↑)	TR (↓)	BRR (↑)	FRR (↑)
LangChain	27.33±2.89	−0.37±1.28	−0.56±1.11	96.90±0.56	−0.91±0.39	−0.50±0.35
NeSyC	54.33±1.76	−2.60±2.51	24.63±0.87	100.00±0.00	0.00±0.00	0.00±0.00
Wall-E 2.0	58.00±2.78	−0.19±0.85	26.67±4.01	93.53±0.82	0.78±0.99	2.00±0.28
NeSyChain (Ours)	**91.33**±0.29	**5.37**±0.32	**43.89**±0.00	**67.23**±0.40	**4.29**±0.66	**19.20**±0.59

CaP-Bench in **Table 1b**, NeSyChain achieves 88.95% SR with 17.14% TR and positive BRR and FRR, whereas WALL-E 2.0 records 65.98% SR with 99.06% TR and negative BRR and FRR. Both methods retain learned knowledge in symbolic form for reasoning. NeSyChain further compiles validated reasoning patterns into neural memory for direct reuse by the target foundation model, reducing symbolic tool reliance on future tasks. On CL-ALFWorld in **Table 1c**, NeSyChain achieves 91.33% SR versus 58.00% for WALL-E 2.0, with BWT and FWT higher by 5.56 pp and 17.22 pp, respectively. WALL-E 2.0 targets world-model adaptation by selecting rules to correct an LM's prediction errors. In contrast, NeSyChain refines symbolic knowledge to reconstruct valid traces and reject invalid ones across accumulated traces. NeSyChain supports the reuse and continual refinement of knowledge across tasks, consistent with its higher BWT and FWT.

4.3 Analysis and Ablation



(a) Task success rate across rounds. (b) Inference time across rounds.

Figure 3: Analysis of symbolic knowledge and neural memory throughout lifelong learning. Each round (Ro.) is a *Knowledge Consolidation–Compilation* cycle. Default adaptively combines consolidated symbolic knowledge (■) and compiled neural memory (■), with relative use shown below the x-axis. Consolidated-only and Compiled-only exclusively use the former and latter, respectively.

Analysis of symbolic knowledge and neural memory. Figure 3 analyzes the effects of consolidated symbolic knowledge and compiled neural memory on task performance and inference efficiency throughout lifelong learning. We extend the CL-ALFWorld setting to 306 learning tasks

(seen) and 228 held-out tasks (unseen). After each of 10 rounds, we evaluate both fixed task sets under Default, Consolidated-only, and Compiled-only inference. In Figure 3a, task success rate under Default inference improves steadily across rounds, while Consolidated-only inference improves earlier and Compiled-only inference catches up in later rounds. Under Default inference, task success rates rise from 26.14% and 27.19% to 81.05% and 73.25% on seen and unseen tasks, respectively. In Figure 3b, Default inference times approach those of Compiled-only inference, decreasing from 7.64s and 4.77s to 0.21s and 0.31s on seen and unseen tasks, respectively. These trends indicate complementary roles, with consolidated symbolic knowledge supporting early task execution and compiled neural memory enabling efficient knowledge reuse, including on unseen tasks.

Table 2: Analysis of modularity and lifelong learning across neurosymbolic configurations.

CL-ALFWorld	Reasoning Flow	Reimplemented with NeSyChain			Lifelong Learning Enabled		
Method	Strategy	SR (↑)	BWT (↑)	FWT (↑)	SR (↑)	BWT (↑)	FWT (↑)
NL-PDDL	Goal Regression	40.00±2.00	0.00±0.00	0.00±0.00	64.67±2.75	1.85±0.85	11.11±1.67
NeSyRo	Verified Exploration	41.00±2.60	0.00±0.00	0.00±0.00	73.00±1.73	11.11±0.96	21.30±0.64
CaSP	Code Refinement	44.17±2.02	0.00±0.00	0.00±0.00	66.67±0.76	9.07±2.10	12.22±1.11

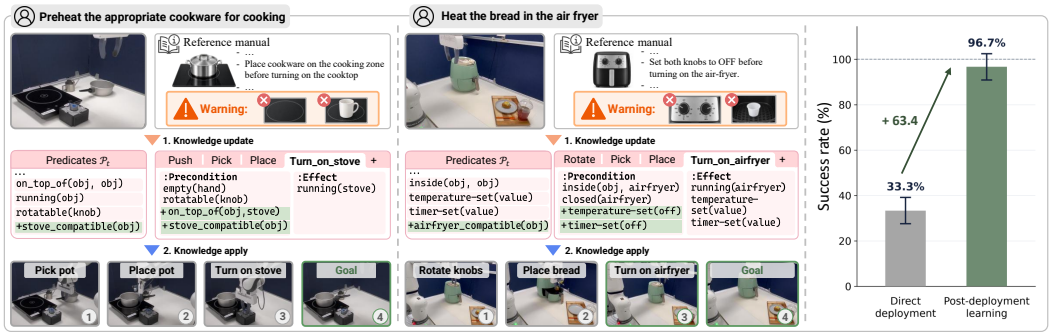
Analysis of modularity and lifelong learning. Table 2 evaluates whether NeSyChain can implement diverse neurosymbolic agents within a common development structure and support lifelong learning through a common procedure. We select NL-PDDL (Liu et al., 2026), NeSyRo (Ahn et al., 2025), and Code-as-Symbolic-Planner (CaSP) (Chen et al., 2025), which use distinct reasoning flows based on regression planning, recursively verified exploration, and iterative symbolic code refinement, respectively. Across these configurations, NeSyChain preserves each method's original reasoning flow, while the common lifelong learning procedure improves SR by 26.39% on average.

Table 3: Ablation study of (i) *Knowledge Consolidation* and (ii) *Knowledge Compilation*.

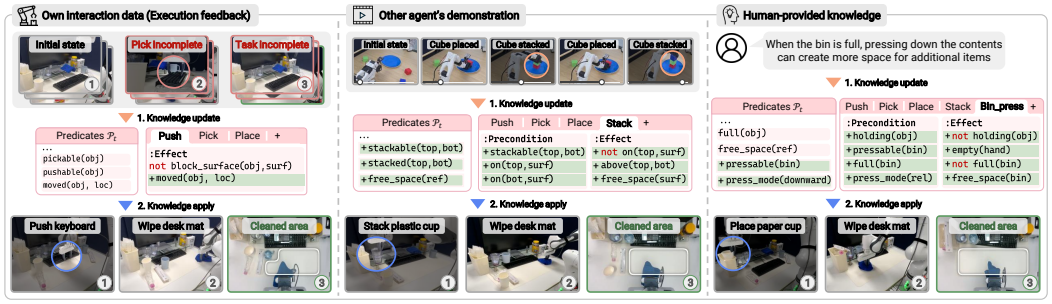
CL-ALFWorld	Task Performance			Symbolic Tool Reliance		
Method	SR (↑)	BWT (↑)	FWT (↑)	TR (↓)	BRR (↑)	FRR (↑)
NeSyChain (Ours)	**91.33**±0.29	**5.37**±0.32	**43.89**±0.00	**67.23**±0.40	**4.29**±0.66	**19.20**±0.59
NeSyChain w/o (i)	34.33±1.53	−0.74±0.64	0.56±0.56	78.12±0.21	2.82±0.07	12.79±0.47
NeSyChain w/o (ii)	88.67±1.61	4.07±1.40	**43.89**±2.00	100.00±0.00	0.00±0.00	0.00±0.00
NeSyChain w/o (i) & (ii)	33.67±1.15	0.00±0.00	0.00±0.00	100.00±0.00	0.00±0.00	0.00±0.00

Ablation study. Table 3 isolates the roles of the two background learning processes. NeSyChain w/o (i) reduces SR from 91.33% to 34.33% while fixing symbolic knowledge at its initial state. Positive FRR of 12.79 pp despite near-zero FWT of 0.56 pp indicates that (ii) reduces symbolic tool reliance on future blocks with little gain in future task success. NeSyChain w/o (ii) raises TR from 67.23% to 100% while reducing SR from 91.33% to 88.67%. Positive BWT of 4.07 pp and preserved FWT of 43.89 pp, alongside zero BRR and FRR, indicate that consolidated symbolic knowledge supports task performance transfer but does not reduce symbolic tool reliance on its own. Removing both components yields zero BWT and FWT with 100% TR.

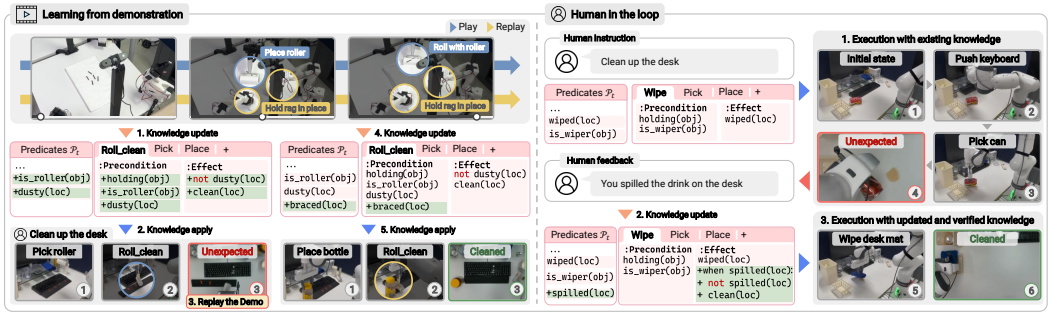
Real-world evaluation. Figure 4 evaluates post-deployment learning in NeSyChain, where newly acquired symbolic knowledge is verified before use in real-world task execution. We instantiate NeSyChain as in Table 1b, initialize it with knowledge acquired in simulation, and evaluate it in three real-world scenarios requiring additional knowledge. Across these scenarios, task success increases from 33.3% under direct deployment with prior knowledge alone to 96.7% after the full post-deployment learning procedure. In **Figure 4a**, using new appliances requires accounting for operating constraints not captured in the agent's existing knowledge. Using reference manuals, NeSyChain adds `stove_compatible(obj)` and `on_top_of(obj,stove)` as preconditions for stove activation, requiring compatible cookware to be on the cooking zone. For the air fryer, `temperature-set(off)` and `timer-set(off)` are added as preconditions requiring both knobs to be reset to OFF before activation. In **Figure 4b**, NeSyChain improves desk-cleaning quality through sequential learning from heterogeneous data sources. Feedback from its own execution refines `Push` by adding the effect `moved(obj, loc)`, enabling the agent to move aside an object it could not pick up. From another agent's demonstration, NeSyChain learns the `stackable` predicate and `Stack` action and transfers this knowledge from cubes to cups. Human-provided knowledge guides NeSyChain to add `Bin_press` with `free_space(bin)` as an effect, enabling the agent to create space in a full bin. Together, these updates progressively free more desk area for cleaning. In **Figure 4c**, unexpected execution outcomes reveal limitations



(a) Post-deployment scenarios where using new appliances requires accounting for operating constraints.



(b) Post-deployment scenarios where learning from heterogeneous data sources improves task quality.



(c) Post-deployment scenarios where agents recover from unexpected execution outcomes.

Figure 4: Real-world evaluation of post-deployment learning with knowledge verified before use.

of the existing knowledge used in the initial attempt. By revisiting a demonstration where two other robots hold a rag in place during roller cleaning, NeSyChain identifies `braced(loc)` as a missing precondition for `Roll_clean`. NeSyChain satisfies this precondition by placing a bottle as a stopper. In another case, human feedback introduces `spilled(loc)` and updates `Wipe` with conditional effects that set `not spilled(loc)` and `clean(loc)` when `spilled(loc)` holds, enabling recovery from an unexpected spill.

5 Conclusion

We present NeSyChain, a modular framework for neurosymbolic embodied agents that decouples reasoning and learning from specific neurosymbolic configurations through a shared knowledge layer. Its two-stage knowledge transformation supports reliable lifelong learning by consolidating acquired data into verifiable symbolic knowledge before compiling it into neural memory. Experiments demonstrate higher task success, lower symbolic tool reliance, and effective post-deployment learning across benchmarks and real-world settings. As illustrated in Figure 4, NeSyChain maintains learned knowledge in a unified symbolic format, but a limitation of this design is the potential synchronization costs when extended to multi-agent settings with asynchronously acquired context. Future work will explore maintaining agent-specific symbolic knowledge independently and flexibly composing world and action knowledge across agents when collaboration is required.

AI USE STATEMENT

Generative AI tools were used only in an assistive capacity, while all research decisions remained solely the responsibility of the authors. During the research process, they were used to obtain feedback on methodological soundness and experimental thoroughness. During implementation, they were used to assist with writing and debugging code and identifying potential implementation issues. During manuscript preparation, they were used to improve grammar, wording, and readability, including paraphrasing individual sentences. All AI-assisted outputs were reviewed and verified by the authors, who take full responsibility for the final content.

ETHICS STATEMENT

Our experiments do not involve human subjects or the collection of personal or sensitive data. The real-world robot experiments were conducted in controlled settings with appropriate safety precautions. Our evaluation focuses on simulated environments and controlled real-world settings, while deployment in unconstrained real-world settings may introduce physical safety risks and should therefore be accompanied by appropriate validation and safeguards.

REPRODUCIBILITY STATEMENT

Additional details of the simulation evaluation are provided in Appendix A, while the real-world experimental setup and evaluation procedure are described in Appendix B. We also provide supplementary materials with the submission to support reproduction of our experiments.

REFERENCES

Alper Ahmetoglu, M Yunus Seker, Justus Piater, Erhan Oztop, and Emre Ugur. Deepsym: Deep symbol generation and rule learning for planning from unsupervised robot interaction. *Journal of Artificial Intelligence Research*, 75:709–745, 2022.

Sanghyun Ahn, Wonje Choi, Junyong Lee, Jinwoo Park, and Honguk Woo. Towards reliable code-as-policies: A neuro-symbolic framework for embodied task planning. In *Advances in Neural Information Processing Systems*, volume 38, pp. 84000–84031, 2025. doi: 10.52202/085713-2533.

Yusra Alkhazraji, Matthias Frorath, Markus Grützner, Malte Helmert, Thomas Liebetraut, Robert Mattmüller, Manuela Ortlieb, Jendrik Seipp, Tobias Springenberg, Philip Stahl, and Jan Wülfing. Pyperplan. `https://doi.org/10.5281/zenodo.3700819`, 2020. URL `https://doi.org/10.5281/zenodo.3700819`.

Tarek R. Besold, Artur d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luis C. Lamb, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. Neural-symbolic learning and reasoning: A survey and interpretation. In Pascal Hitzler and Md Kamruzzaman Sarker (eds.), *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pp. 1–51. IOS Press, 2021.

Nicolas Carion, Laura Gustafson, Yuan-Ting Hu, et al. Sam 3: Segment anything with concepts. In *International Conference on Learning Representations*, 2026.

Harrison Chase. LangChain, 2022. URL `https://github.com/langchain-ai/langchain`.

Siyi Chen, Hugo Hadfield, Alex Zook, Mikaela Angelina Uy, Chan Hee Song, Erwin Coumans, Xuning Yang, Faisal Ladhak, Qing Qu, Stan Birchfield, et al. Volo: A physical orchestrator for open-vocabulary long-horizon manipulation. *arXiv preprint arXiv:2606.07723*, 2026.

Yongchao Chen, Yilun Hao, Yang Zhang, and Chuchu Fan. Code-as-symbolic-planner: Foundation model-based robot planning via symbolic code generation. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 19248–19254. IEEE, 2025. doi: 10.1109/IROS60139.2025.11247174.

Rohan Chitnis, Tom Silver, Joshua B Tenenbaum, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *2022 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 4166–4173. IEEE, 2022.

Wonje Choi, Jinwoo Park, Sanghyun Ahn, Daehee Lee, and Honguk Woo. Nesyc: A neuro-symbolic continual learner for complex embodied tasks in open domains. *arXiv preprint arXiv:2503.00870*, 2025.

Leonardo de Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337–340. Springer, 2008. doi: 10.1007/978-3-540-78800-3_24.

Hao-Shu Fang, Chenxi Wang, Hongjie Fang, Minghao Gou, Jirong Liu, Hengxu Yan, Wenhai Liu, Yichen Xie, and Cewu Lu. Anygrasp: Robust and efficient grasp perception in spatial and temporal domains. *IEEE Transactions on Robotics*, 39(5):3929–3945, 2023.

Guillem Francés, Miquel Ramirez, and Collaborators. Tarski: An AI planning modeling framework. `https://github.com/aig-upf/tarski`, 2018.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Guanzhi Wang, Dantong Niu, Fei-Fei Li, Guanya Shi, Jiajun Wu, Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. CaP-X: A framework for benchmarking and improving coding agents for robot manipulation. In *Proceedings of the 43rd International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2026.

Caelan Reed Garrett, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 440–448, 2020.

Weifan Guan, Qinghao Hu, Huasen Xi, Chenxiao Zhang, Aosheng Li, and Jian Cheng. Roboneuron: A middle-layer infrastructure for agent-driven orchestration in embodied ai. *arXiv preprint arXiv:2512.10394*, 2025.

Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In *Proceedings of the 7th Python in Science Conference*, pp. 11–15, 2008. doi: 10.25080/TCWV9851. URL `https://doi.org/10.25080/TCWV9851`.

Malte Helmert. The fast downward planning system. *Journal of Artificial Intelligence Research*, 2006.

Zhiyan Hou, Dan Zhang, Tao Feng, Liyuan Wang, Wei Li, Xiangzhao Hao, Hongyan An, Junfeng Fang, Haokai Ma, Zhaohui Xu, et al. Continual learning in transition. *arXiv preprint arXiv:2608.06216*, 2026.

Richard Howey, Derek Long, and Maria Fox. Val: Automatic plan validation, continuous effects and mixed initiative planning using pddl. In *16th IEEE International Conference on Tools with Artificial Intelligence*, 2004.

Joy Hsu, Jiayuan Mao, Josh Tenenbaum, and Jiajun Wu. What's left? concept grounding with logic-enhanced foundation models. *Advances in Neural Information Processing Systems*, 36: 38798–38814, 2023.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *ICLR*, 2022.

S Hamidreza Kasaei, Jorik Melsen, Floris van Beers, Christiaan Steenkist, and Klemen Voncina. The state of lifelong learning in service robots: Current bottlenecks in object perception and manipulation. *Journal of Intelligent & Robotic Systems*, 103(1):8, 2021.

Byeonghwi Kim, Minhyuk Seo, and Jonghyun Choi. Online continual learning for interactive instruction following agents. In *International Conference on Learning Representations*, 2024.

LangChain Team. LangGraph, 2024. URL https://github.com/langchain-ai/langgraph.

Amber Li and Tom Silver. Embodied active learning of relational state abstractions for bilevel planning. *arXiv preprint arXiv:2303.04912*, 2023.

Yichao Liang, Nishanth Kumar, Hao Tang, Adrian Weller, Joshua B Tenenbaum, Tom Silver, João F Henriques, and Kevin Ellis. Visualpredicator: Learning abstract world models with neuro-symbolic predicates for robot planning. *arXiv preprint arXiv:2410.23156*, 2024.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. LIBERO: Benchmarking knowledge transfer for lifelong robot learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 44776–44791, 2023a.

Xiaotian Liu, Armin Toroghi, Jiazhou Liang, David Courtis, Ruiwen Li, Ali Pesaranghader, Jaehong Kim, Tanmana Sadhu, Hyejeong Jeon, and Scott Sanner. Natural language pddl (nl-pddl) for open-world goal-oriented commonsense regression planning in embodied ai. In *The Fourteenth International Conference on Learning Representations*, 2026.

Zuxin Liu, Jesse Zhang, Kavosh Asadi, Yao Liu, Ding Zhao, Shoham Sabach, and Rasool Fakoor. Tail: Task-specific adapters for imitation learning with large pretrained models. *arXiv preprint arXiv:2310.05905*, 2023b.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30, 2017.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL — the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

OpenAI. GPT-4.1 Mini. https://developers.openai.com/api/docs/models/gpt-4.1-mini, 2025. Accessed: 2026-09-11.

OpenAI. GPT-5.6 Sol, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-sol. Accessed: 2026-08-25.

OpenRAL Contributors. OpenRAL: Open-Source Robot Agentic Layer. https://github.com/OpenRAL/openral, 2026. Accessed: 2026-09-11.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

Wasu Top Piriyakulkij, Yichao Liang, Hao Tang, Adrian Weller, Marta Kryven, and Kevin Ellis. Poe-world: Compositional world modeling with products of programmatic experts. *Advances in Neural Information Processing Systems*, 38:26609–26638, 2025.

Cheng Qian, Emre Can Acikgoz, Hongru Wang, Xiusi Chen, Avirup Sil, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Smart: Self-aware agent for tool overuse mitigation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 4604–4621, 2025.

Qwen Team. Qwen3-4B-Instruct-2507, 2025. URL https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507. Accessed: 2026-08-25.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026. URL https://qwen.ai/blog?id=qwen3.5.

Ralf Römer, Yi Zhang, Yuming Li, and Angela P Schoellig. Clare: Continual learning for vision-language-action models via autonomous adapter routing and expansion. *IEEE Robotics and Automation Letters*, 2026.

Gabriel Sarch, Yue Wu, Michael Tarr, and Katerina Fragkiadaki. Open-ended instructable embodied agents with memory-augmented large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 3468–3500, 2023.

Mohit Shridhar, Xingdi Yuan, Marc-Alexandre Côté, Yonatan Bisk, Adam Trischler, and Matthew Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

Tom Silver, Rohan Chitnis, Nishanth Kumar, Willie McClinton, Tomás Lozano-Pérez, Leslie Kaelbling, and Joshua B Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 12120–12129, 2023.

Hao Tang, Darren Key, and Kevin Ellis. Worldcoder, a model-based llm agent: Building world models by writing code and interacting with the environment. In *Advances in Neural Information Processing Systems*, 2024.

Georgios Tziafas and Hamidreza Kasaei. Lifelong robot library learning: Bootstrapping composable and generalizable skills for embodied control with language models. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 515–522. IEEE, 2024.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023a.

Huiyi Wang, Haodong Lu, Lina Yao, and Dong Gong. Self-expansion of pre-trained models with mixture of adapters for continual learning. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10087–10098. IEEE, 2025.

Renhao Wang, Jiayuan Mao, Joy Hsu, Hang Zhao, Jiajun Wu, and Yang Gao. Programmatically grounded, compositionally generalizable robotic manipulation. *arXiv preprint arXiv:2304.13826*, 2023b.

Wenguan Wang, Yi Yang, and Fei Wu. Towards data-and knowledge-driven ai: a survey on neuro-symbolic computing. *IEEE transactions on pattern analysis and machine intelligence*, 47(2): 878–899, 2024.

Xudong Wang, Zebin Han, Zhiyu Liu, Gan Li, Jiahua Dong, Baichen Liu, Lianqing Liu, and Zhi Han. Lifelong language-conditioned robotic manipulation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 18629–18637, 2026.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Hongshen Xu, Zihan Wang, Zichen Zhu, Lei Pan, Xingyu Chen, Shuai Fan, Lu Chen, and Kai Yu. Alignment for efficient tool calling of large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 17776–17792, 2025.

Zhutian Yang, Caelan Garrett, Dieter Fox, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. Guiding long-horizon task and motion planning with vision language models. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 16847–16853. IEEE, 2025.

Dongran Yu, Bo Yang, Dayou Liu, Hui Wang, and Shirui Pan. A survey on neural-symbolic learning systems. *Neural Networks*, 166:105–126, 2023.

Yixian Zhang, Huanming Zhang, Feng Gao, Xiao Li, Zhihao Liu, Chunyang Zhu, Jiaxing Qiu, Yuchen Yan, Jiyuan Liu, Wenhao Tang, Zhengru Fang, Yi Nie, Changxu Wei, Yu Wang, Wenbo Ding, and Chao Yu. Harness vla: Steering frozen vlas into reliable manipulation primitives via memory-guided agents. *arXiv preprint arXiv:2607.08448*, 2026. URL https://arxiv.org/abs/2607.08448.

Junhao Zheng, Shengjie Qiu, Chengming Shi, and Qianli Ma. Towards lifelong learning of large language models: A survey. *ACM Computing Surveys*, 57(8):1–35, 2025.

Junhao Zheng, Chengming Shi, Xidi Cai, Qiuke Li, Duzhen Zhang, Chenxing Li, Dong Yu, and Qianli Ma. Lifelong learning of large language model based agents: A roadmap. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2026.

Siyu Zhou, Tianyi Zhou, Yijun Yang, Guodong Long, Deheng Ye, Jing Jiang, and Chengqi Zhang. Wall-e: World alignment by neurosymbolic learning improves world model-based llm agents. *Advances in Neural Information Processing Systems*, 38:65352–65391, 2026.

Xueyang Zhou, Yangming Xu, Guiyao Tie, Yongchao Chen, Guowen Zhang, Duanfeng Chu, Pan Zhou, and Lichao Sun. Libero-pro: Towards robust and fair evaluation of vision-language-action models beyond memorization. *arXiv preprint arXiv:2510.03827*, 2025.

Dekang Zhu, Qianyi Bu, Zhongpan Zhu, Yujie Zhang, and Zhipeng Wang. Advancing autonomy through lifelong learning: a survey of autonomous intelligent systems. *Frontiers in neurorobotics*, 18:1385778, 2024.

A EVALUATION

A.1 LIBERO-PRO

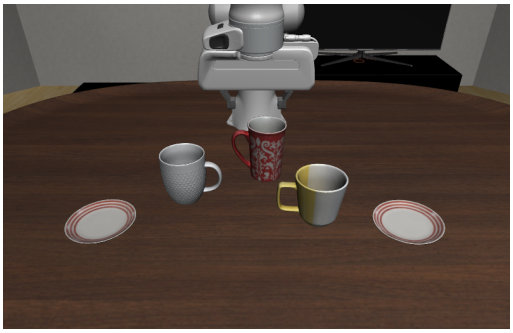
LIBERO-PRO (Zhou et al., 2025) is a benchmark for evaluating agents on continuous-control robotic manipulation tasks. Agents receive visual observations and natural-language instructions and execute continuous robot actions for tasks such as object placement and drawer opening. The benchmark evaluates robustness to controlled perturbations in object properties, initial states, task instructions, and environment configurations. We construct a sequential task stream of 160 LIBERO-PRO task instances derived from 58 underlying tasks instantiated under varying initial conditions. Figure 5 shows representative tasks from this stream.



(a) Put black bowl inside bottom drawer and close it.



(b) Switch the stove on and put the moka pot on it.



(c) Place white mug on left plate and yellow-white mug on right plate.



(d) Place yellow and white mug in microwave and close door.

Figure 5: Representative tasks from the LIBERO-PRO sequential task stream.

A.1.1 BASELINES

Baseline overview. We consider baselines from two categories, modular agent frameworks and neurosymbolic approaches. For modular agent frameworks, we include OpenRAL (OpenRAL Contributors, 2026) and HarnessVLA (Zhang et al., 2026), which augment VLA policies with reasoning and execution components for robotic manipulation. For neurosymbolic approaches, we include VisualPredicator (Liang et al., 2024) and NeSyC (Choi et al., 2025), which learn and use explicit symbolic knowledge for embodied reasoning and planning. We additionally include a LIBERO-finetuned $\pi_{0.5}$ policy as a VLA reference. Across the baselines and NESYCHAIN, when foundation model-based symbolic knowledge generation or refinement is required, we use GPT-5.6-Sol for those learning operations. Any method-specific use of foundation models for inference-time reasoning or action generation follows the corresponding baseline implementation and is described below.

Common evaluation protocol. For fair comparison, all methods are evaluated in the same LIBERO-PRO environments and designated initial states, with the same settling procedure applied before task execution. For all $\pi_{0.5}$-based methods, we use the same LIBERO-finetuned $\pi_{0.5}$ backbone checkpoint and provide external and wrist RGB observations, robot proprioceptive state, and language instructions through the same policy interface and preprocessing pipeline. At most five actions from each $\pi_{0.5}$ inference are executed before obtaining a new observation. The maximum environment-step budgets are 440, 560, 600, and 990 for the Spatial, Object, Goal, and Long-Horizon suites, respectively. Within each suite, all methods are subject to the same environment-step budget, and task success is determined using the benchmark's success criterion.

LIBERO-finetuned $\pi_{0.5}$ policy. The LIBERO-finetuned $\pi_{0.5}$ policy is a VLA model (Physical Intelligence et al., 2025) fine-tuned on LIBERO demonstrations (Liu et al., 2023a) that generates continuous action chunks from visual observations, proprioceptive state, and language instructions. We include this policy as a VLA reference for LIBERO-PRO, since all other $\pi_{0.5}$-based methods build on the same frozen checkpoint.

We use the publicly released `pi05_libero` checkpoint from OpenPI [1] and keep its parameters frozen throughout evaluation, without additional fine-tuning or updates. GPT-5.6-Sol is not used in this baseline. The corresponding hyperparameter settings are summarized in Table 4.

Since this baseline executes each task solely through the policy without using any symbolic tools, TR, BRR, and FRR are not applicable.

Table 4: Hyperparameter settings for the LIBERO-finetuned $\pi_{0.5}$ policy on LIBERO-PRO.

Hyperparameters	**Value**
Policy checkpoint	`pi05_libero`
Policy weights	Frozen
Input image resolution	224×224
Number of camera views	2
Proprioceptive state dimension	8
Action dimension	7
Control frequency	20 Hz
Action horizon	10
Executed actions per policy query	5
Initial settling steps	10
Evaluation seeds	17, 18, 19

OpenRAL. OpenRAL is a robot agent framework that follows a dual-system architecture in which a slow LM-based reasoner selects tools and robot skills while a fast visuomotor policy executes continuous robot actions (OpenRAL Contributors, 2026). We include OpenRAL as a modular robot-agent baseline because it augments a frozen VLA policy with external reasoning and tool use.

[1] `https://github.com/Physical-Intelligence/openpi`

Following OpenRAL's official implementation [2], we use GPT-4.1-mini (OpenAI, 2025) as the high-level reasoner and the LIBERO-finetuned $\pi_{0.5}$ policy as the visuomotor policy. GPT-5.6-Sol is not used in OpenRAL. For adaptation to LIBERO-PRO, we use a graph-based symbolic validation tool that grounds the observed state and checks the effects of executed skills, and a symbolic planner that plans subsequent skills and uses PDDLStream for motion grounding before robot execution. The hyperparameter settings are summarized in Table 5, and the visuomotor policy follows the same configuration as the $\pi_{0.5}$-only baseline in Table 4.

For symbolic tool reliance, use of the graph-based symbolic validation tool or the symbolic planner, including its PDDLStream-based motion grounding, is counted as symbolic tool use. Since symbolic reasoning is performed before every VLA call, TR remains 100%, with BRR and FRR equal to 0.

Table 5: Hyperparameter settings for OpenRAL on LIBERO-PRO.

Hyperparameters	**Value**
Sampling temperature (GPT-4.1-mini)	0.0
Max. generation tokens (GPT-4.1-mini)	API default
Max. agent iterations per reasoning step	4

HarnessVLA. HarnessVLA is a memory-guided agentic framework that combines a frozen VLA with a small fixed library of analytic manipulation primitives and uses an LM planner to compose them for long-horizon robot manipulation (Zhang et al., 2026). We include HarnessVLA as a modular agent baseline that augments a frozen VLA with a coding-agent planner, Task-Specific Memory, and Global Memory.

Following the original HarnessVLA design and its public implementation [3], we retain the planner, memory modules, and fixed primitive library. We instantiate the agentic planner with the Codex backend using GPT-5.6-Sol and expose the LIBERO-finetuned $\pi_{0.5}$ policy as `VLA_ACT`. The corresponding hyperparameter settings are summarized in Table 6.

For symbolic tool reliance, Task-Specific Memory and Global Memory provide context to the agentic planner, while the fixed manipulation primitives serve as execution primitives. Under our definition, these components are not counted as symbolic tool invocations. Accordingly, TR, BRR, and FRR are not applicable to HarnessVLA.

Table 6: Hyperparameter settings for HarnessVLA on LIBERO-PRO.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	Codex default
Max. accepted execution commands per episode	100

VisualPredicator. VisualPredicator learns neurosymbolic visual predicates and abstract action models from interaction data and uses the resulting world model for bilevel robot planning (Liang et al., 2024). We include VisualPredicator as a neurosymbolic learning baseline that acquires explicit symbolic abstractions for robot planning from visual interaction experience.

We use GPT-5.6-Sol for predicate proposal and GPT-4.1-mini for primitive visual predicate evaluation, while derived predicates are computed from other predicates. Following the planning setup of VisualPredicator, we use an A*-based symbolic planner (Alkhazraji et al., 2020) with a delete-relaxation heuristic, switching to a goal-count heuristic when derived predicates are present. For LIBERO-PRO, we adapt the perception, goal, and skill interfaces and carry the learned predicates, high-level action models, and transition data across sequential task blocks. The high-level actions selected by the symbolic planner are executed through predefined manipulation skills using the LIBERO-finetuned $\pi_{0.5}$ policy. The hyperparameter settings are summarized in Table 7.

[2] `https://github.com/OpenRAL/openral`

[3] `https://github.com/RLinf/RPent`

For symbolic tool reliance, high-level reasoning performed by the symbolic planner over learned predicates is counted as symbolic tool use. Since all high-level action selection is performed by the symbolic planner, TR remains 100%, with BRR and FRR equal to 0.

Table 7: Hyperparameter settings for VisualPredicator on LIBERO-PRO.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4096
Sampling temperature (GPT-4.1-mini)	0.0
Max. generation tokens (GPT-4.1-mini)	1024
Planning budget n_{abstract}	4
Max. predicate-invention iterations	1
Predicate simplicity weight α	-0.01
Satisficing-plan threshold	3

NeSyC. NeSyC is a neurosymbolic continual learning method that incrementally generalizes actionable knowledge through LM-guided hypothesis generation, contrastive symbolic validation, rule-based planning, and execution-driven knowledge refinement (Choi et al., 2025). We include NeSyC as a continual-learning baseline that explicitly updates and reuses symbolic action knowledge from interaction experience.

Following the NeSyC paper and official implementation [4], we preserve its ASP- and ILP-based rule representation, LM-guided rule generation and refinement, contrastive symbolic validation, and ASP-based planning. For LIBERO-PRO, visual observations and execution outcomes are converted into grounded ASP facts, while learned rules and transition data are retained across sequential task blocks. We use GPT-5.6-Sol to generate and refine ASP rules from accumulated interaction experience. During inference, the ASP planner applies the learned rules to select parameterized high-level actions, which are mapped to predefined manipulation skills and executed using the frozen LIBERO-finetuned $\pi_{0.5}$ policy. The corresponding hyperparameter settings are summarized in Table 8.

For symbolic tool reliance, high-level reasoning performed at inference time by the ASP planner over learned rules is counted as symbolic tool use. Since all online high-level action selection is performed through the ASP planner, TR remains 100%, with BRR and FRR equal to 0.

Table 8: Hyperparameter settings for NeSyC on LIBERO-PRO.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4096
ASP optimization mode	`optN`
ASP solver threads	1
ASP solver seeds	1, 2, 3

[4] `https://github.com/pjw971022/NeSyC-LLM`

A.1.2 NESYCHAIN IMPLEMENTATIONS

Neurosymbolic configuration. For LIBERO-PRO, we instantiate NESYCHAIN with a LIBERO-finetuned $\pi_{0.5}$ policy as the target foundation model for robot manipulation. As symbolic tools, we use NetworkX for graph processing, PDDLStream for task-and-motion planning, and Fast Downward for symbolic planning. During task-and-motion planning, geometric grounding uses the MuJoCo model for kinematic and collision checks together with inverse-kinematics computation. Symbolic knowledge is maintained in a graph-based format. World knowledge is represented by parameter types and predicates over scene-graph entities and relations, while the current scene instantiates these predicates over object instances. Behavioral knowledge is represented using PDDL-like typed action schemas with typed parameters, preconditions, and effects.

Reasoning flow. At each reasoning step, the current scene is grounded into the scene graph, and PDDLStream constructs a task-and-motion plan over the current predicates and action schemas, using Fast Downward for discrete planning. MuJoCo-based kinematics, inverse kinematics, and collision checking resolve geometric feasibility. The grounded symbolic actions are executed to establish the intermediate states required for subsequent manipulation. The symbolic plan itself is not provided to $\pi_{0.5}$. Instead, after symbolic execution updates the robot and scene state, the frozen policy observes the resulting state through its standard observation interface and continues manipulation from the updated observation. The observed outcome is grounded back into the symbolic state to verify the expected effects and trigger replanning when necessary.

Knowledge Consolidation. For LIBERO-PRO, *Knowledge Consolidation* constructs grounded symbolic traces from interaction trajectories and iteratively refines graph-based symbolic knowledge against these traces. To construct the traces, we segment each interaction trajectory around task-relevant manipulation events, such as changes in gripper state, object motion, symbolic relations, and goal state. Each segment is represented as a symbolic transition containing its prestate, executed action, poststate, and object bindings, and the transitions are assembled into a grounded symbolic trace that inherits the validity label of the source trajectory. Scene states are grounded using environment state information, including object relations, poses, attachment states, articulated states, and goal conditions.

Given the accumulated positive and negative traces, GPT-5.6-Sol performs symbol abstraction by proposing revisions to the current predicates and typed action schemas. At each refinement iteration, the model also receives validation feedback from the preceding iteration. When a candidate introduces or revises predicates, the recorded pre- and poststates are re-evaluated from the stored environment states under the candidate predicate definitions before transition reconstruction. Each candidate is then regrounded over the recorded scene states, where its action schemas are instantiated transition-wise and their preconditions and effects are applied to reconstruct the successor states. Differences between the reconstructed and recorded poststates are returned as validation feedback for the next refinement iteration. Following the contrastive validation procedure in Section 3.2, we select the highest-scoring knowledge state across the initial and refined candidates as the consolidated symbolic knowledge.

The hyperparameter settings for *Knowledge Consolidation* are summarized in Table 9.

Table 9: Hyperparameter settings for *Knowledge Consolidation* on LIBERO-PRO.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4096
Max. refinement iterations N	5
Negative trace weight λ	1.0

Knowledge Compilation. For LIBERO-PRO, we apply *Knowledge Compilation* to verified execution traces while keeping the LIBERO-finetuned $\pi_{0.5}$ backbone frozen. Compilation instances are collected from executions verified against the current consolidated symbolic knowledge. Symbolic planning and execution spans and subsequent frozen-VLA continuations are separated using the

execution boundaries recorded during symbolic execution. Each instance retains the grounded object bindings, the observation immediately preceding execution, the corresponding five-step action chunk, and the verified outcome. Instances supported by the same symbolic knowledge and exhibiting compatible execution contexts are grouped for compilation according to the pattern-selection criteria in Section 3.3.

For each selected group, the semantic adapter receives a mean-pooled representation of the frozen PaliGemma prefix in $\pi_{0.5}$ together with a binding-conditioned execution context comprising robot state, task-goal progress, and geometric and articulation information. The semantic adapter is trained to reconstruct this input representation, while the paired procedural adapter is attached to the frozen $\pi_{0.5}$ policy and optimized so that the policy generates the corresponding five-step continuous action chunk. Only the adapter parameters are updated, while the $\pi_{0.5}$ backbone remains frozen.

At inference, the semantic adapters are ranked by their reconstruction errors, and the procedural adapter paired with the lowest-error semantic adapter is activated following the sparse-activation procedure in Section 3.3 with $k = 1$. After the resulting action chunk is executed, the agent observes the updated state and repeats the inference procedure.

Neural memory grows incrementally as new adapter pairs are added for newly consolidated reasoning patterns. If a subsequent *Knowledge Consolidation* process revises the symbolic knowledge from which an existing adapter pair was constructed, the affected pair is identified through its association with that symbolic knowledge and grounded bindings, and is removed or retrained. Adapter pairs associated with unaffected symbolic knowledge are retained.

The hyperparameter settings for *Knowledge Compilation* are summarized in Table 10.

Table 10: Hyperparameter settings for *Knowledge Compilation* on LIBERO-PRO.

Hyperparameters	**Value**
Prefix pooling	Mean pooling
Prefix representation dimension	2048
Execution-context dimension	62
Semantic adapter input dimension	2110
Semantic adapter architecture (AE)	2110–128–32–128–2110
Action chunk length	5
Sparse activation k	1
Semantic adapter optimizer	Adam
Semantic adapter learning rate	1×10^{-3}
Semantic adapter training epochs	1,500
Procedural adapter optimizer	AdamW
Procedural adapter learning rate	3×10^{-4}
Procedural adapter weight decay	1×10^{-3}
Procedural adapter training epochs	4,000
Batch size	Full batch

A.2 CaP-Bench

CaP-Bench, a robotic manipulation benchmark in the CaP-X framework (Fu et al., 2026), is built on CaP-Gym, an interactive environment for code-based robot control. Agents receive natural-language task instructions and generate executable programs that compose perception and control primitives to solve robotic manipulation tasks. The benchmark evaluates whether these generated programs successfully complete the specified manipulation tasks. We use the LIBERO-PRO integration of CaP-Bench and construct a sequential task stream of 123 task instances across four LIBERO suites and four perturbation types, as summarized in Table 11. The stream excludes 37 of the original 160 candidate task instances without executable reference programs under the available CaP-Gym primitive interface. The stream is organized into 10 sequential blocks under a fixed task order, with 13 tasks in each of the first three blocks and 12 tasks in each of the remaining seven blocks. Figure 6 shows an example task execution.

Table 11: CaP-Bench sequential task stream across LIBERO suites and perturbation types.

Suite	Language	Object	Swap	Task	Total
LIBERO-Spatial	10	9	8	5	32
LIBERO-Object	10	10	10	10	40
LIBERO-Goal	7	3	5	9	24
LIBERO-10	8	6	6	7	27
Total	35	28	29	31	123

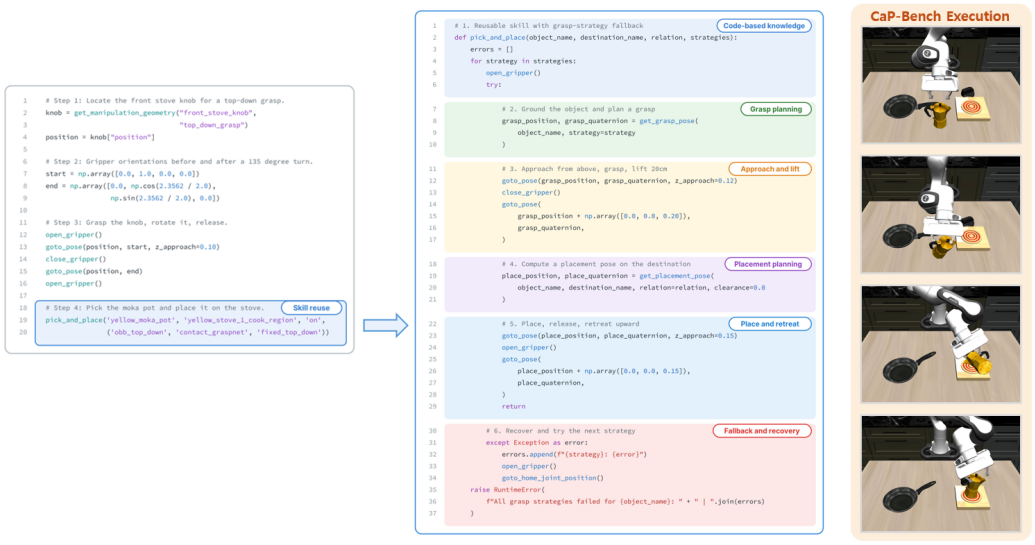


Figure 6: Example task execution in CaP-Bench.

A.2.1 BASELINES

Baseline overview. We consider baselines from two categories, modular agent frameworks and neurosymbolic approaches. For modular agent frameworks, we include LangChain (Chase, 2022), which provides reusable abstractions for composing foundation models and external tools. For neurosymbolic approaches, we include NeSyC (Choi et al., 2025) and WALL-E 2.0 (Zhou et al., 2026), which learn and use explicit symbolic knowledge for embodied reasoning and planning. We additionally include CaP-Agent0 (Fu et al., 2026), the reference agent introduced with CaP-Bench for code-based robot manipulation. Across the baselines and NESYCHAIN, when a learning procedure uses a foundation model to generate or refine an explicit symbolic knowledge representation, we use GPT-5.6-Sol for that operation. GPT-5.6-Sol is not used for inference-time reasoning or action generation in the CaP-Bench experiments.

Common evaluation protocol. For fair comparison, all methods are evaluated in the LIBERO-PRO Franka environment under CaP-Bench's M3 tier, a multi-turn setting with code execution feedback and natural-language visual feedback. At each turn, the agent receives the task instruction together with the feedback accumulated from preceding turns, but not raw image observations. All methods generate executable programs through the shared CaP-Gym interface using the same perception, geometric-reasoning, grasping, placement, and robot-control primitives. We retain the same grasp and manipulation corrections provided by the evaluation interface, including oracle-assisted fallback within the execution APIs. Each episode is limited to 20 turns and 4,000 environment steps, and task success is determined using the benchmark's success criterion.

CaP-Agent0. CaP-Agent0 is a training-free agentic framework introduced in CaP-X (Fu et al., 2026) that generates executable robot-control programs and iteratively refines them using visual execution feedback, an automatically synthesized skill library, and parallel ensemble reasoning. We include CaP-Agent0 as the native CaP-X agent baseline, providing a direct in-domain reference for code-based manipulation under the same perception and control interface as CaP-Bench.

Our implementation is based on the official CaP-X repository [5] and retains the original online skill-library mechanism. To use a consistent code generation model across CaP-Bench methods, we replace the original frontier model parallel reasoning configuration with Qwen3.5-4B. GPT-5.6-Sol is not used in CaP-Agent0. The corresponding hyperparameter settings are summarized in Table 12.

For symbolic tool reliance, we count reuse of executable functions from CaP-Agent0's maintained skill library as symbolic tool use, while benchmark-provided grasping, placement, and robot-control primitives used only for execution are excluded.

Table 12: Hyperparameter settings for CaP-Agent0 on CaP-Bench.

Hyperparameters	**Value**
Sampling temperatures (Qwen3.5-4B)	0.1–0.9 (step 0.1)
Max. generation tokens (Qwen3.5-4B)	512
Parallel reasoning branches	9

LangChain. LangChain is a modular agent framework that provides standardized abstractions for composing language models with external tools through model-directed tool calling (Chase, 2022). We include LangChain as a modular agent baseline to evaluate whether general-purpose model-directed tool use can support code-based robot manipulation without a specialized neurosymbolic reasoning or lifelong-learning architecture.

Following the official LangChain implementation [6], we expose the CaP-Bench APIs for perception, geometric reasoning, grasping, placement, robot control, and reusable skills as callable tools. Qwen3.5-4B selects and calls these tools from the task instruction and execution feedback to generate and execute programs. GPT-5.6-Sol is not used in LangChain. The corresponding hyperparameter settings are summarized in Table 13.

[5] https://github.com/capgym/cap-x

[6] https://github.com/langchain-ai/langchain

For symbolic tool reliance, perception and geometric-reasoning APIs exposed to the model as selectable external tools are counted as symbolic tool use, while grasping, placement, and robot-control APIs used only for execution are excluded. Since every reasoning step invokes at least one perception or geometric-reasoning API, TR remains 100%, with BRR and FRR equal to 0.

Table 13: Hyperparameter settings for LangChain on CaP-Bench.

Hyperparameters	**Value**
Sampling temperature (Qwen3.5-4B)	0.0
Max. generation tokens (Qwen3.5-4B)	512

NeSyC. NeSyC is a neurosymbolic continual learning method that incrementally induces and refines actionable symbolic rules through LM-guided hypothesis generation, contrastive symbolic validation, rule-based inference, and execution-driven feedback (Choi et al., 2025). We select NeSyC because its continual rule-learning and refinement mechanism provides a natural basis for adapting reusable symbolic action knowledge to CaP-Bench's code-based manipulation setting.

Our implementation is based on the NeSyC paper and official repository [7], preserving its ASP- and ILP-based rule representation, LM-driven hypothesis refinement, contrastive symbolic validation, and ASP-based planning. To adapt NeSyC to CaP-Bench, we map the benchmark observations and perception outputs into grounded ASP facts and deterministically translate the parameterized symbolic actions selected by the ASP planner into the corresponding CaP Python primitive calls. We use GPT-5.6-Sol for ASP rule generation and refinement and Qwen3.5-4B for execution-time error handling, while the Python programs serve only as the execution interface and the learned knowledge remains in the ASP rule representation. The corresponding hyperparameter settings are summarized in Table 14.

For symbolic tool reliance, ASP-based action selection is counted as symbolic tool use. Since all online action selection is performed through the ASP planner, TR remains 100%, with BRR and FRR equal to 0.

Table 14: Hyperparameter settings for NeSyC on CaP-Bench.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4096
Sampling temperature (Qwen3.5-4B)	0.0
Max. generation tokens (Qwen3.5-4B)	512
ASP optimization mode	`optN`
ASP solver threads	1

WALL-E 2.0. WALL-E 2.0 is a model-based LM agent that continually aligns an internal world model with observed environment dynamics by extracting symbolic knowledge from prediction errors and using the updated model for look-ahead decision making (Zhou et al., 2026). We include WALL-E 2.0 as a complementary neurosymbolic continual-learning baseline because its world-model adaptation mechanism can be applied to programmatic manipulation, where execution outcomes provide evidence for continually refining knowledge used in subsequent decision making.

Following the official WALL-E 2.0 implementation [8], we represent the learned world model in CaP-Bench as executable code-based rules over task-relevant states and action effects. Qwen3.5-4B proposes candidate action sequences and performs look-ahead over the full proposed sequence, while the symbolic world model evaluates predicted transitions. Selected actions are translated into the corresponding CaP Python primitive calls, and GPT-5.6-Sol updates the symbolic world model from accumulated prediction errors during learning. The corresponding hyperparameter settings are summarized in Table 15.

[7] `https://github.com/pjw971022/NeSyC-LLM`
[8] `https://github.com/elated-sawyer/WALL-E`

For symbolic tool reliance, look-ahead action selection that queries the symbolic world model is counted as symbolic tool use. Action selection performed by Qwen3.5-4B without querying the symbolic world model is not counted as symbolic tool use.

Table 15: Hyperparameter settings for WALL-E 2.0 on CaP-Bench.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4096
Sampling temperature (Qwen3.5-4B)	0.0
Max. generation tokens (Qwen3.5-4B)	512

A.2.2 NeSyChain Implementations

Neurosymbolic configuration. For CaP-Bench, we instantiate NeSyChain with Qwen3.5-4B as the target foundation model for code generation. As symbolic tools, we use a Python interpreter for executing code-based knowledge, a forward BFS planner implemented with Pyperplan (Alkhazraji et al., 2020), and a Tarski-based argument binder (Francés et al., 2018). Symbolic knowledge is maintained in a code-based format aligned with the programmatic manipulation interface of CaP-Bench. World knowledge is represented by parameter types and typed predicate definitions over task-relevant states and relations, with grounding functions used to evaluate these predicates from observations. Behavioral knowledge is represented by parameterized executable action definitions with typed parameters, applicability conditions, expected effects, and compositions of the available CaP perception and control primitives.

Reasoning flow. At each reasoning step, the current observation and task goal are interpreted using the executable predicate definitions to construct the grounded symbolic state. The argument binder resolves valid object bindings, and the forward BFS planner evaluates the applicability conditions and expected effects of the available action definitions to select the next manipulation. The selected action, grounded arguments, current reasoning context, relevant symbolic knowledge, and CaP API specification are then provided to Qwen3.5-4B, which generates the corresponding grounded Python program. Before execution, the generated program is checked for syntactic validity, permitted API usage, argument consistency, and agreement with the selected action definition. The program is then executed through the CaP interface. After execution, the resulting observation is interpreted using the same predicate definitions and compared with the expected effects of the selected action. When the expected transition is not observed, planning is repeated from the updated symbolic state.

Knowledge Consolidation. For CaP-Bench, *Knowledge Consolidation* operates over the symbolic states, selected actions, grounded bindings, generated programs, and execution outcomes recorded during reasoning. Each reasoning step is converted into a grounded symbolic transition by evaluating the executable predicate definitions before and after execution. The resulting transition preserves the selected action, object bindings, generated Python segment, executed API calls, and observed state change. Bindings are subsequently lifted according to their types and relational roles so that equivalent manipulation patterns can be compared across different objects and tasks.

Given the accumulated positive and negative traces, GPT-5.6-Sol performs symbol abstraction by proposing revisions to the current code-based symbolic knowledge. These revisions can modify predicate definitions, action applicability conditions, expected effects, or the primitive composition associated with an action while preserving the existing CaP API contract. At each refinement iteration, candidate definitions are evaluated in an isolated copy of the knowledge registry. When the candidate changes the predicate vocabulary, including by introducing new predicate definitions, all recorded states are re-evaluated under the candidate definitions before action grounding and transition reconstruction. The reconstructed transitions are compared with the observed transitions, and the resulting discrepancies are returned as validation feedback for the next refinement iteration. Following the contrastive validation procedure in Section 3.2, we select the highest-scoring knowledge state across the initial and refined candidates as the consolidated symbolic knowledge.

The hyperparameter settings for *Knowledge Consolidation* are summarized in Table 16.

Table 16: Hyperparameter settings for *Knowledge Consolidation* on CaP-Bench.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4096
Max. refinement iterations N	5
Negative trace weight λ	1.0

Knowledge Compilation. For CaP-Bench, *Knowledge Compilation* is applied to verified reasoning steps. Compilation instances associate each reasoning context with the corresponding symbolic knowledge, grounded bindings, and verified Python program. Only instances whose observed effects agree with the current consolidated knowledge are retained, and compatible instances supported by the same knowledge are grouped according to the pattern-selection criteria in Section 3.3.

For each selected group, a semantic adapter is trained to characterize the associated reasoning contexts through representation reconstruction, while its paired procedural adapter is implemented as a LoRA module on the frozen Qwen3.5-4B model and optimized so that the model generates the corresponding grounded Python program.

During inference, the semantic adapters are ranked by their reconstruction errors, and the procedural adapter paired with the lowest-error semantic adapter is activated following the sparse-activation procedure in Section 3.3 with $k = 1$. The generated program is then checked for syntax, API usage, binding consistency, and agreement with the selected action definition. If the generated program fails validation, reasoning falls back to planning over the current symbolic knowledge.

Neural memory grows incrementally as new adapter pairs are added for newly consolidated reasoning patterns. If a subsequent *Knowledge Consolidation* process revises the symbolic knowledge from which an existing adapter pair was constructed, the affected pair is identified through its association with that symbolic knowledge and grounded bindings, and is removed or retrained. Adapter pairs associated with unaffected symbolic knowledge are retained.

The hyperparameter settings for *Knowledge Compilation* are summarized in Table 17.

Table 17: Hyperparameter settings for *Knowledge Compilation* on CaP-Bench.

Hyperparameters	**Value**
Context representation dimension	2560
Semantic adapter architecture (AE)	2560–128–64–128–2560
Optimizer	AdamW
Semantic adapter learning rate	2×10^{-4}
Semantic adapter training epochs	300
Semantic adapter negative margin	0.25
LoRA rank / alpha	8 / 8
LoRA learning rate	5×10^{-5}
LoRA batch size	1
Minimum procedural adapter training steps	120
Max. training samples per memory unit	128
Gradient clipping (LoRA)	1.0
Sparse activation k	1

A.3 CL-ALFWORLD

CL-ALFWorld, our continual task extension of ALFWorld (Shridhar et al., 2021), is a text-based embodied benchmark for language-guided household tasks. Agents receive textual observations and natural-language instructions and execute high-level discrete actions to accomplish tasks involving object states and relations. The benchmark evaluates multi-step task planning over object states and relations. Following the incremental-learning setup of CL-ALFRED (Kim et al., 2024), we construct a sequential task stream of 200 language-guided mobile manipulation tasks. The stream is organized into 10 sequential blocks of 20 tasks each, with each block serving as the unit of incremental learning and evaluation. Figure 7 shows representative tasks from this stream.



(a) Slice potato, heat it, then store it in fridge. (b) Slice tomato, cool it, then place it on countertop.

Figure 7: Representative tasks from the CL-ALFWorld sequential task stream.

A.3.1 BASELINES

Baseline overview. We consider baselines from two categories, modular agent frameworks and neurosymbolic approaches. For modular agent frameworks, we include LangChain (Chase, 2022), which provides reusable abstractions for composing language models and external tools. For neurosymbolic approaches, we include NeSyC (Choi et al., 2025) and WALL-E 2.0 (Zhou et al., 2026), which learn and use explicit symbolic knowledge for embodied reasoning and planning. Across the baselines and NESYCHAIN, when a learning procedure requires foundation-model-based symbolic knowledge generation or refinement, we use GPT-5.6-Sol for that operation. GPT-5.6-Sol is not used for inference-time planning or action generation in CL-ALFWorld.

Common evaluation protocol. For fair comparison, all methods are evaluated on the same fixed CL-ALFWorld task stream using the same sequential block order. Textual observations and natural-language instructions are provided via the benchmark's text-based interaction interface. Task execution proceeds through high-level discrete actions supported by the ALFWorld action interface. For methods that perform continual updates, learning is synchronized with block boundaries after every 20 tasks.

LangChain. LangChain is a modular agent framework that provides standardized abstractions for composing language models with external tools through model-directed tool calling (Chase, 2022). We include it as a modular agent baseline that combines neural reasoning with symbolic tool use.

Our implementation follows the tool-calling pattern provided in the official LangChain repository [9]. We use Qwen3-4B-Instruct-2507 (Qwen Team, 2025) as the agent model and expose Fast Downward (Helmert, 2006) and a grounded PDDL precondition checker as callable symbolic tools. The PDDL domain remains fixed throughout evaluation, with no symbolic knowledge generation or refinement, model update, or memory retained across tasks. Accordingly, GPT-5.6-Sol is not used in this baseline. The corresponding hyperparameter settings are summarized in Table 18.

For symbolic tool reliance, Fast Downward and the grounded PDDL precondition checker exposed as selectable external tools are counted as symbolic tool use.

NeSyC. NeSyC is a neurosymbolic continual learning method that refines and reuses symbolic action knowledge through LM-guided hypothesis generation and contrastive validation (Choi et al.,

[9] https://github.com/langchain-ai/langchain

Table 18: Hyperparameter settings for LangChain on CL-ALFWorld.

Hyperparameters	Value
Sampling temperature (Qwen3-4B-Instruct-2507)	0.0
Max. generation tokens (Qwen3-4B-Instruct-2507)	Library default
Max. agent iterations per attempt	8
Max. attempts per reasoning step	3

2025). We include NeSyC as a continual-learning baseline that explicitly updates and reuses symbolic action knowledge from interaction experience.

Following the original method and official implementation [10], we adapt its ASP- and ILP-based representation to PDDL for CL-ALFWorld. At each block boundary, GPT-5.6-Sol (OpenAI, 2026) proposes revisions to PDDL operator preconditions from accumulated successful and failed transitions, which are validated for consistency and planner-based goal reachability. During inference, Fast Downward performs planning, while a grounded PDDL precondition checker validates action applicability. The corresponding hyperparameter settings are summarized in Table 19.

For symbolic tool reliance, inference-time use of Fast Downward or the grounded PDDL precondition checker is counted as symbolic tool use. Since all online action selection uses these symbolic tools, TR remains 100%, with BRR and FRR equal to 0.

Table 19: Hyperparameter settings for NeSyC on CL-ALFWorld.

Hyperparameters	Value
Rule-refinement reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens for rule refinement (GPT-5.6-Sol)	4,000

WALL-E 2.0. WALL-E 2.0 is a model-based LM agent that aligns its world model with environment dynamics by learning symbolic knowledge from discrepancies between predicted and observed transitions and uses the aligned world model for look-ahead action selection (Zhou et al., 2026). We use this method as a neurosymbolic continual-learning baseline that accumulates symbolic knowledge from interaction and applies it to planning over high-level discrete actions after deployment.

Our implementation is based on the WALL-E 2.0 paper and its official codebase [11]. We retain LM-based action proposal, one-step look-ahead, transition-mismatch-driven refinement, and reuse of the updated world model across subsequent tasks. The main changes are the use of PDDL to represent symbolic knowledge and transition dynamics and the synchronization of world-model updates with CL-ALFWorld block boundaries. At each block boundary, GPT-5.6-Sol refines the PDDL world model from accumulated prediction errors. During inference, Qwen3-4B-Instruct-2507 proposes actions, while Fast Downward and the PDDL world model support planning and one-step look-ahead before execution. The corresponding hyperparameter settings are summarized in Table 20.

For symbolic tool reliance, inference-time action selection that invokes Fast Downward or performs symbolic look-ahead over the PDDL world model is counted as symbolic tool use.

Table 20: Hyperparameter settings for WALL-E 2.0 on CL-ALFWorld.

Hyperparameters	Value
World-model refinement reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens for world-model refinement (GPT-5.6-Sol)	4,000
Sampling temperature (Qwen3-4B-Instruct-2507)	0.0
Look-ahead horizon	1 step

[10] https://github.com/pjw971022/NeSyC-LLM
[11] https://github.com/elated-sawyer/WALL-E

A.3.2 NESYCHAIN IMPLEMENTATIONS

Neurosymbolic configuration. For CL-ALFWorld, we instantiate NESYCHAIN with Qwen3-4B-Instruct-2507 as the target foundation model for grounded action generation. As the symbolic reasoning tool, we use Fast Downward for PDDL-based planning. We additionally use a deterministic PDDL precondition evaluator following standard PDDL validation semantics (Howey et al., 2004) as an execution safeguard to check the applicability of grounded actions in the current symbolic state. Symbolic knowledge is maintained in PDDL format. World knowledge is represented by object types and predicates describing task-relevant object properties, states, and relations, while behavioral knowledge is represented by typed PDDL action schemas with preconditions and effects.

Reasoning flow. At each reasoning step, the textual observation is mapped to grounded PDDL facts representing the current symbolic state. Together with the task goal, these facts define the current planning problem. For explicit reasoning, Fast Downward plans over the accumulated PDDL knowledge, and the resulting grounded action is deterministically converted into the corresponding executable ALFWorld command. As reusable reasoning patterns are compiled into neural memory, Qwen3-4B-Instruct-2507 can generate grounded actions directly without calling the symbolic planner. A generated action is accepted only if it has the required arguments, its preconditions hold in the current symbolic state, and it remains consistent with the task goal without returning to the current or a previously visited state. Otherwise, the agent falls back to Fast Downward. After execution, the new observation is mapped to an updated symbolic state, and the observed transition is checked against the selected action's preconditions and expected effects. Execution outcomes and transition mismatches are retained through the Knowledge Interface as evidence for subsequent knowledge updates.

Knowledge Consolidation. For CL-ALFWorld, *Knowledge Consolidation* operates over the discrete state and action transitions accumulated throughout the task stream. Interaction histories are converted into grounded symbolic traces by mapping the observations before and after each executed action to PDDL states together with the grounded action and its object bindings. We implement the semantic parser Ψ as a deterministic grounding module that maps the available environment state information to predicates in the current PDDL domain following standard PDDL grounding conventions (Francés et al., 2018). The resulting grounded facts are validated against the predicate signatures and object types defined in the current PDDL domain. Successful and failed interaction outcomes are used to construct positive and negative traces for contrastive validation.

Given the accumulated traces, GPT-5.6-Sol proposes revisions to the current predicates and typed PDDL action schemas using validation feedback from preceding refinement iterations. When a candidate introduces or revises predicates, the recorded pre- and post-action states are regrounded under the candidate PDDL domain and the corresponding transition evidence is recomputed. Each candidate domain is evaluated by reconstructing the recorded transitions and validating them against the corresponding PDDL preconditions and effects, with Fast Downward supporting goal reachability checks. Reconstruction discrepancies are returned as feedback for subsequent refinement. Following the contrastive validation procedure in Section 3.2, we select the highest-scoring knowledge state across the initial and refined candidates as the consolidated symbolic knowledge.

The hyperparameter settings for *Knowledge Consolidation* are summarized in Table 21.

Table 21: Hyperparameter settings for *Knowledge Consolidation* on CL-ALFWorld.

Hyperparameters	**Value**
Reasoning effort (GPT-5.6-Sol)	Medium
Max. generation tokens (GPT-5.6-Sol)	4000
Max. refinement iterations N	5
Negative trace weight λ	1.0

Knowledge Compilation. For CL-ALFWorld, *Knowledge Compilation* is applied to verified PDDL-grounded decision patterns while keeping Qwen3-4B-Instruct-2507 frozen. Compilation instances associate each decision context, including the current symbolic state and task goal, with

the grounded action validated against the current consolidated symbolic knowledge. Compatible instances supported by the same knowledge are grouped according to the pattern-selection criteria in Section 3.3.

For each selected group, the semantic adapter is trained on selected hidden representations of the frozen Qwen3-4B-Instruct-2507 model to characterize reusable decision contexts through representation reconstruction, while the paired procedural adapter is trained to adapt the model's output decisions toward the corresponding grounded action.

During inference, the semantic adapters are ranked by their mean standardized reconstruction errors across the selected layers, and the procedural adapter paired with the lowest-error semantic adapter is activated with $k = 1$. Generated actions undergo the validity checks described in the reasoning flow before execution, with failed proposals falling back to Fast Downward.

Neural memory grows incrementally as new adapter pairs are added for newly consolidated reasoning patterns. If a subsequent *Knowledge Consolidation* process revises the symbolic knowledge from which an existing adapter pair was constructed, the affected pair is identified through its association with that symbolic knowledge and grounded bindings, and is removed or retrained. Adapter pairs associated with unaffected symbolic knowledge are retained.

The hyperparameter settings for *Knowledge Compilation* are summarized in Table 22.

Table 22: Hyperparameter settings for *Knowledge Compilation* on CL-ALFWorld.

Hyperparameters	**Value**
Selected transformer layers	12–24 (zero-based)
Hidden representation dimension	2,560
Semantic adapter architecture (AE)	2560–256–2560
Procedural adapter architecture	2560–64–2560
Sparse activation k	1
Semantic adapter optimizer	AdamW
Semantic adapter learning rate	2×10^{-3}
Semantic adapter training steps	2,000
Semantic adapter batch size	Full batch
Procedural adapter optimizer	AdamW
Procedural adapter learning rate	2×10^{-4}
Procedural adapter training epochs	8–20
Procedural adapter batch size	1

A.4 METRICS

We divide each task stream into $K = 10$ sequential blocks and evaluate the block-level task performance and symbolic tool reliance measurements required to compute the lifelong learning metrics. These block-wise measurements are aggregated according to the definitions below to obtain the overall metrics reported in the main text. Let $R_{i,j}$ denote the task success rate on block j after completing block i, with $R_{0,j}$ denoting initial performance.

For task performance, we report final success rate (SR), backward transfer (BWT), and forward transfer (FWT) (Lopez-Paz & Ranzato, 2017):

$$\mathrm{SR} = \frac{1}{K}\sum_{j=1}^{K} R_{K,j}, \quad \mathrm{BWT} = \frac{1}{K-1}\sum_{j=1}^{K-1}\left(R_{K,j} - R_{j,j}\right), \quad \mathrm{FWT} = \frac{1}{K-1}\sum_{j=2}^{K}\left(R_{j-1,j} - R_{0,j}\right). \tag{8}$$

SR measures the average final performance across all blocks after completing the full task stream. BWT compares the final performance on each previously learned block with its performance immediately after that block was learned, measuring how subsequent learning affects earlier tasks. A positive BWT indicates improved performance on previously learned blocks, whereas a negative BWT indicates performance degradation on those blocks. FWT compares performance on each future block immediately before it is learned with its initial performance, measuring whether knowledge acquired from preceding blocks transfers to tasks not yet learned. A positive FWT indicates beneficial transfer from preceding blocks to future blocks before direct learning, whereas a negative FWT indicates lower performance on those blocks than at initialization.

We count a symbolic component as symbolic tool use only when it is invoked within the agent's reasoning flow and contributes to reasoning or action selection. Components outside the reasoning flow, such as environment- or benchmark-provided mechanisms used solely for task execution or validity checking, are not counted. If their outputs trigger symbolic reasoning or replanning within the reasoning flow, the resulting symbolic tool invocation is counted.

For symbolic tool reliance, let $D_{i,j}$ denote the fraction of evaluated reasoning units on block j that rely on symbolic tools after completing block i (Xu et al., 2025). We report final tool reliance (TR), backward reliance reduction (BRR), and forward reliance reduction (FRR):

$$\mathrm{TR} = \frac{1}{K}\sum_{j=1}^{K} D_{K,j}, \quad \mathrm{BRR} = \frac{1}{K-1}\sum_{j=1}^{K-1}\left(D_{j,j} - D_{K,j}\right), \quad \mathrm{FRR} = \frac{1}{K-1}\sum_{j=2}^{K}\left(D_{0,j} - D_{j-1,j}\right). \tag{9}$$

TR measures the average final symbolic tool reliance across all blocks after completing the full task stream. BRR compares the tool reliance on each previously learned block immediately after that block was learned with its final reliance, measuring how subsequent learning changes tool reliance on earlier tasks. A positive BRR indicates reduced symbolic tool reliance on previously learned blocks, whereas a negative BRR indicates increased reliance on those blocks. FRR compares the initial tool reliance on each future block with its reliance immediately before that block is learned, measuring whether knowledge acquired from preceding blocks reduces tool reliance on tasks not yet learned. A positive FRR indicates reduced symbolic tool reliance on future blocks before direct learning, whereas a negative FRR indicates increased reliance on future blocks. We interpret these reliance metrics jointly with task success (Qian et al., 2025), since reduced symbolic tool use alone does not establish effective compilation. We report SR and TR as percentages (%) and BWT, FWT, BRR, and FRR in percentage points (pp).

For task performance, we evaluate $R_{0,j}$ and $R_{j-1,j}$ for $j = 2, \ldots, K$, $R_{j,j}$ for $j = 1, \ldots, K-1$, and $R_{K,j}$ for all $j = 1, \ldots, K$, corresponding to the Initial, Before Learning, After Learning, and Final entries, respectively. For methods to which symbolic tool reliance is applicable, we evaluate the corresponding entries $D_{0,j}$, $D_{j-1,j}$, $D_{j,j}$, and $D_{K,j}$ over the same index ranges. Entries not required by the aggregate metrics are marked "-". Appendix A.5.1 reports these block-level measurements.

A.5 Additional Evaluation Results

A.5.1 Block-wise Performance Comparison

To complement the aggregate results reported in Section 4.2, we expand the main performance comparison in Table 1 into block-wise measurements for each benchmark. The tables below report the Initial, Before Learning, After Learning, and Final entries defined in Appendix A.4, which are used to compute SR, BWT, FWT, TR, BRR, and FRR. These results provide a more detailed view of how task performance and symbolic tool reliance evolve throughout the sequential task stream.

Table 23: Block-wise task performance on LIBERO-PRO underlying the aggregate SR, BWT, and FWT results. "-" denotes entries not required for metric computation. Results are reported as mean success rates (%) with sample standard deviations over three seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
$\pi_{0.5}$ (LIBERO-FT)	Initial	-	39.58±3.61	43.75±6.25	50.00±0.00	58.33±14.43
	Before Learning	-	39.58±3.61	43.75±6.25	50.00±0.00	58.33±14.43
	After Learning	43.75±6.25	39.58±3.61	43.75±6.25	50.00±0.00	58.33±14.43
	Final	43.75±6.25	39.58±3.61	43.75±6.25	50.00±0.00	58.33±14.43
OpenRAL	Initial	-	45.83±13.01	58.33±3.61	62.50±6.25	56.25±10.83
	Before Learning	-	45.83±13.01	58.33±3.61	62.50±6.25	56.25±10.83
	After Learning	52.08±21.95	45.83±13.01	58.33±3.61	62.50±6.25	56.25±10.83
	Final	52.08±21.95	45.83±13.01	58.33±3.61	62.50±6.25	56.25±10.83
HarnessVLA	Initial	-	47.92±7.22	68.75±10.83	72.92±3.61	45.83±13.01
	Before Learning	-	56.25±6.25	54.17±3.61	77.08±3.61	47.92±9.55
	After Learning	58.33±7.22	52.08±7.22	72.92±13.01	75.00±6.25	56.25±10.83
	Final	60.42±9.55	58.33±7.22	75.00±6.25	83.33±3.61	64.58±3.61
VisualPredicator	Initial	-	52.08±3.61	54.17±7.22	56.25±0.00	60.42±7.22
	Before Learning	-	58.33±3.61	58.33±14.43	62.50±10.83	62.50±10.83
	After Learning	62.50±6.25	54.17±3.61	58.33±14.43	60.42±7.22	64.58±14.43
	Final	60.42±3.61	50.00±6.25	54.17±7.22	56.25±0.00	56.25±0.00
NeSyC	Initial	-	33.33±3.61	29.17±3.61	43.75±6.25	41.67±3.61
	Before Learning	-	43.75±10.83	29.17±7.22	43.75±6.25	47.92±7.22
	After Learning	41.67±3.61	35.42±3.61	29.17±9.55	41.67±3.61	47.92±7.22
	Final	58.33±3.61	47.92±3.61	39.58±7.22	47.92±9.55	54.17±9.55
NeSyChain (Ours)	Initial	-	43.06±5.24	45.14±6.70	50.00±0.00	58.33±14.43
	Before Learning	-	43.06±5.24	45.14±6.70	50.00±6.25	58.33±9.55
	After Learning	46.53±3.18	43.06±5.24	45.83±3.61	50.00±6.25	59.38±3.13
	Final	75.00±12.50	72.92±3.61	75.00±0.00	70.83±3.61	79.17±7.22

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
$\pi_{0.5}$ (LIBERO-FT)	Initial	45.83±3.61	43.75±10.83	50.00±6.25	47.92±3.61	52.08±3.61
	Before Learning	45.83±3.61	43.75±10.83	50.00±6.25	47.92±3.61	52.08±3.61
	After Learning	45.83±3.61	43.75±10.83	50.00±6.25	47.92±3.61	-
	Final	45.83±3.61	43.75±10.83	50.00±6.25	47.92±3.61	52.08±3.61
OpenRAL	Initial	66.67±3.61	58.33±7.22	60.42±3.61	70.83±9.55	60.42±9.55
	Before Learning	66.67±3.61	58.33±7.22	60.42±3.61	70.83±9.55	60.42±9.55
	After Learning	66.67±3.61	58.33±7.22	60.42±3.61	70.83±9.55	-
	Final	66.67±3.61	58.33±7.22	60.42±3.61	70.83±9.55	60.42±9.55
HarnessVLA	Initial	50.00±0.00	41.67±3.61	56.25±10.83	56.25±0.00	62.50±6.25
	Before Learning	64.58±13.01	52.08±3.61	62.50±10.83	62.50±6.25	87.50±0.00
	After Learning	64.58±9.55	43.75±16.54	64.58±20.09	68.75±0.00	-
	Final	66.67±3.61	45.83±9.55	64.58±3.61	62.50±6.25	79.17±7.22
VisualPredicator	Initial	52.08±3.61	66.67±3.61	60.42±3.61	58.33±3.61	62.50±10.83
	Before Learning	58.33±14.43	64.58±7.22	58.33±3.61	60.42±7.22	66.67±3.61
	After Learning	56.25±10.83	68.75±0.00	58.33±3.61	56.25±6.25	-
	Final	52.08±3.61	66.67±3.61	58.33±3.61	56.25±6.25	64.58±7.22
NeSyC	Initial	41.67±7.22	41.67±3.61	45.83±9.55	43.75±0.00	50.00±6.25
	Before Learning	43.75±6.25	39.58±3.61	47.92±7.22	47.92±3.61	52.08±9.55
	After Learning	43.75±6.25	43.75±0.00	45.83±7.22	45.83±3.61	-
	Final	45.83±3.61	54.17±3.61	54.17±9.55	52.08±3.61	58.33±7.22
NeSyChain (Ours)	Initial	45.83±3.61	46.53±7.89	51.39±2.41	50.00±6.25	54.17±3.61
	Before Learning	55.21±9.02	52.08±14.43	62.50±12.50	77.08±3.61	72.92±3.61
	After Learning	55.21±9.02	52.08±14.43	75.00±6.25	77.08±3.61	-
	Final	75.00±6.25	77.08±3.61	75.00±6.25	77.08±3.61	72.92±3.61

Table 24: Block-wise symbolic tool reliance on LIBERO-PRO underlying the aggregate TR, BRR, and FRR results. The VLA-only $\pi_{0.5}$ and HarnessVLA baselines are omitted because symbolic tool reliance is not applicable. "-" denotes entries not required for metric computation. Results are reported as mean tool reliance (%) with sample standard deviations over three seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
OpenRAL	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
VisualPredicator	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
NeSyC	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
NeSyChain (Ours)	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	33.94±5.92
	Final	48.97±4.00	53.72±8.79	56.56±10.82	57.19±12.47	60.53±13.95

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
OpenRAL	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
VisualPredicator	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
NeSyC	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
NeSyChain (Ours)	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	42.89±4.38	39.10±2.95	37.68±1.61	50.60±7.38	55.00±10.88
	After Learning	42.89±4.38	39.10±2.95	52.56±11.51	50.60±7.38	-
	Final	54.17±8.38	53.96±10.49	52.56±11.51	50.60±7.38	55.00±10.88

LIBERO-PRO. Table 23 provides a block-wise view of task performance. For NeSyChain, final performance remains consistently high across all blocks, ranging from 70.83% to 79.17%. On earlier blocks, performance further improves after subsequent learning. For example, Block 1 increases from 46.53% after learning to 75.00% at the end of the stream, while Block 2 increases from 43.06% to 72.92%. Forward transfer becomes more evident in later blocks, where performance before direct learning exceeds the initial performance, including 77.08% versus 50.00% on Block 9 and 72.92% versus 54.17% on Block 10. These block-wise trends are consistent with the positive BWT and FWT reported in Table 1.

Table 24 shows the corresponding evolution of symbolic tool reliance. OpenRAL, VisualPredicator, and NeSyC retain 100% reliance throughout the evaluated entries, whereas NeSyChain progressively reduces reliance as learning proceeds. Final reliance for NeSyChain ranges from 48.97% to 60.53% across the ten blocks. Reliance on future blocks is also reduced before direct learning, decreasing from the initial 100% to 42.89%, 39.10%, 37.68%, 50.60%, and 55.00% for Blocks 6–10, respectively. Earlier blocks additionally show substantial reductions after subsequent learning, with Block 1 decreasing from 100% after learning to 48.97% at the end of the stream.

Together, these block-wise results show that NeSyChain improves task performance across the sequential task stream while progressively reducing symbolic tool reliance. The improvements in both dimensions support effective knowledge transfer across tasks together with reduced dependence on symbolic tools.

Table 25: Block-wise task performance on CaP-Bench underlying the aggregate SR, BWT, and FWT results. "-" denotes entries not required for metric computation. Results are reported as mean success rates (%) with sample standard deviations over three seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
CaP-Agent0	Initial	-	41.03±16.01	82.05±4.44	69.44±12.73	94.44±4.81
	Before Learning	-	46.15±0.00	79.49±4.44	72.22±9.62	91.67±0.00
	After Learning	41.03±11.75	56.41±4.44	82.05±16.01	72.22±4.81	88.89±9.62
	Final	41.03±16.01	61.54±7.69	82.05±8.88	61.11±19.25	94.44±4.81
LangChain	Initial	-	56.41±16.01	84.62±0.00	52.78±4.81	83.33±14.43
	Before Learning	-	58.97±11.75	84.62±0.00	50.00±8.33	86.11±12.73
	After Learning	64.10±8.88	51.28±19.36	87.18±4.44	52.78±12.73	83.33±16.67
	Final	64.10±8.88	56.41±11.75	84.62±0.00	52.78±4.81	88.89±9.62
NeSyC	Initial	-	89.74±3.63	84.62±12.56	47.22±10.39	86.11±3.93
	Before Learning	-	64.10±3.63	64.10±15.81	50.00±0.00	61.11±7.86
	After Learning	66.67±9.59	64.10±3.63	66.67±13.07	52.78±3.93	61.11±3.93
	Final	71.79±3.63	66.67±3.63	69.23±12.56	52.78±3.93	61.11±10.39
WALL-E 2.0	Initial	-	46.15±7.69	89.74±4.44	66.67±22.05	83.33±8.33
	Before Learning	-	41.03±11.75	87.18±4.44	75.00±8.33	91.67±8.33
	After Learning	41.03±8.88	61.54±0.00	89.74±8.88	61.11±17.35	94.44±4.81
	Final	69.23±15.38	58.97±4.44	87.18±11.75	63.89±19.25	86.11±17.35
NeSyChain (Ours)	Initial	-	76.91±15.40	89.74±4.44	88.89±4.81	94.44±4.81
	Before Learning	-	84.62±7.69	92.31±0.00	88.89±9.62	100.00±0.00
	After Learning	76.92±0.00	69.23±7.69	92.31±0.00	77.78±4.81	97.22±4.81
	Final	89.74±4.44	76.92±7.69	92.31±0.00	86.11±4.81	97.22±4.81

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
CaP-Agent0	Initial	69.44±4.81	66.67±14.43	52.78±17.35	55.56±4.81	63.89±19.25
	Before Learning	66.67±16.67	38.89±19.25	61.11±9.62	44.44±12.73	55.56±9.62
	After Learning	55.56±9.62	61.11±9.62	50.00±0.00	52.78±9.62	-
	Final	44.44±12.73	47.22±9.62	55.56±9.62	47.22±4.81	69.44±9.62
LangChain	Initial	50.00±14.43	41.67±0.00	55.56±4.81	58.33±16.67	52.78±12.73
	Before Learning	50.00±8.33	41.67±8.33	55.56±4.81	61.11±12.73	50.00±14.43
	After Learning	50.00±8.33	38.89±4.81	55.56±4.81	61.11±20.97	-
	Final	44.44±9.62	41.67±0.00	50.00±0.00	55.56±20.97	52.78±12.73
NeSyC	Initial	88.89±7.86	63.89±7.86	80.56±3.93	77.78±7.86	83.33±6.80
	Before Learning	72.22±7.86	47.22±3.93	52.78±3.93	41.67±6.80	33.33±0.00
	After Learning	72.22±7.86	47.22±3.93	50.00±6.80	44.44±7.86	-
	Final	72.22±7.86	47.22±3.93	52.78±3.93	47.22±3.93	36.11±3.93
WALL-E 2.0	Initial	55.56±17.35	38.89±4.81	61.11±12.73	58.33±8.33	61.11±9.62
	Before Learning	69.44±17.35	52.78±4.81	63.89±26.79	63.89±12.73	52.78±17.35
	After Learning	77.78±4.81	44.44±12.73	58.33±16.67	63.89±4.81	-
	Final	69.44±12.73	58.33±14.43	69.44±4.81	41.67±22.05	55.56±9.62
NeSyChain (Ours)	Initial	88.89±9.62	69.44±4.81	75.00±8.33	80.56±4.81	86.11±9.62
	Before Learning	94.44±9.62	80.56±9.62	83.33±8.33	91.67±0.00	100.00±0.00
	After Learning	80.56±4.81	77.78±4.81	83.33±0.00	88.89±4.81	-
	Final	80.56±4.81	91.67±8.33	86.11±4.81	88.89±4.81	100.00±0.00

CaP-Bench. Table 25 provides a block-wise view of task performance. For NeSyChain, final performance remains high across all blocks, ranging from 76.92% to 100.00%. Performance on previously learned blocks is maintained or further improved after subsequent learning. For example, Block 1 increases from 76.92% after learning to 89.74% at the end of the stream, while Block 7 increases from 77.78% to 91.67%, consistent with the positive BWT reported in Table 1. Forward transfer is also observed across later blocks, with performance before direct learning exceeding the initial performance on most future blocks. For example, Block 7 increases from 69.44% initially to 80.56% before learning, while Block 10 increases from 86.11% to 100.00%, consistent with the positive FWT reported in Table 1.

Table 26 shows substantial reductions in symbolic tool reliance. While the baselines remain close to full reliance across the stream, NeSyChain reduces final reliance to between 4.60% and 27.80% across all blocks. Reliance on several previously learned blocks further decreases after subsequent learning. For example, Block 1 decreases from 39.60% after learning to 20.30% at the end of the stream, consistent with the positive BRR reported in Table 1. Reduced reliance also transfers to future blocks before direct learning. For Blocks 6–10, reliance decreases from the initial 100%

Table 26: Block-wise symbolic tool reliance on CaP-Bench underlying the aggregate TR, BRR, and FRR results. "-" denotes entries not required for metric computation. Results are reported as mean tool reliance (%) with sample standard deviations over three seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
CaP-Agent0	Initial	-	99.71±0.51	100.00±0.00	99.49±0.89	100.00±0.00
	Before Learning	-	99.13±0.06	96.28±1.39	100.00±0.00	98.72±2.22
	After Learning	100.00±0.00	99.39±0.53	100.00±0.00	100.00±0.00	99.37±1.09
	Final	99.77±0.40	100.00±0.00	99.36±1.11	100.00±0.00	99.10±1.56
LangChain	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
NeSyC	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
WALL-E 2.0	Initial	-	98.85±1.25	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	99.14±0.78	100.00±0.00	98.10±1.85	100.00±0.00
	After Learning	95.95±7.01	100.00±0.00	100.00±0.00	98.61±2.41	100.00±0.00
	Final	99.03±0.24	99.30±1.22	100.00±0.00	99.49±0.87	100.00±0.00
NESYCHAIN (Ours)	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	73.85±2.44	15.24±7.87	38.35±6.97	14.26±3.13
	After Learning	39.60±12.21	35.28±4.11	14.34±3.84	28.29±4.54	3.76±0.22
	Final	20.30±7.48	27.80±6.86	9.22±2.84	22.85±6.18	4.60±7.96

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
CaP-Agent0	Initial	98.27±1.52	100.00±0.00	99.49±0.87	100.00±0.00	100.00±0.00
	Before Learning	99.17±1.44	97.88±3.67	99.48±0.90	100.00±0.00	100.00±0.00
	After Learning	91.29±13.03	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	98.88±1.95	97.58±2.26	100.00±0.00	100.00±0.00	100.00±0.00
LangChain	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
NeSyC	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
WALL-E 2.0	Initial	99.22±1.34	88.25±9.15	87.35±14.87	97.78±2.60	99.59±0.70
	Before Learning	97.61±3.01	96.45±6.14	99.38±1.07	99.54±0.80	98.13±1.06
	After Learning	91.61±10.11	99.32±0.59	93.90±9.46	99.06±0.83	-
	Final	96.86±1.93	99.63±0.63	99.62±0.66	97.62±1.63	99.03±0.89
NESYCHAIN (Ours)	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	43.11±4.61	42.24±7.78	38.77±7.30	30.55±1.80	22.78±1.96
	After Learning	20.39±9.92	38.36±14.60	21.71±5.38	15.84±9.02	-
	Final	23.11±8.54	22.11±8.43	23.88±3.80	10.61±2.62	6.96±1.50

to 43.11%, 42.24%, 38.77%, 30.55%, and 22.78%, respectively, before those blocks are directly learned. The reduction becomes increasingly pronounced toward later blocks, consistent with the positive FRR reported in Table 1.

Together, these results show that NESYCHAIN maintains high task performance while substantially reducing symbolic tool reliance across the sequential task stream. The improvements in both dimensions support effective knowledge transfer across tasks together with reduced dependence on symbolic tools.

Table 27: Block-wise task performance on CL-ALFWorld underlying the aggregate SR, BWT, and FWT results. "-" denotes entries not required for metric computation. Results are reported as mean success rates (%) with sample standard deviations over three seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
LangChain	Initial	-	33.33±7.64	1.67±2.89	46.67±14.43	16.67±2.89
	Before Learning	-	31.67±7.64	0.00±0.00	48.33±7.64	18.33±2.89
	After Learning	38.33±20.21	31.67±10.41	5.00±5.00	48.33±7.64	15.00±0.00
	Final	40.00±17.32	30.00±5.00	3.33±2.89	48.33±7.64	16.67±2.89
NeSyC	Initial	-	33.33±7.64	6.67±2.89	50.00±5.00	18.33±2.89
	Before Learning	-	73.33±2.89	15.00±0.00	71.67±2.89	50.00±0.00
	After Learning	73.33±2.89	75.00±5.00	15.00±0.00	70.00±5.00	48.33±2.89
	Final	63.33±16.07	75.00±5.00	15.00±0.00	70.00±8.66	48.33±2.89
WALL-E 2.0	Initial	-	33.33±7.64	6.67±2.89	48.33±7.64	18.33±2.89
	Before Learning	-	78.33±2.89	13.33±2.89	71.67±5.77	50.00±10.00
	After Learning	78.33±7.64	73.33±2.89	15.00±0.00	73.33±5.77	56.67±10.41
	Final	78.33±2.89	75.00±5.00	16.67±2.89	73.33±7.64	56.67±5.77
NESYCHAIN (Ours)	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
	Before Learning	-	85.00±0.00	15.00±0.00	80.00±0.00	70.00±0.00
	After Learning	85.00±0.00	85.00±0.00	75.00±0.00	80.00±0.00	80.00±0.00
	Final	85.00±0.00	90.00±0.00	83.33±2.89	95.00±0.00	95.00±0.00

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
LangChain	Initial	26.67±2.89	41.67±17.56	25.00±8.66	20.00±5.00	25.00±5.00
	Before Learning	30.00±5.00	35.00±20.00	26.67±5.77	16.67±2.89	25.00±5.00
	After Learning	23.33±2.89	38.33±15.28	25.00±5.00	23.33±5.77	-
	Final	23.33±7.64	35.00±15.00	26.67±5.77	21.67±7.64	28.33±5.77
NeSyC	Initial	28.33±2.89	45.00±20.00	28.33±7.64	26.67±7.64	28.33±2.89
	Before Learning	50.00±0.00	61.67±10.41	60.00±8.66	46.67±2.89	58.33±12.58
	After Learning	56.67±12.58	65.00±10.00	60.00±10.00	45.00±0.00	-
	Final	50.00±10.00	61.67±17.56	58.33±7.64	43.33±2.89	58.33±12.58
WALL-E 2.0	Initial	28.33±2.89	43.33±16.07	28.33±7.64	23.33±5.77	28.33±7.64
	Before Learning	51.67±18.93	63.33±2.89	53.33±7.64	58.33±10.41	58.33±15.28
	After Learning	60.00±15.00	58.33±10.41	51.67±7.64	58.33±10.41	-
	Final	56.67±2.89	58.33±7.64	50.00±5.00	58.33±12.58	56.67±14.43
NESYCHAIN (Ours)	Initial	40.00±0.00	65.00±0.00	35.00±0.00	25.00±0.00	35.00±0.00
	Before Learning	85.00±0.00	80.00±0.00	100.00±0.00	85.00±0.00	95.00±0.00
	After Learning	85.00±0.00	95.00±0.00	100.00±0.00	85.00±0.00	-
	Final	90.00±0.00	95.00±0.00	100.00±0.00	85.00±0.00	95.00±0.00

CL-ALFWorld. Table 27 provides a block-wise view of task performance. For NESYCHAIN, final performance remains high across all blocks, ranging from 83.33% to 100.00%. Performance on several previously learned blocks further improves after subsequent learning. For example, Block 4 increases from 80.00% after learning to 95.00% at the end of the stream, while Block 5 increases from 80.00% to 95.00%, consistent with the positive BWT reported in Table 1. A distinctive pattern on CL-ALFWorld is the strong forward transfer across future blocks. Performance before direct learning exceeds the initial performance on every future block. For example, Block 8 increases from 35.00% initially to 100.00% before learning, while Blocks 9 and 10 increase from 25.00% to 85.00% and from 35.00% to 95.00%, respectively. These gains are consistent with the positive FWT reported in Table 1.

Table 28 shows the corresponding changes in symbolic tool reliance. NeSyC retains 100% reliance throughout the evaluated entries, while LangChain and WALL-E 2.0 remain close to full reliance across the stream. In contrast, final reliance for NESYCHAIN ranges from 57.07% to 76.33% across the ten blocks. Reliance on most previously learned blocks is further reduced after subsequent learning. For example, Block 1 decreases from 81.03% after learning to 70.02% at the end of the stream, while Block 2 decreases from 83.14% to 65.56%, consistent with the positive BRR reported in Table 1. Reduced reliance also transfers to future blocks before direct learning. For later blocks, reliance decreases from the initial 100% to 72.04%, 73.37%, 74.43%, and 70.83% on Blocks 7–10, respectively, consistent with the positive FRR reported in Table 1.

Together, these results show that NESYCHAIN achieves strong transfer in task performance while reducing symbolic tool reliance across the sequential task stream. The improvements in both dimen-

Table 28: Block-wise symbolic tool reliance on CL-ALFWorld underlying the aggregate TR, BRR, and FRR results. "-" denotes entries not required for metric computation. Results are reported as mean tool reliance (%) with sample standard deviations over three seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
LangChain	Initial	-	93.97±1.46	96.63±1.78	96.65±0.85	96.51±0.87
	Before Learning	-	95.70±1.23	97.32±0.62	98.89±1.30	97.84±1.91
	After Learning	95.21±0.78	95.34±0.53	96.20±0.06	97.73±1.32	96.35±3.53
	Final	98.30±1.52	96.52±1.04	96.32±0.82	98.22±1.71	98.30±1.82
NeSyC	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
WALL-E 2.0	Initial	-	93.69±1.46	95.24±1.51	96.80±1.17	96.67±1.14
	Before Learning	-	92.70±1.30	94.57±0.72	96.53±1.81	91.42±2.47
	After Learning	94.58±2.66	94.34±0.71	94.70±1.26	95.96±1.36	92.87±1.63
	Final	93.12±0.92	94.78±2.02	93.64±1.19	96.27±1.36	91.29±1.73
NESYCHAIN (Ours)	Initial	-	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	-	100.00±0.00	94.00±0.00	80.00±0.74	81.52±1.48
	After Learning	81.03±0.00	83.14±0.66	80.00±0.00	66.67±0.00	65.52±0.21
	Final	70.02±0.42	65.56±0.00	76.33±1.25	65.28±0.00	69.53±0.53

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
LangChain	Initial	93.55±2.79	93.51±2.46	98.09±0.45	98.51±0.42	100.00±0.00
	Before Learning	93.90±1.83	94.77±2.18	96.84±0.58	98.98±0.16	97.70±1.79
	After Learning	93.92±4.32	92.32±2.79	97.68±0.96	96.93±0.72	-
	Final	93.18±2.39	94.26±1.32	97.67±1.27	97.07±1.96	99.14±0.89
NeSyC	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	After Learning	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	-
	Final	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
WALL-E 2.0	Initial	93.90±2.29	93.97±3.30	98.44±1.62	97.11±0.80	97.75±1.60
	Before Learning	91.97±2.80	91.34±1.05	95.73±2.60	95.98±1.17	95.37±0.76
	After Learning	92.47±1.43	91.00±2.35	95.52±0.92	94.37±1.07	-
	Final	89.93±2.92	89.94±3.29	94.39±2.65	95.41±2.70	96.57±1.84
NESYCHAIN (Ours)	Initial	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00	100.00±0.00
	Before Learning	81.04±0.91	72.04±1.68	73.37±1.46	74.43±2.77	70.83±0.29
	After Learning	77.58±2.65	58.59±0.44	65.66±1.16	66.37±1.01	-
	Final	72.43±2.40	57.07±0.44	63.38±0.44	66.37±0.51	66.34±0.57

sions show that NESYCHAIN's two-stage knowledge transformation enables acquired knowledge to support future tasks while reducing dependence on symbolic tools.

A.5.2 Detailed Analysis of Symbolic Knowledge and Neural Memory

To complement the analysis in Figure 3 of Section 4.3, we provide a more detailed round-wise view of symbolic knowledge and neural memory throughout lifelong learning. The figures below separately report task performance and inference time on seen and unseen tasks under Default, Consolidated-only, and Compiled-only inference across the *Knowledge Consolidation–Compilation* rounds. These results further illustrate how the contributions of consolidated symbolic knowledge and compiled neural memory evolve as learning progresses.

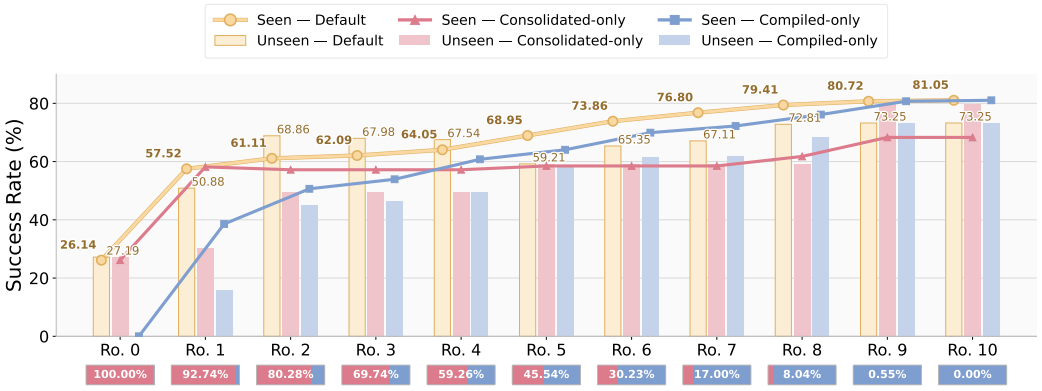


Figure 8: Detailed round-wise success rates on seen and unseen tasks under Default, Consolidated-only, and Compiled-only inference modes. Each round (Ro.) is one *Knowledge Consolidation–Compilation* cycle. Under the Default inference mode, success rates are annotated for both seen and unseen tasks, with seen task values in bold and unseen task values in regular font. The relative use of consolidated symbolic knowledge and compiled neural memory under Default inference is shown below the x-axis.

Task performance. Figure 8 provides a detailed view of task performance across *Knowledge Consolidation–Compilation* rounds. Under Default inference, seen task success increases from 26.14% at Round 0 to 81.05% at Round 10, while unseen task success increases from 27.19% to 73.25%. The controlled inference modes reveal different learning dynamics for consolidated symbolic knowledge and compiled neural memory. Consolidated-only inference improves more rapidly in the early rounds and initially outperforms Compiled-only inference, whereas Compiled-only inference starts lower and progressively catches up as more validated patterns are compiled. By Round 10, Compiled-only inference matches Default inference at 81.05% on seen tasks and 73.25% on unseen tasks. On seen tasks, Compiled-only inference surpasses Consolidated-only inference from the middle rounds onward, reaching 81.05% at Round 10 compared with 68.30% for Consolidated-only inference. On unseen tasks, the relative advantage varies across rounds, but Consolidated-only inference ultimately reaches 79.82%, exceeding the 73.25% of Compiled-only inference at the end of learning. These results suggest that consolidated symbolic knowledge and compiled neural memory generalize differently. Compiled neural memory is particularly effective at reusing recurring patterns within the learned tasks, whereas the lifted predicates and action schemas in consolidated symbolic knowledge may support broader transfer to held-out tasks.

Inference time. Figure 9 provides a detailed view of inference time across *Knowledge Consolidation–Compilation* rounds. Under Default inference, inference time on seen tasks decreases from 7.64s at Round 0 to 0.21s at Round 10, while that on unseen tasks decreases from 4.77s to 0.31s. The controlled inference modes reveal different efficiency profiles for consolidated symbolic knowledge and compiled neural memory. Consolidated-only inference remains relatively expensive throughout the later rounds, whereas Compiled-only inference maintains substantially lower inference time from Round 1 onward. At Round 10, Consolidated-only inference requires 4.07s on seen tasks and 3.55s on unseen tasks, while Compiled-only inference requires only 0.21s and 0.31s, respectively. As Default inference increasingly relies on compiled neural memory, its inference time progressively approaches the Compiled-only setting. From Round 7 onward, Default inference falls below 1s on both seen and unseen tasks, and at Round 10, when it no longer relies

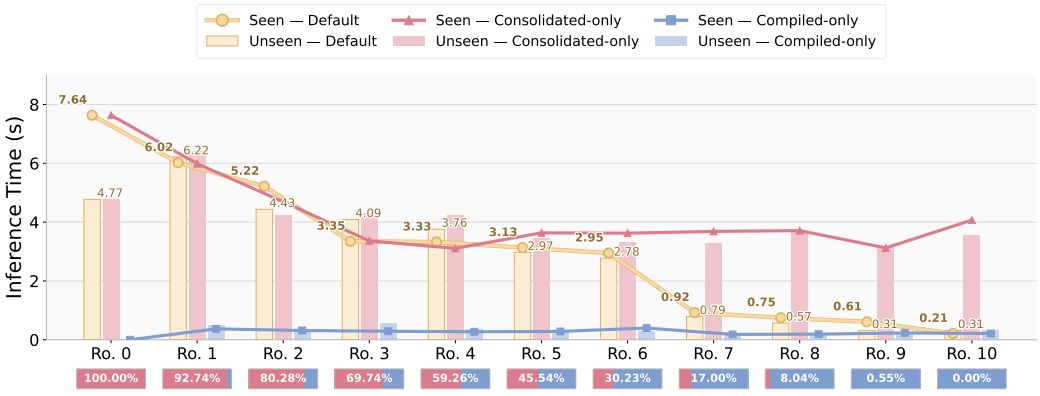


Figure 9: Detailed round-wise inference times on seen and unseen tasks under Default, Consolidated-only, and Compiled-only inference modes. Each round (Ro.) is one *Knowledge Consolidation–Compilation* cycle. Under the Default inference mode, inference times are annotated for both seen and unseen tasks, with seen task values in bold and unseen task values in regular font. The relative use of consolidated symbolic knowledge and compiled neural memory under Default inference is shown below the x-axis.

on consolidated symbolic knowledge, its inference time matches Compiled-only inference on both evaluation sets. These results highlight the differing inference costs of consolidated symbolic knowledge and compiled neural memory. Consolidated symbolic knowledge supports explicit reasoning over acquired knowledge but incurs the runtime cost of symbolic tool use, whereas compiled neural memory enables substantially faster reuse by making validated reasoning patterns directly available to the frozen foundation model.

Overall trade-off. Taken together, these analyses reveal complementary trade-offs between consolidated symbolic knowledge and compiled neural memory. Consolidated symbolic knowledge is immediately available for explicit reasoning after validation and may better support transfer to held-out tasks, but incurs higher inference cost through continued symbolic tool use. Compiled neural memory substantially reduces inference cost and becomes particularly effective for recurring patterns in learned tasks, but its benefits emerge progressively as validated patterns are accumulated and compiled. These results suggest that lifelong neurosymbolic agents should balance explicit symbolic reasoning and efficient neural reuse rather than relying exclusively on either. Maintaining symbolic knowledge while progressively compiling recurring validated patterns into neural memory provides a way to adapt this balance throughout lifelong learning.

A.5.3 Detailed Analysis of Modular Implementation and Lifelong Learning

To complement Table 2 and its analysis in Section 4.3, we provide implementation details and block-wise results for the three neurosymbolic configurations. We first describe how NL-PDDL, NeSyRo, and CaSP are instantiated within the common modular structure of NeSyChain while preserving their original reasoning flows. We then examine how the common lifelong learning procedure affects each configuration throughout the sequential task stream.

NL-PDDL (Liu et al., 2026) uses an LM for commonsense entailment over natural-language predicates and a symbolic planner for lifted first-order regression planning. Within NeSyChain, NL-PDDL is instantiated with Qwen3-4B-Instruct-2507 through the Model Interface, a lifted first-order regression planner through the Tool Interface, and natural-language predicates and action knowledge accessed through the Knowledge Interface. During reasoning, `read()` retrieves the maintained knowledge, `execute()` regresses the current subgoal over candidate actions, and `infer()` evaluates entailment between subgoals and action effects. Accordingly, `read()` accesses the maintained knowledge, `execute()` calls the symbolic planner, and `infer()` delegates commonsense entailment to the LM. Starting from the task goal, these operations recursively derive pre-action conditions until a conditional plan is constructed.

NeSyRo (Ahn et al., 2025) uses an LM to generate task specifications and policy code, an SMT solver (de Moura & Bjørner, 2008) to verify logical consistency, and a symbolic planner (Helmert, 2006) to assess skill feasibility. Within NeSyChain, NeSyRo is instantiated with Qwen3-4B-Instruct-2507 through the Model Interface, Z3 and Fast Downward through the Tool Interface, and PDDL knowledge accessed through the Knowledge Interface. During reasoning, `read()` retrieves the maintained PDDL knowledge, `infer()` generates and refines task specifications and policy code, and `execute()` invokes Z3 and Fast Downward for consistency and feasibility checks. These operations recursively verify and refine the policy, while missing observations trigger exploratory code generation through `infer()`, whose resulting observations are used for further refinement.

Code-as-Symbolic-Planner (CaSP) (Chen et al., 2025) uses an LM to generate executable planner, solver, and checker code as symbolic tools, together with a rule-based Symbolic Complexity Checker that assesses the use of symbolic computation. Within NeSyChain, CaSP is instantiated with Qwen3-4B-Instruct-2507 through the Model Interface, a runtime for generated symbolic code together with a plan validator and the Symbolic Complexity Checker through the Tool Interface, and typed action definitions derived from PDDL accessed through the Knowledge Interface. During reasoning, `read()` retrieves the action definitions, `infer()` generates or refines symbolic planning and checker code, and `execute()` runs and validates the generated code while assessing its symbolic complexity. The resulting feedback guides `infer()` to iteratively refine the code until the plan satisfies correctness and symbolic-complexity criteria.

Across all three methods, these interface mappings preserve each original reasoning flow within the same modular structure. The reimplemented setting retains each reasoning flow without the common lifelong learning procedure, while the lifelong learning setting additionally enables *Knowledge Consolidation* and *Knowledge Compilation*. Table 2 reports the aggregate results, and Table 29 provides the corresponding block-wise measurements.

Table 29: Block-wise lifelong learning performance across reimplemented neurosymbolic configurations on CL-ALFWorld. "-" denotes entries not required for metric computation. Results report mean performance and standard deviation across three random seeds.

Method	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
NL-PDDL	Initial	55.00±5.00	60.00±8.66	23.33±2.89	41.67±10.41	26.67±2.89
	Before Learning	-	70.00±0.00	20.00±0.00	60.00±0.00	30.00±0.00
	After Learning	70.00±10.00	73.33±2.89	86.67±7.64	55.00±17.32	36.67±7.64
	Final	71.67±2.89	73.33±2.89	90.00±8.66	55.00±17.32	36.67±7.64
NeSyRo	Initial	41.67±2.89	76.67±2.89	31.67±2.89	41.67±2.89	26.67±2.89
	Before Learning	-	76.67±2.89	46.67±2.89	56.67±2.89	45.00±0.00
	After Learning	41.67±2.89	76.67±2.89	51.67±2.89	61.67±2.89	51.67±2.89
	Final	41.67±2.89	76.67±2.89	71.67±2.89	61.67±2.89	90.00±0.00
CaSP	Initial	55.00±15.00	48.33±5.77	30.00±5.00	50.00±5.00	15.00±0.00
	Before Learning	-	48.33±5.77	50.00±5.00	56.67±2.89	20.00±0.00
	After Learning	58.33±20.21	48.33±5.77	58.33±2.89	55.00±0.00	20.00±0.00
	Final	71.67±7.64	70.00±5.00	58.33±2.89	61.67±7.64	60.00±0.00

Method	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
NL-PDDL	Initial	40.00±5.00	38.33±7.64	31.67±7.64	53.33±7.64	30.00±5.00
	Before Learning	45.00±0.00	50.00±0.00	55.00±0.00	65.00±0.00	50.00±0.00
	After Learning	55.00±5.00	48.33±16.07	61.67±2.89	76.67±7.64	-
	Final	58.33±5.77	55.00±10.00	60.00±5.00	80.00±8.66	66.67±2.89
NeSyRo	Initial	26.67±2.89	10.00±0.00	66.67±2.89	46.67±2.89	41.67±2.89
	Before Learning	70.00±0.00	36.67±2.89	70.00±0.00	81.67±2.89	76.67±2.89
	After Learning	76.67±2.89	41.67±2.89	70.00±0.00	81.67±2.89	-
	Final	95.00±0.00	61.67±2.89	70.00±0.00	85.00±0.00	76.67±2.89
CaSP	Initial	43.33±2.89	50.00±5.00	35.00±0.00	63.33±10.41	51.67±5.77
	Before Learning	50.00±0.00	71.67±2.89	53.33±7.64	76.67±7.64	70.00±0.00
	After Learning	73.33±2.89	71.67±2.89	53.33±7.64	76.67±7.64	-
	Final	73.33±2.89	71.67±2.89	53.33±7.64	76.67±7.64	70.00±0.00

A.5.4 Ablation of Foundation Models and Symbolic Tool Use in Knowledge Consolidation

We analyze how foundation model choice and symbolic tool use affect *Knowledge Consolidation*. We compare three foundation models for symbol abstraction, each with and without symbolic tool use during consolidation. To isolate these effects, we disable *Knowledge Compilation* in all settings. We first report aggregate lifelong learning performance and then provide block-wise measurements across the sequential task stream.

Table 30: Ablation of foundation model choice and symbolic tool use in *Knowledge Consolidation*, with *Knowledge Compilation* disabled. "Full Consolidation" denotes the complete consolidation procedure. Results report mean performance and standard deviation across three random seeds.

Foundation Model	**Consolidation**	**SR (↑)**	**BWT (↑)**	**FWT (↑)**
GPT-5.6-Sol (Default)	Full Consolidation	88.67 ± 1.61	4.07 ± 1.40	43.89 ± 2.00
GPT-5.6-Sol (Default)	w/o Symbolic Tool	80.83 ± 0.76	1.11 ± 0.56	37.22 ± 2.00
GPT-5.6-Luna	Full Consolidation	77.67 ± 1.26	1.11 ± 0.96	36.11 ± 2.22
GPT-5.6-Luna	w/o Symbolic Tool	51.50 ± 1.32	-5.00 ± 0.00	20.56 ± 2.55
GPT-5.4	Full Consolidation	82.67 ± 0.76	1.48 ± 0.64	40.56 ± 1.92
GPT-5.4	w/o Symbolic Tool	68.17 ± 1.44	16.48 ± 1.40	10.93 ± 1.70

Table 30 shows that both foundation model choice and symbolic tool use affect consolidation performance. Under Full Consolidation, GPT-5.6-Sol achieves 88.67% SR, 4.07 pp BWT, and 43.89 pp FWT, compared with 77.67%, 1.11 pp, and 36.11 pp for GPT-5.6-Luna and 82.67%, 1.48 pp, and 40.56 pp for GPT-5.4. Removing symbolic tool use lowers SR and FWT for all three foundation models, to 80.83% and 37.22 pp for GPT-5.6-Sol, 51.50% and 20.56 pp for GPT-5.6-Luna, and 68.17% and 10.93 pp for GPT-5.4. BWT varies across settings, reaching 1.11 pp, -5.00 pp, and 16.48 pp without symbolic tool use for GPT-5.6-Sol, GPT-5.6-Luna, and GPT-5.4, respectively.

Table 31 provides the corresponding block-wise measurements. Initial denotes performance before learning the stream, Before Learning and After Learning denote performance immediately before and after learning each block, and Final denotes performance after the full stream. Entries not required for BWT or FWT are omitted.

The block-wise results further clarify the differences observed in the aggregate metrics. Under Full Consolidation, Final performance is generally higher than without symbolic tool use across foundation models. For GPT-5.6-Sol, Final performance reaches 91.67% versus 76.67% on Block 5 and 96.67% versus 75.00% on Block 8. The gap is larger for GPT-5.6-Luna on several blocks, including 76.67% versus 15.00% on Block 3 and 86.67% versus 25.00% on Block 6. The Before Learning entries are also generally higher under Full Consolidation, consistent with its higher FWT for all three foundation models. GPT-5.4 is an exception in BWT, where removing symbolic tool use yields 16.48 pp despite lower SR and FWT. This high BWT mainly reflects large recoveries on Blocks 4 and 5, where performance rises from 0.00% After Learning to 68.33% and 73.33% at Final, rather than uniformly strong retention across blocks. Overall, the block-wise measurements support the aggregate finding that foundation model choice affects consolidation performance and that symbolic tool use improves SR and FWT across all three models.

Table 31: Block-wise task performance for the *Knowledge Consolidation* ablation on CL-ALFWorld. "-" denotes entries not required for metric computation. Results provide the block-level measurements underlying the aggregate SR, BWT, and FWT reported in Table 30.

Foundation Model	Consolidation	Entry	Block 1	Block 2	Block 3	Block 4	Block 5
GPT-5.6-Sol	Full Consolidation	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
		Before Learning	-	85.00±0.00	15.00±0.00	76.67±5.77	78.33±2.89
		After Learning	85.00±0.00	85.00±0.00	75.00±5.00	76.67±5.77	88.33±2.89
		Final	85.00±5.00	90.00±0.00	80.00±5.00	95.00±0.00	91.67±2.89
GPT-5.6-Sol	w/o Symbolic Tool	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
		Before Learning	-	85.00±0.00	15.00±0.00	80.00±0.00	73.33±2.89
		After Learning	85.00±0.00	85.00±0.00	75.00±0.00	80.00±0.00	86.67±2.89
		Final	85.00±0.00	85.00±0.00	75.00±0.00	80.00±0.00	76.67±5.77
GPT-5.6-Luna	Full Consolidation	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
		Before Learning	-	85.00±0.00	15.00±0.00	70.00±0.00	75.00±8.66
		After Learning	85.00±0.00	85.00±0.00	75.00±0.00	70.00±0.00	75.00±8.66
		Final	85.00±0.00	85.00±0.00	76.67±2.89	75.00±0.00	75.00±8.66
GPT-5.6-Luna	w/o Symbolic Tool	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
		Before Learning	-	85.00±0.00	15.00±0.00	70.00±0.00	76.67±7.64
		After Learning	85.00±0.00	85.00±0.00	15.00±0.00	70.00±0.00	76.67±7.64
		Final	85.00±0.00	40.00±0.00	15.00±0.00	70.00±0.00	76.67±7.64
GPT-5.4	Full Consolidation	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
		Before Learning	-	85.00±0.00	15.00±0.00	80.00±0.00	71.67±5.77
		After Learning	85.00±0.00	85.00±0.00	75.00±0.00	80.00±0.00	78.33±2.89
		Final	85.00±0.00	88.33±2.89	76.67±2.89	81.67±2.89	78.33±2.89
GPT-5.4	w/o Symbolic Tool	Initial	-	30.00±0.00	10.00±0.00	40.00±0.00	20.00±0.00
		Before Learning	-	85.00±0.00	15.00±0.00	0.00±0.00	0.00±0.00
		After Learning	85.00±0.00	85.00±0.00	0.00±0.00	0.00±0.00	0.00±0.00
		Final	81.67±2.89	80.00±0.00	15.00±0.00	68.33±5.77	73.33±7.64

Foundation Model	Consolidation	Entry	Block 6	Block 7	Block 8	Block 9	Block 10
GPT-5.6-Sol	Full Consolidation	Initial	41.67±2.89	56.67±7.64	36.67±2.89	25.00±5.00	30.00±5.00
		Before Learning	76.67±7.64	78.33±2.89	96.67±2.89	83.33±2.89	95.00±0.00
		After Learning	76.67±7.64	88.33±2.89	96.67±2.89	83.33±2.89	-
		Final	81.67±7.64	88.33±2.89	96.67±2.89	83.33±2.89	95.00±0.00
GPT-5.6-Sol	w/o Symbolic Tool	Initial	41.67±2.89	56.67±7.64	36.67±2.89	25.00±5.00	30.00±5.00
		Before Learning	56.67±5.77	60.00±0.00	75.00±0.00	90.00±0.00	90.00±0.00
		After Learning	56.67±5.77	75.00±0.00	75.00±0.00	90.00±0.00	-
		Final	76.67±2.89	75.00±0.00	75.00±0.00	90.00±0.00	90.00±0.00
GPT-5.6-Luna	Full Consolidation	Initial	41.67±2.89	56.67±7.64	36.67±2.89	25.00±5.00	30.00±5.00
		Before Learning	85.00±5.00	76.67±2.89	65.00±8.66	68.33±2.89	75.00±0.00
		After Learning	85.00±5.00	76.67±2.89	71.67±2.89	68.33±2.89	-
		Final	86.67±5.77	76.67±2.89	73.33±5.77	68.33±2.89	75.00±0.00
GPT-5.6-Luna	w/o Symbolic Tool	Initial	41.67±2.89	56.67±7.64	36.67±2.89	25.00±5.00	30.00±5.00
		Before Learning	25.00±0.00	30.00±0.00	53.33±5.77	65.00±0.00	55.00±0.00
		After Learning	25.00±0.00	30.00±0.00	53.33±5.77	65.00±0.00	-
		Final	25.00±0.00	30.00±0.00	53.33±5.77	65.00±0.00	55.00±0.00
GPT-5.4	Full Consolidation	Initial	43.33±2.89	58.33±5.77	35.00±0.00	28.33±2.89	28.33±5.77
		Before Learning	78.33±2.89	75.00±0.00	73.33±2.89	90.00±0.00	90.00±0.00
		After Learning	78.33±2.89	76.67±2.89	75.00±0.00	90.00±0.00	-
		Final	78.33±2.89	78.33±2.89	76.67±2.89	93.33±5.77	90.00±0.00
GPT-5.4	w/o Symbolic Tool	Initial	41.67±2.89	56.67±7.64	36.67±2.89	25.00±5.00	30.00±5.00
		Before Learning	0.00±0.00	71.67±2.89	66.67±2.89	75.00±0.00	75.00±0.00
		After Learning	75.00±0.00	71.67±2.89	66.67±2.89	75.00±0.00	-
		Final	75.00±0.00	71.67±2.89	66.67±2.89	75.00±0.00	75.00±0.00

B Real-World Deployment Setup and Results

To provide additional details on the real-world evaluation presented in Figure 4, which appears in Section 4.3, we describe the deployment setup and corresponding results. In contrast to the benchmark-scale evaluations, this real-world evaluation focuses on knowledge revision and reuse for real-robot control under realistic perception and manipulation conditions.

B.1 Real-World Deployment Setup



Figure 10: Franka Research 3.

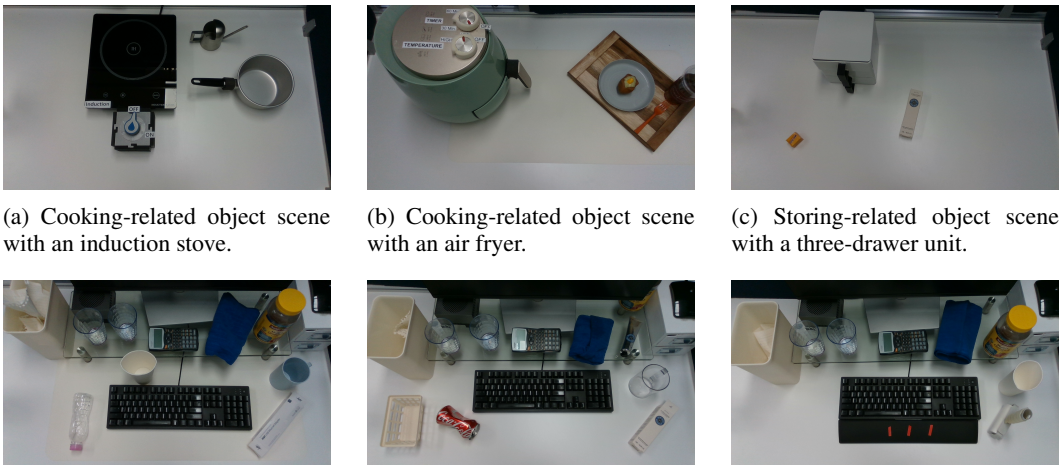
Table 32: Hardware and perception setup.

Component	Configuration
Robot Platform	
Robot	Franka Research 3
Degrees of freedom	7-DoF
End-effector	Two-finger gripper
Workspace	Tabletop
Perception	
Eye-in-hand camera	Intel RealSense D435
Sensor modality	RGB-D
Camera placement	End-effector-mounted
Object segmentation	SAM 3 (Carion et al., 2026)
Grasp pose generation	AnyGrasp (Fang et al., 2023)
External Views	
External cameras	2× Intel RealSense D435
Camera views	Top-down and front views
Purpose	Workspace visualization only
Recording	
Recording cameras	2× smartphone cameras
Purpose	Experimental sequence recording only

Real-robot setup. Figure 10 shows the robot platform used in our real-world experiments, and Table 32 summarizes the hardware and perception setup. We use a 7-DoF Franka Research 3 (FR3) robotic arm with a two-finger gripper, operating in a tabletop workspace. For perception, the robot uses an Intel RealSense D435 RGB-D camera mounted on the end-effector in an eye-in-hand configuration. The captured RGB-D observations are processed by Segment Anything Model 3 (SAM 3) (Carion et al., 2026) for class-agnostic segmentation and by AnyGrasp (Fang et al., 2023) for grasp pose generation. Two additional D435 cameras provide fixed top-down and front views of the workspace for visualization, while two smartphone cameras record the experimental sequences from third-person viewpoints and are not used for robot perception or control.

Table 33: Task-relevant and task-irrelevant objects in the real-world experiments.

Category	Object list
Task-relevant	monitor stand, keyboard, plastic cups, blue rag, trash bin, water bottle, paper cup, plastic container, desk mat, soda can, paper box, candy box, vitamin bottle, dust, lint roller, wrist rest, plastic mug, induction stove, control knob, pot, acrylic mug, hand cream box, three-drawer unit, air fryer, bread, timer knob, temperature knob
Task-irrelevant	monitor, single-board computer, two-drawer unit, calculator, desktop microphone, hand cream tube, kettle, crumpled paper, gray plate, fork, wooden tray, coffee cup



(a) Cooking-related object scene with an induction stove.

(b) Cooking-related object scene with an air fryer.

(c) Storing-related object scene with a three-drawer unit.

(d) Cleaning-related object scene with a keyboard.

(e) Cleaning-related object scene with a soda can.

(f) Cleaning-related object scene with a lint roller.

Figure 11: Representative end-effector views of objects used in real-world experiments.

Real-world objects. We construct the real-world task settings using the objects listed in Table 33. Task-relevant objects include both target objects and other objects whose relations are required for task completion. Task-irrelevant objects are placed in the workspace as surrounding objects but are not involved in satisfying the task goals. For each task scenario, a subset of these objects is selected and arranged on a tabletop within the robot's reachable workspace. Figure 11 shows representative observations containing these objects in the real-world settings.

Table 34: Examples of task instructions and action schemas in the real-world experiments.

	Type	**Example**
Tasks	Cookware preheating	Preheat the appropriate cookware for cooking
	Bread heating	Heat the bread in the air fryer
	Desk cleaning	Clean up the desk
Action Schemas	def pick(object: str):	pick("plastic_mug")
	def place(object: str, target: str):	place("plastic_mug", "monitor_stand")
	def push(object: str, dist: float):	push("keyboard", 0.15)
	def wipe(surface: str, dist: float):	wipe("desk_mat", 0.15)
	def rotate(object: str, angle: float):	rotate("timer_knob", 90.0)

Real-world task scenarios. Table 34 summarizes representative task instructions and the action schemas used in the real-world experiments. The action schemas are executed through the corresponding primitive APIs. We evaluate post-deployment learning in three scenarios. The first scenario examines learning operating constraints from reference manuals through two tasks involving an induction stove and an air fryer. The second scenario examines continual learning from heterogeneous data sources by repeatedly executing the same desk-cleaning task as post-deployment data are introduced sequentially and the resulting knowledge is accumulated. The post-deployment data are obtained sequentially from the agent's own execution feedback, another agent's demonstration, and human-provided knowledge. The third scenario examines recovery from unexpected execution outcomes through two cases involving object displacement during roller cleaning and an unexpected spill during object manipulation. For the overall task performance evaluation, we additionally include three tasks involving a drawer unit, shown in Figure 16, Figure 17, and Figure 18. Where task variants are used, they are constructed by changing task-relevant objects and their relations. All real-world tasks are executed using action schemas listed in Table 34, including pick, place, push, wipe, and rotate.

B.2 Real-World Deployment Results

This section details the real-world results in Figure 4. We evaluate how NeSyChain expands and revises its symbolic knowledge using post-deployment data, with candidate updates verified before incorporation and use in subsequent execution. The agent is initialized with symbolic knowledge from the simulation experiments associated with Table 1b and evaluated in three post-deployment learning scenarios requiring extensions or revisions to this knowledge.

In Figure 4a, we evaluate two tasks involving new appliances, with missing operating constraints acquired from reference manuals. In Figure 4b, we evaluate a single desk-cleaning task across three sequential learning stages, where learning from heterogeneous sources, including the agent's own execution feedback, another agent's demonstration, and human-provided knowledge, progressively improves task quality. In Figure 4c, we evaluate two cases where unexpected outcomes reveal limitations in the knowledge guiding initial execution, with recovery enabled by revising and verifying this knowledge before reuse.

In addition to the seven task instances above, we evaluate three CaP-Bench tasks transferred directly to the real-world setting, shown in Figure 16, Figure 17, and Figure 18, resulting in a total of 10 evaluated task instances. Across all 10 task instances, post-deployment learning increases the overall success rate from 33.3% to 96.7%, as shown in the right panel of Figure 4a.

B.2.1 Learning Operating Constraints from Reference Manuals

This section provides a detailed description of the first scenario in Figure 4a. The scenario evaluates whether NeSyChain can revise its symbolic knowledge to account for operating constraints absent from its existing knowledge when encountering new appliances. We consider two tasks involving an induction stove and an air fryer, which require cookware placement before stove activation and knob reset before air fryer activation, respectively. For both tasks, the agent acquires the missing constraints from reference manuals and verifies the corresponding candidate updates through *Knowledge Consolidation* before incorporating them into its symbolic knowledge for subsequent task execution.

Preheat the appropriate cookware for cooking. In the first task, activating the induction stove requires compatible cookware on the cooking zone. Because the agent encountered no violations of this constraint during the CaP-Bench simulation experiments, its existing symbolic knowledge supports cookware placement and control knob operation but omits this activation precondition.

During post-deployment learning, the agent uses a provided reference manual to identify the missing operating constraint. The symbolic knowledge update introduces the predicate `stove_compatible(obj)` and adds it along with `on_top_of(obj, stove)` to the preconditions for stove activation. This update is verified through *Knowledge Consolidation* before incorporation into the symbolic knowledge. The verified knowledge guides subsequent execution, with the agent placing the pot on the cooking zone before activating the stove.

Figure 12 illustrates the post-deployment execution using the revised knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a pot lies beside the induction stove. Steps 2–4 show the robot picking up the pot and placing it on the cooking zone before activating the stove. Steps 5–7 show the robot grasping the control knob and rotating it to turn on the stove. Finally, Step 8 shows the goal state, where the pot is placed on the cooking zone and the stove is on.

In contrast, Figure 13 shows direct deployment with the existing knowledge, with step indices in the upper-left corner of each image. The task fails because the agent attempts to activate the stove without compatible cookware on the cooking zone, and subsequent pot placement does not activate the stove. Step 1 shows the initial state, where a pot lies beside an induction stove with a control knob attached. Steps 2–4 show the robot grasping and rotating the control knob in an attempt to turn on the stove. Steps 5–7 show the robot picking up the pot and placing it on the cooking zone. Finally, Step 8 shows the task failure state.

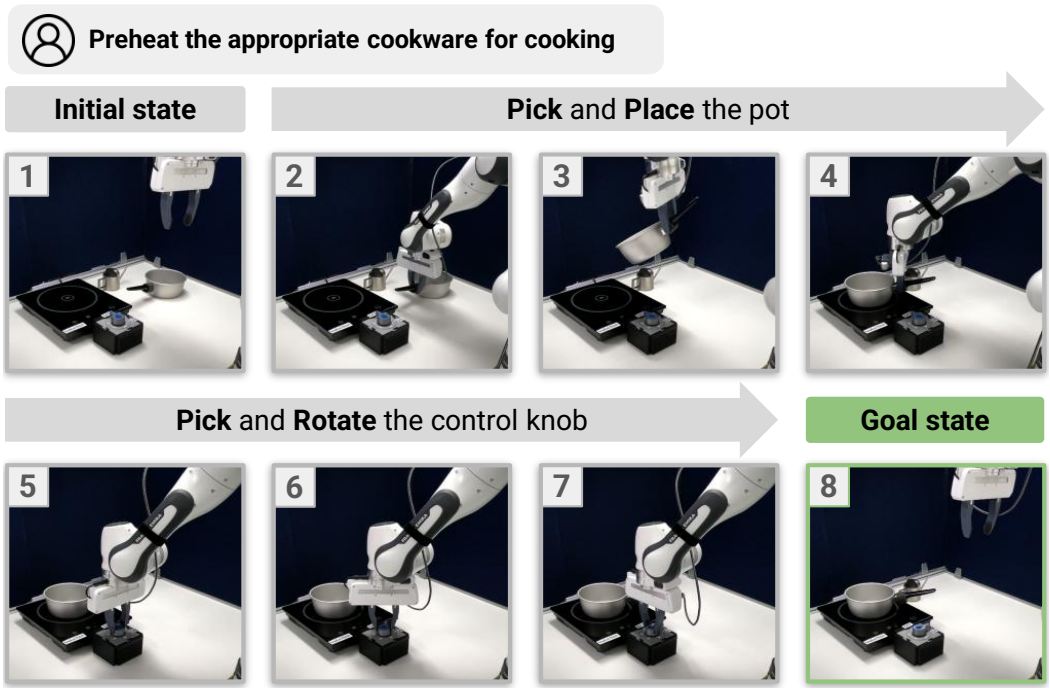


Figure 12: Preheat the appropriate cookware for cooking, post-deployment learning case.

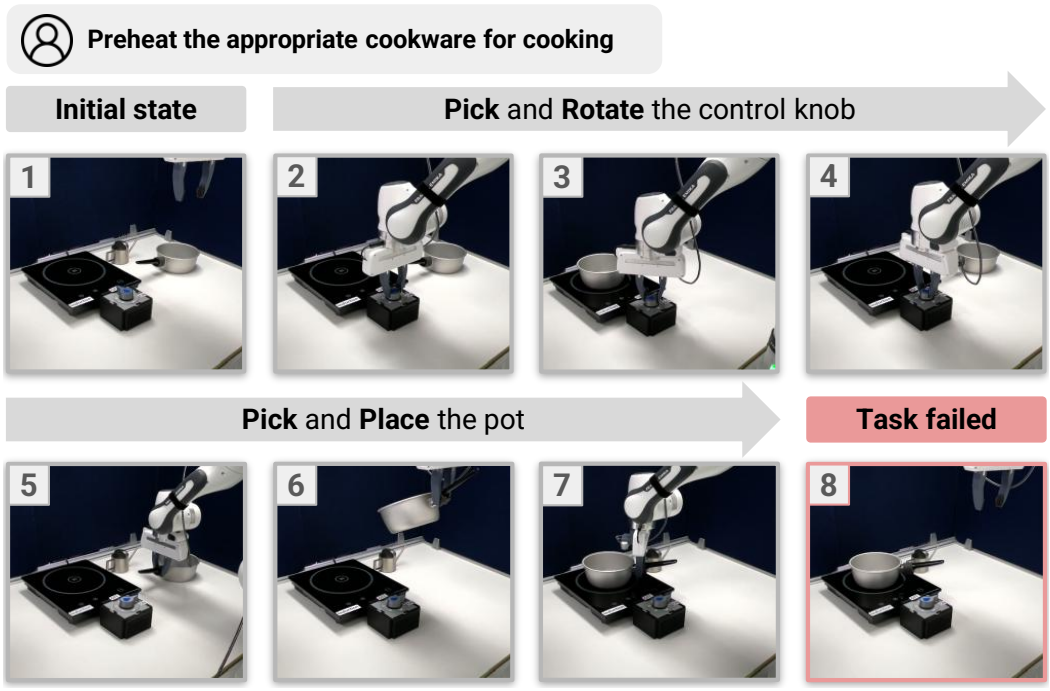


Figure 13: Preheat the appropriate cookware for cooking, direct deployment case.

Heat the bread in the air fryer. In the second task, activating the air fryer requires both the temperature and timer knobs to be reset to OFF. Because the agent had no experience operating an air fryer during the CaP-Bench simulation experiments, its existing symbolic knowledge supports opening and closing the appliance and placing bread inside but omits this reset requirement.

During post-deployment learning, the agent uses a provided reference manual to identify the missing operating constraints. The symbolic knowledge updates introduce the predicate `airfryer_compatible(obj)` and add `temperature-set(off)` and `timer-set(off)` as preconditions for air fryer activation. These updates are verified through *Knowledge Consolidation* before incorporation into the symbolic knowledge. The verified knowledge guides subsequent execution, with the agent resetting both knobs to OFF, placing the bread inside, and then activating the air fryer.

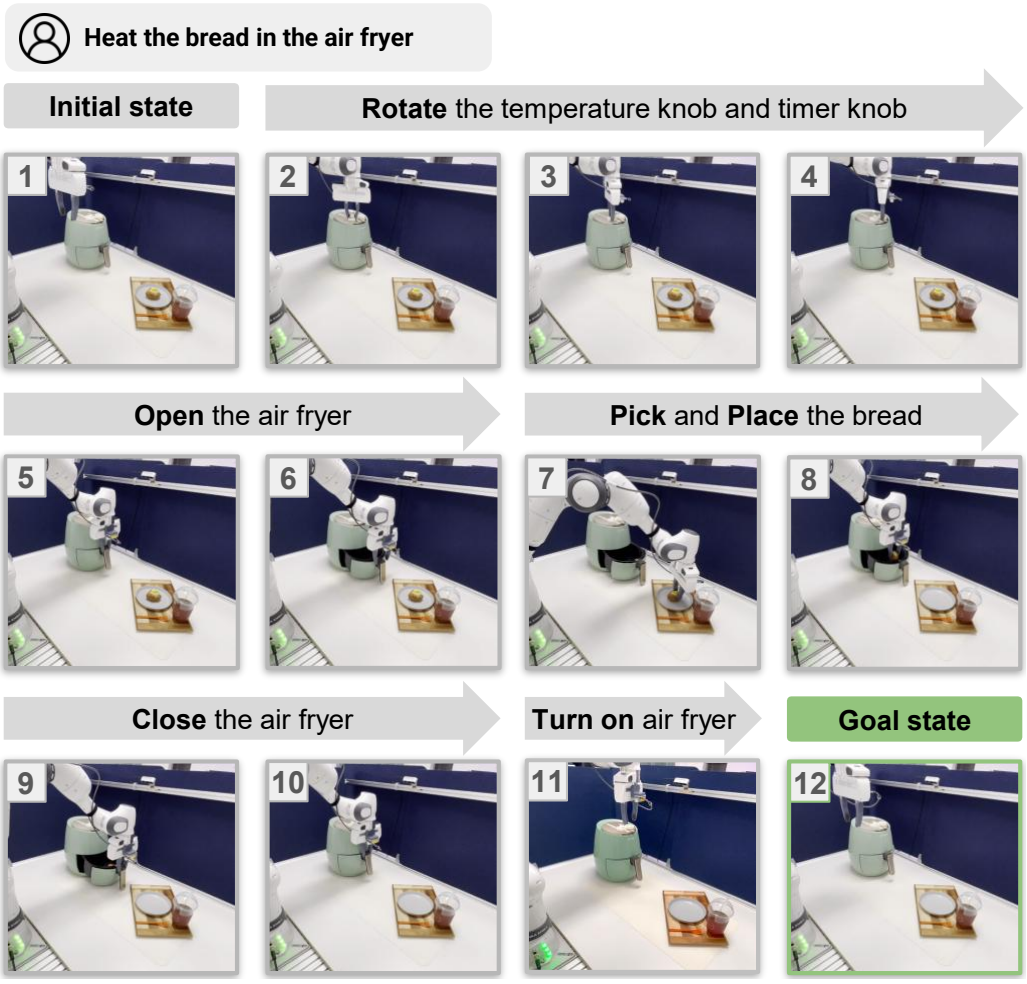


Figure 14: Heat the bread in the air fryer, post-deployment learning case.

Figure 14 illustrates the post-deployment execution using the revised knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where the bread lies on a plate beside the air fryer. Steps 2–4 show the robot rotating the temperature and timer knobs to reset them to OFF. Steps 5–6 show the robot grasping the handle and opening the air fryer. Steps 7–8 show the robot picking up the bread and placing it inside the air fryer. Steps 9–10 show the robot grasping the handle and closing the air fryer. Step 11 shows the robot turning on the air fryer. Finally, Step 12 shows the goal state, where the bread is inside the closed air fryer and the appliance has been activated after the required knob reset.

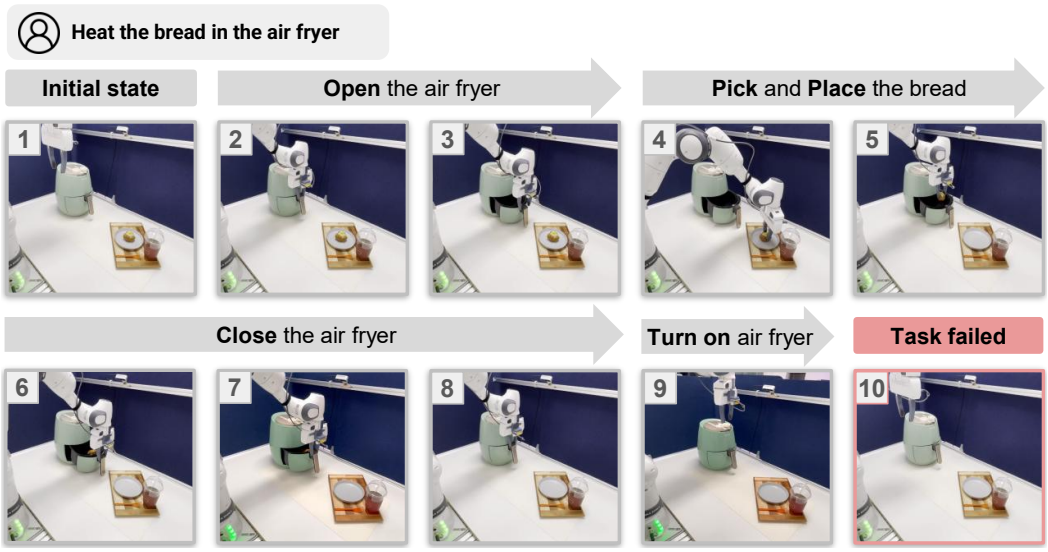


Figure 15: Heat the bread in the air fryer, direct deployment case.

In contrast, Figure 15 shows direct deployment with the existing knowledge, with step indices in the upper-left corner of each image. The task fails because the agent attempts to operate the air fryer without resetting the temperature and timer knobs to OFF. Step 1 shows the initial state, with the bread on a plate beside the air fryer. Steps 2–3 show the robot grasping the handle and opening the air fryer. Steps 4–5 show the robot picking up the bread and placing it inside the air fryer. Steps 6–8 show the robot grasping the handle and closing the air fryer. Step 9 shows the robot grasping and rotating the knob in an attempt to turn on the air fryer. Finally, Step 10 shows the task failure state.

Additional real-world tasks. We additionally evaluate three CaP-Bench tasks transferred directly to the real-world setting, all involving a drawer unit. No additional operating constraints are explicitly imposed on these tasks. Their results are included in the overall success rate reported in the right panel of Figure 4a.

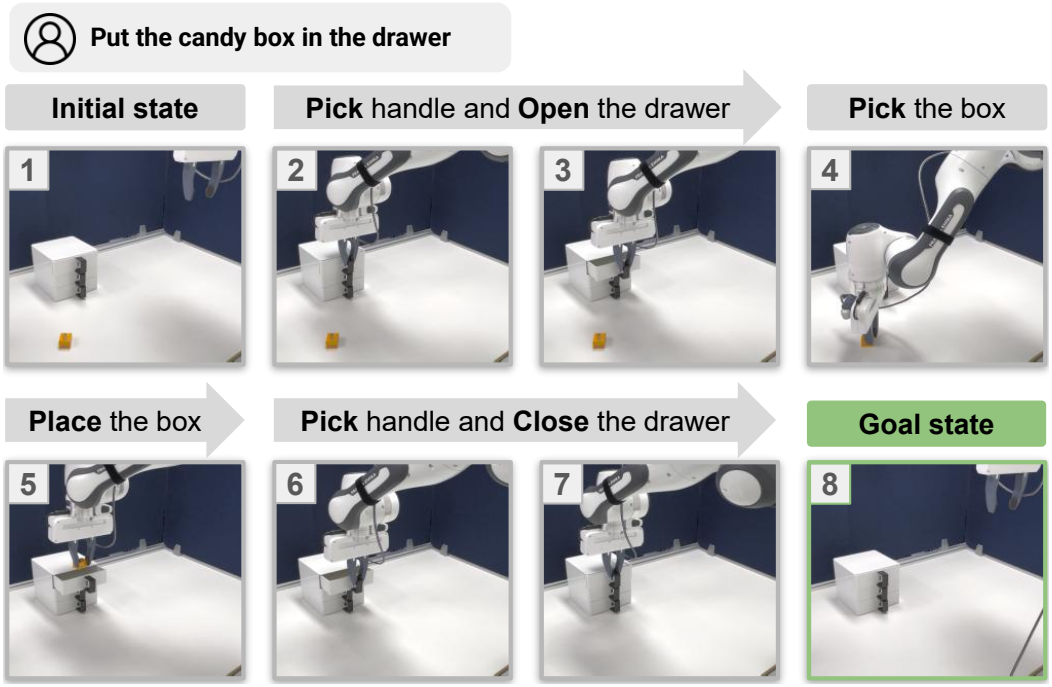


Figure 16: Put the candy box in the drawer.

Figure 16 illustrates the execution of the first task, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a candy box lies on the desk beside a three-drawer unit. Steps 2–3 show the robot grasping the handle and opening the drawer. Steps 4–5 show the robot picking up the candy box and placing it inside the drawer. Steps 6–7 show the robot grasping the handle and closing the drawer. Finally, Step 8 shows the goal state, where the candy box is stored inside the closed drawer.

Figure 17 illustrates the execution of the second task, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a hand cream box and a candy box lie on the desk beside a three-drawer unit. Steps 2–4 show the robot picking up the hand cream box. Steps 5–7 show the robot placing it on top of the drawer unit. Finally, Step 8 shows the goal state, where the hand cream box is placed on top of the drawer unit.

Figure 18 illustrates the execution of the third task, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a hand cream box and a candy box lie on the desk beside a three-drawer unit. Steps 2–4 show the robot picking up the hand cream box and placing it on top of the drawer unit. Steps 5–6 show the robot grasping the handle and opening the drawer. Steps 7–8 show the robot picking up the candy box and placing it inside the drawer. Steps 9–11 show the robot grasping the handle and closing the drawer. Finally, Step 12 shows the goal state, where the candy box is stored inside the closed drawer and the hand cream box is placed on top of the drawer unit.

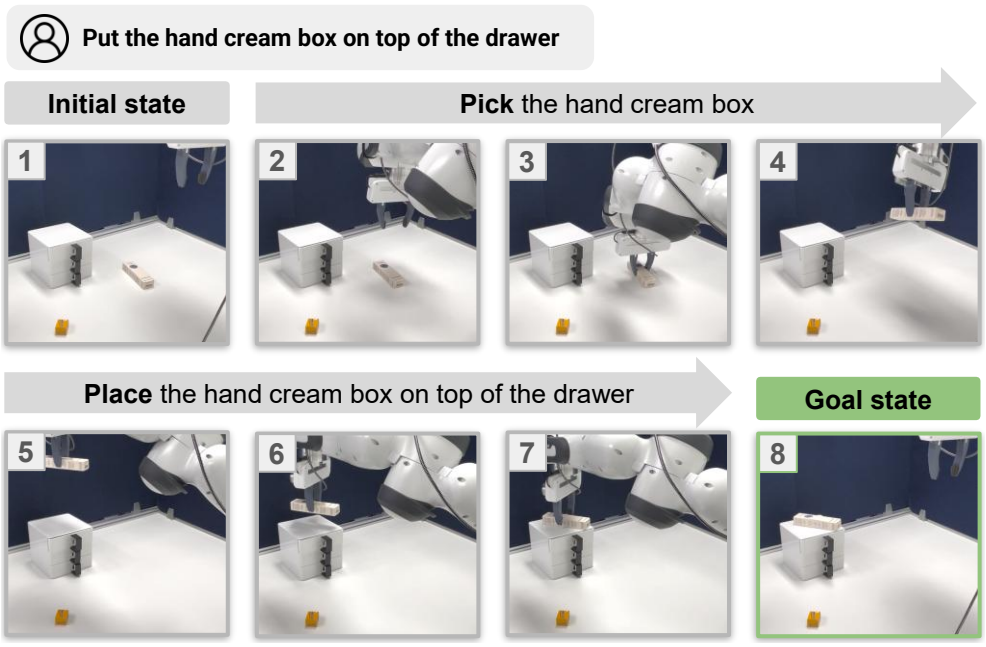


Figure 17: Put the hand cream box on top of the drawer.

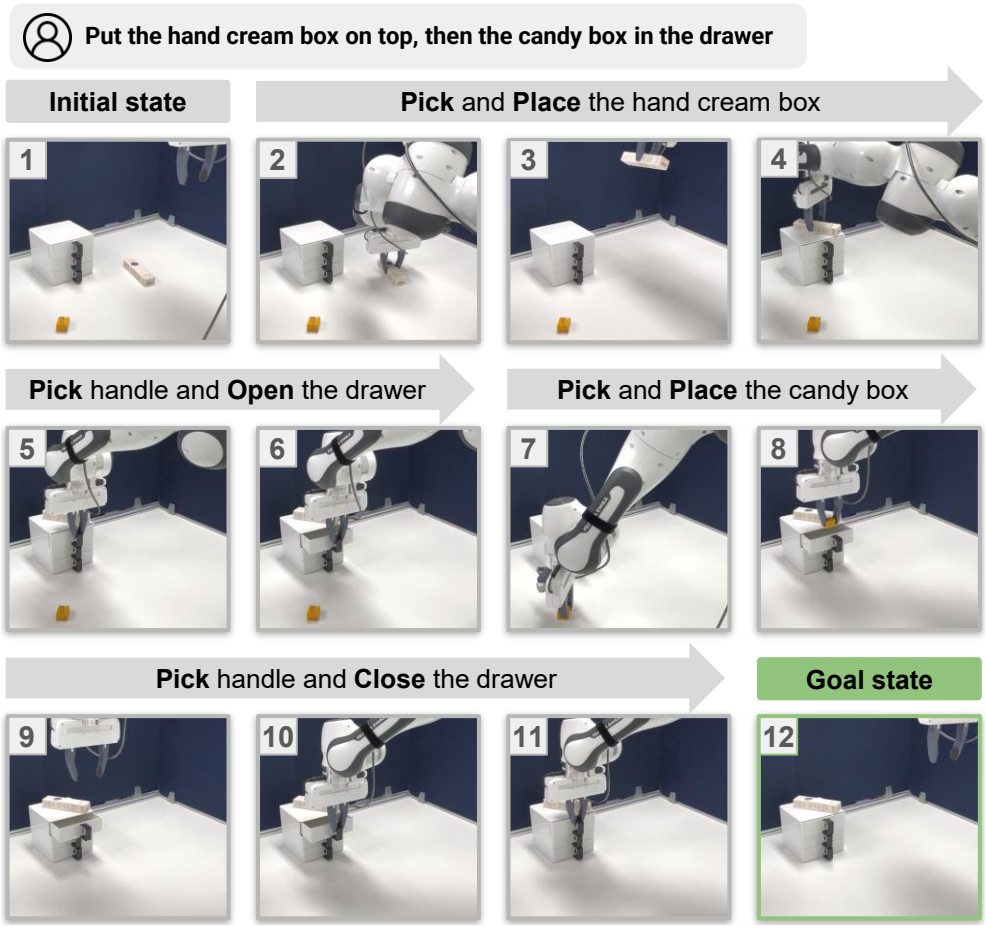


Figure 18: Put the hand cream box on top, then the candy box in the drawer.

B.2.2 Continual Learning from Heterogeneous Data Sources

This section provides a detailed description of the second scenario in Figure 4b. The scenario repeatedly evaluates the same desk-cleaning task while heterogeneous post-deployment data are provided sequentially from different sources. The data are introduced in order from the agent's own execution feedback, another agent's demonstration, and human-provided knowledge. Through *Knowledge Consolidation*, the symbolic knowledge is progressively expanded as each new data source is incorporated, and the agent reuses the accumulated knowledge in subsequent task executions, enabling it to clear and clean progressively larger areas of the desk mat. Figure 19 shows this progression in task quality as heterogeneous post-deployment data are sequentially incorporated.



(a) Resulting cleaning area after learning from the agent's own execution feedback.

(b) Resulting cleaning area after additionally learning from another agent's demonstration.

(c) Resulting cleaning area after additionally learning from human-provided knowledge.

Figure 19: Top-down views showing the progressive expansion of the cleaning area as heterogeneous post-deployment data are sequentially incorporated.

In Figure 19a, the robot relocates the water bottle, paper cup, paper box, and plastic mug to the far left of the desk mat, leaving an irregular cleaning area constrained by the gathered objects. In Figure 19b, after acquiring stacking knowledge from another agent's demonstration, the robot stacks the plastic cups and places the plastic mug in the freed space, exposing a larger area of the mat for cleaning. In Figure 19c, human-provided knowledge enables the robot to create additional space in a full trash bin through bin pressing, allowing it to dispose of the paper cup, water bottle, and paper box while retaining the previously acquired behaviors and leaving most of the mat available for cleaning. The following paragraphs describe how the agent learns from each sequentially introduced data source, verifies the acquired knowledge before use, and reuses the knowledge during task execution.

Learning from own execution feedback. In the first stage, the agent learns from execution feedback collected through its own interaction with the environment. During the initial execution, the agent attempts to clear objects from the desk mat before wiping, primarily through pick-and-place operations, but the keyboard cannot be successfully picked up, resulting in an incomplete task.

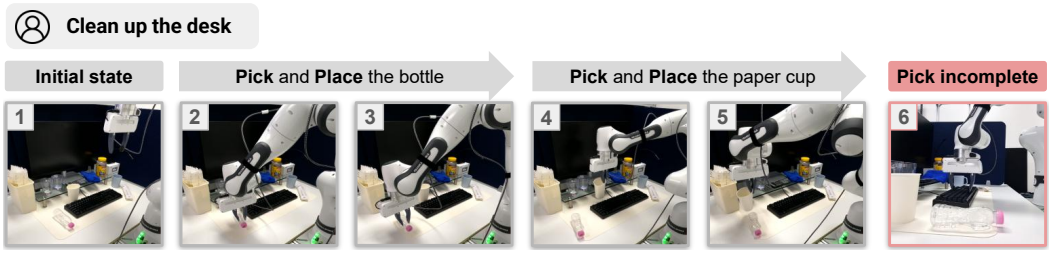


Figure 20: FR3 desk-cleaning task with a keyboard pick attempt, direct deployment case.

Figure 20 illustrates task execution using the existing symbolic knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a water bottle, a paper cup, a keyboard, a plastic mug, and a paper box obstruct the target wiping region. Steps 2–5 show the robot picking up the water bottle and paper cup and placing them on the left side of the desk mat. Finally, Step 6 shows the robot attempting to pick up the keyboard, but the pick operation does not complete, leaving the task incomplete.

The agent's existing symbolic knowledge supports `Pick`, `Place`, `Push`, and `Wipe` operations for desk cleaning, but does not fully specify the effects of `Push`. In particular, the existing knowledge represents that pushing can make an object no longer block a surface but does not encode the resulting relocation as `moved(obj, loc)`. Consequently, the planner cannot use `Push` to establish `moved(obj, loc)`, which is required for object clearing. From the incomplete execution trace, NESYCHAIN identifies a candidate revision to the `Push` operation that adds `moved(obj, loc)` to its effects. Through *Knowledge Consolidation*, the candidate revision is verified before being incorporated into the symbolic knowledge. The verified symbolic knowledge then provides a reliable basis for symbolic planning, allowing the planner to derive a plan that pushes the keyboard under the monitor stand instead of attempting to pick it up.

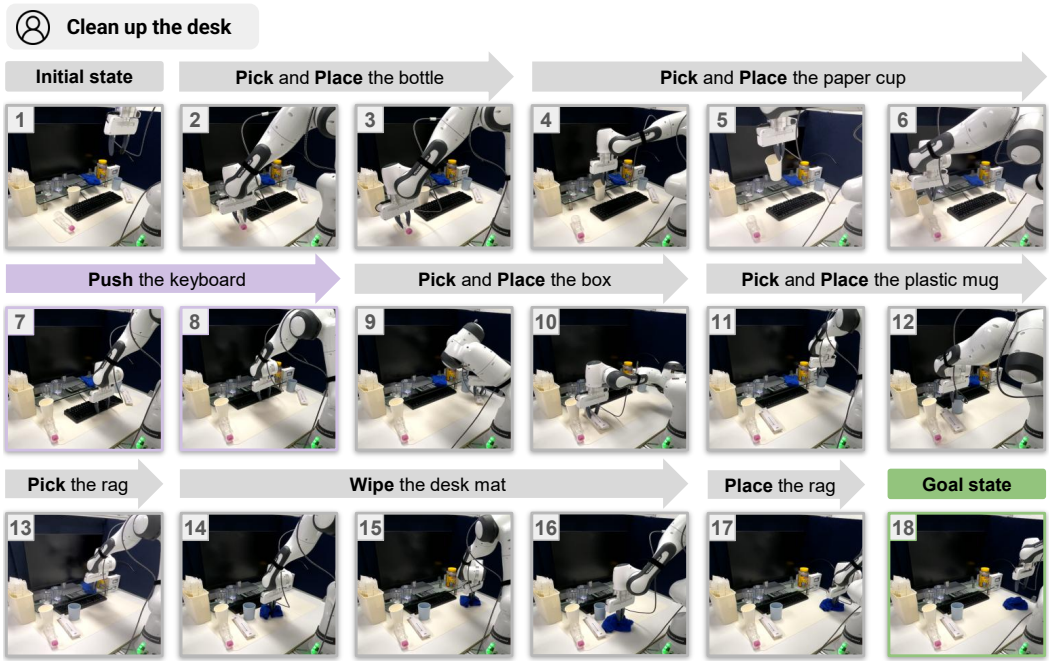


Figure 21: FR3 desk-cleaning task execution sequence after post-deployment learning from the agent's own execution feedback. Purple arrows (⟶) indicate push-based keyboard relocation enabled by the revised symbolic knowledge.

Figure 21 illustrates the post-deployment execution using the revised symbolic knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state. Steps 2–3 show the robot picking up the water bottle and placing it on the left side of the desk mat. Steps 4–6 show the robot picking up the paper cup and placing it on the left side of the desk mat. Steps 7–8 show the robot pushing the keyboard under the monitor stand, rather than attempting to pick it up as in the initial execution. Steps 9–12 show the robot picking up the paper box and the plastic mug and placing them on the left side of the desk mat. Step 13 shows the robot picking up the blue rag. Steps 14–16 show the robot wiping the exposed area of the desk mat. Step 17 shows the robot placing the rag at the designated location. Finally, Step 18 shows the goal state, where the obstructing objects have been cleared from the wiping region, the exposed surface has been wiped, and the rag has been placed at its designated location.

Learning from another agent's demonstration. In the second stage, the agent learns from a cube-stacking demonstration performed by another agent. During the demonstration, an SO-101 arm sequentially picks up and places the cubes to form a stack.

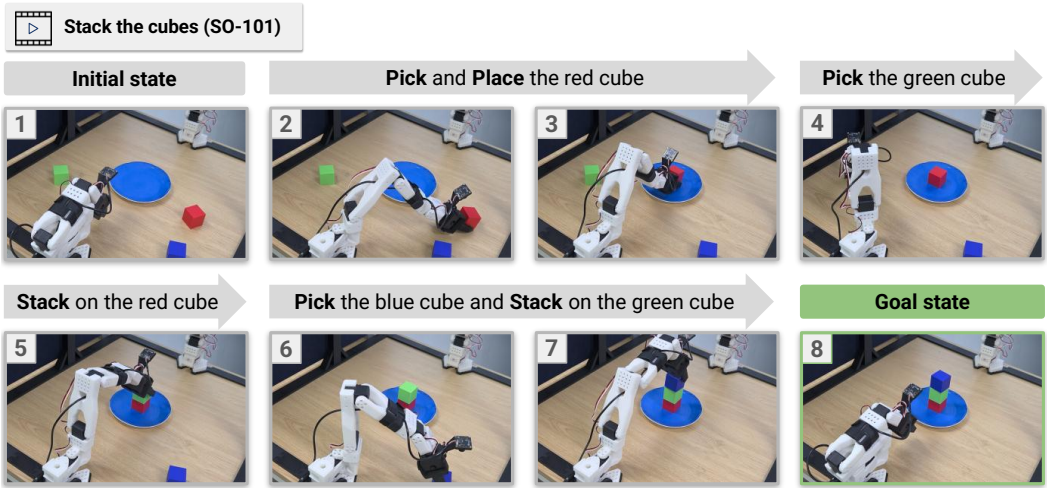


Figure 22: SO-101 cube-stacking demonstration used to acquire stacking knowledge.

Figure 22 illustrates a cube-stacking demonstration performed by an SO-101 arm, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where multiple cubes are placed on the desk. Steps 2–3 show the robot picking up the red cube and placing it on the blue plate. Steps 4–5 show the robot picking up the green cube and stacking it on the red cube. Steps 6–7 show the robot picking up the blue cube and stacking it on the green cube. Finally, Step 8 shows the goal state, where the cubes are stacked.

The agent's existing symbolic knowledge supports `Pick`, `Place`, `Push`, and `Wipe` operations for desk cleaning, but does not account for stacking objects to reduce the desk area they occupy. In particular, the existing knowledge does not represent which objects can be stacked or include an action for stacking one object on top of another. From the cube-stacking demonstration, NeSyChain identifies a candidate revision that adds the `stackable` predicate and introduces the `Stack` action. Through *Knowledge Consolidation*, the candidate revision is verified before being incorporated into the symbolic knowledge. The verified symbolic knowledge then provides a reliable basis for symbolic planning, allowing the planner to apply the newly acquired `Stack` action to plastic cups during subsequent task execution. The agent also retains the previously acquired `Push` knowledge, which the planner reuses to relocate the keyboard under the monitor stand. By stacking the plastic cups and placing the plastic mug in the freed space, the agent exposes a larger area of the desk mat for cleaning.

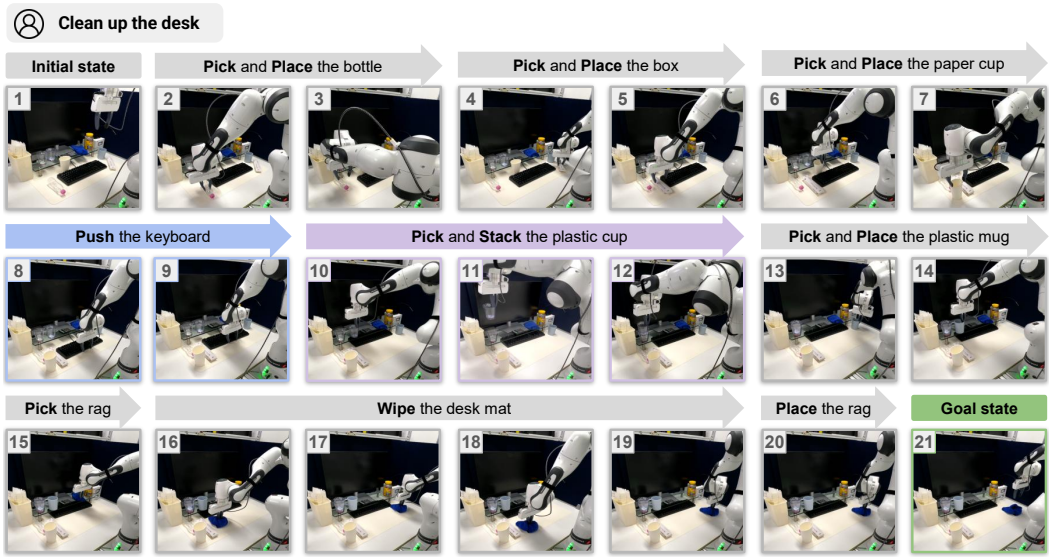


Figure 23: FR3 desk-cleaning task after post-deployment learning from another agent's demonstration. Purple arrows (⟶) indicate stacking enabled by the newly acquired `Stack` action, while blue arrows (⟶) indicate push-based keyboard relocation enabled by the previously acquired pushing knowledge.

Figure 23 illustrates the post-deployment execution using the revised knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a water bottle, a paper box, a paper cup, a keyboard, two plastic cups, and a plastic mug obstruct the target wiping region. Steps 2–7 show the robot picking up the water bottle, the paper box, and the paper cup and placing them on the left side of the desk mat. Steps 8–9 show the robot pushing the keyboard under the monitor stand using the previously acquired pushing knowledge. Steps 10–12 show the robot applying the newly acquired stacking knowledge by picking up one plastic cup and stacking it on the other. Steps 13–14 show the robot picking up the plastic mug and placing it in the space freed by stacking the plastic cups. Steps 15–19 show the robot picking up the blue rag and wiping the desk mat, and Step 20 shows the robot placing the rag at the designated location. Finally, Step 21 shows the goal state, where the target objects have been cleared from the wiping region, a larger area of the desk mat has been exposed and wiped, and the rag has been placed at its designated location.

Learning from human-provided knowledge. In the third stage, the agent learns from human-provided knowledge. The provided knowledge states that pressing down the contents of a full trash bin can create space for additional items.

The agent's existing symbolic knowledge supports `Pick`, `Place`, `Push`, `Stack`, and `Wipe` operations for desk cleaning, but does not account for creating additional space in a full trash bin. In particular, the existing knowledge does not include an action for pressing down the contents of a full trash bin to create additional free space. Consequently, before incorporating the human-provided knowledge, the agent cannot use the full trash bin to dispose of additional objects, limiting the area available for cleaning.

From the human-provided knowledge, NESYCHAIN identifies a candidate revision that introduces the `Bin_press` action with `free_space(bin)` as an effect. Through *Knowledge Consolidation*, the candidate revision is verified before being incorporated into the symbolic knowledge. The verified symbolic knowledge then provides a reliable basis for symbolic planning, allowing the planner to use the newly acquired `Bin_press` action to create space in the full trash bin and dispose of additional objects. The agent also retains the previously acquired `Push` and `Stack` knowledge, which the planner reuses during subsequent task execution. By disposing of additional objects while reusing the previously acquired behaviors, the agent leaves most of the desk mat available for cleaning.

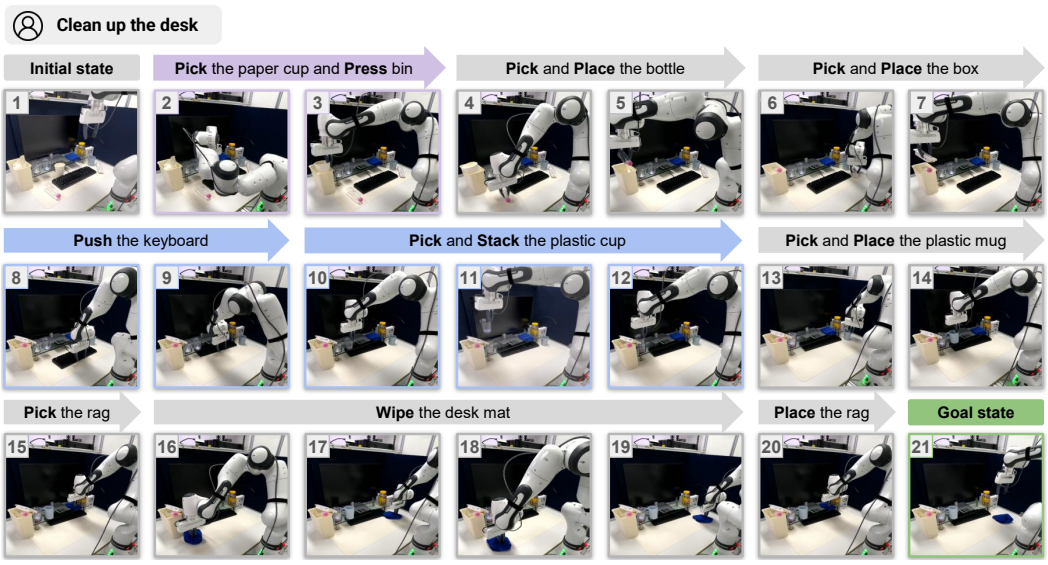


Figure 24: FR3 desk-cleaning task after post-deployment learning from human-provided knowledge. Purple arrows (→) indicate bin pressing enabled by the newly acquired `Bin_press` action, while blue arrows (→) indicate reuse of the previously acquired pushing and stacking knowledge.

Figure 24 illustrates the post-deployment execution using the revised knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a water bottle, a paper cup, a keyboard, two plastic cups, a plastic mug, and a paper box obstruct the target wiping region. The trash bin is initially full. Steps 2–3 show the robot picking up the paper cup and pressing down the contents of the full bin before disposing of the cup. Steps 4–5 show the robot picking up the water bottle and placing it in the trash bin. Steps 6–7 show the robot picking up the paper box and placing it in the trash bin. Steps 8–9 show the robot pushing the keyboard under the monitor stand using the previously acquired pushing knowledge. Steps 10–12 show the robot reusing the previously acquired stacking knowledge by picking up one plastic cup and stacking it on the other. Steps 13–14 show the robot picking up the plastic mug and placing it in the space freed by stacking the plastic cups. Steps 15–19 show the robot picking up the blue rag and wiping the exposed area of the desk mat. Step 20 shows the robot placing the rag at the designated location. Finally, Step 21 shows the goal state, where the target objects have been cleared from the wiping region, the exposed surface has been wiped, and the rag has been placed at its designated location.

B.2.3 Recovery from Unexpected Execution Outcomes

This section provides a detailed description of the third scenario shown in Figure 4c. The scenario evaluates whether NeSyChain can recover from unexpected execution outcomes by revising the preconditions and effects represented in its symbolic knowledge. We consider two desk-cleaning cases involving object displacement during roller cleaning and an unexpected spill during task execution. Across the two cases, the agent revisits an earlier demonstration or uses human feedback to identify missing knowledge, verifies the resulting revisions, and reuses the updated knowledge in subsequent task execution.

Desk-cleaning with a lint roller. In the first case, the agent initially learns how to clean a dusty surface with a lint roller from the demonstration shown in Figure 25.

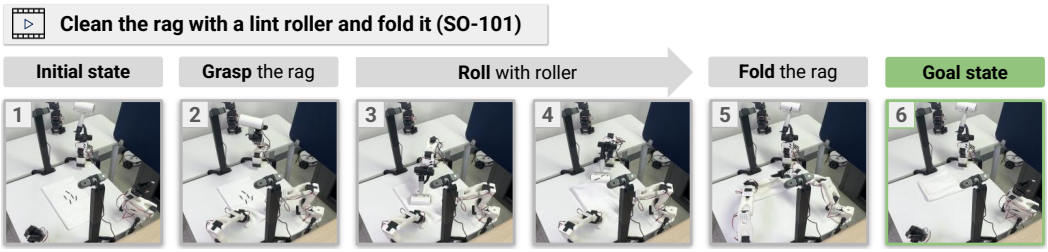


Figure 25: SO-101 roller-cleaning demonstration used to acquire and refine roller-cleaning knowledge.

Figure 25 illustrates a roller-cleaning demonstration performed using three SO-101 arms, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a rag is on the table and one arm is grasping a lint roller. Step 2 shows two arms holding the rag in place. Steps 3–4 show another arm rolling the lint roller over the rag while the rag remains held in place. Step 5 shows the rag being folded. Finally, Step 6 shows the goal state with the cleaned rag folded.

From this demonstration, the agent acquires the `Roll_clean` schema and the predicates `is_roller(obj)` and `dusty(loc)`. The candidate symbolic knowledge is verified against the available traces before being incorporated into the agent's symbolic knowledge. However, the acquired schema does not specify that the target surface must remain braced during rolling. The agent therefore derives a plan that is valid under the current symbolic knowledge but applies `Roll_clean` without first securing the wrist rest.

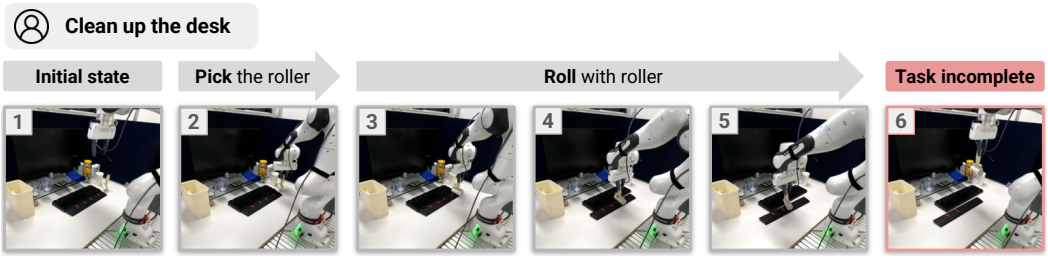


Figure 26: FR3 desk-cleaning execution sequence with a lint roller after the initial demonstration-based knowledge update.

Figure 26 illustrates execution using the initially acquired roller-cleaning knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state, where a keyboard, a dusty wrist rest, a lint roller, and a paper cup lie on the desk. Step 2 shows the robot picking up the lint roller. Steps 3–5 show the robot applying the lint roller to the dusty wrist rest. Finally, Step 6 shows the unexpected outcome, where the wrist rest is displaced during rolling and the task remains incomplete.

Following this unexpected outcome, the agent revisits the demonstration and observes that two arms hold the rag in place while the third applies the lint roller. The agent identifies target bracing as a missing condition and proposes adding `braced(loc)` as a precondition of `Roll_clean`, where `loc` denotes the target surface. The existing effects, `not dusty(loc)` and `clean(loc)`, remain unchanged. Through *Knowledge Consolidation*, the candidate revision is verified before being incorporated into the symbolic knowledge. Using the revised knowledge, the agent satisfies the bracing condition by placing a vitamin bottle beside the wrist rest as a stopper before rolling, as shown in Figure 27.

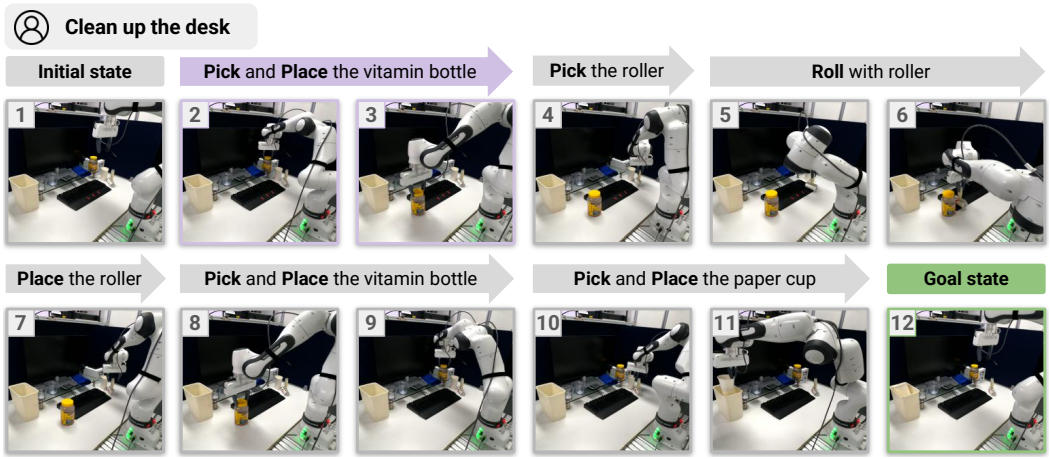


Figure 27: FR3 desk-cleaning execution sequence with a lint roller after replay-based knowledge refinement. Purple arrows (→) indicate placement of the vitamin bottle as a stopper to brace the wrist rest before rolling.

Figure 27 illustrates the post-deployment execution using the revised knowledge, with step indices shown in the upper-left corner of each image. Step 1 shows the initial state. Steps 2–3 show the robot picking up the vitamin bottle and placing it beside the wrist rest as a stopper to satisfy the newly introduced bracing condition. Step 4 shows the robot picking up the lint roller. Steps 5–6 show the robot rolling the lint roller over the braced wrist rest without displacing it. Step 7 shows the robot placing the lint roller back on the desk. Steps 8–9 show the robot picking up the vitamin bottle and returning it to its original position. Steps 10–11 show the robot picking up the paper cup and placing it in the trash bin. Finally, Step 12 shows the goal state, where the wrist rest has been cleaned without displacement during roller cleaning and the remaining object has been cleared from the workspace.

Desk-cleaning with an unexpected spill. In the second case, the agent begins the desk-cleaning task using existing symbolic knowledge that supports `Pick`, `Place`, `Push`, and `Wipe`. During object manipulation, the remaining contents of a soda can unexpectedly spill onto the desk. Although `Wipe` is already available, the existing symbolic knowledge does not include the predicate `spilled(loc)` or specify the effects of wiping on a spill.

Using human feedback about the spill as post-deployment data, the agent proposes a knowledge update that introduces `spilled(loc)` and extends the effects of `Wipe`. The proposed update retains the existing preconditions and the unconditional effect `wiped(loc)`. It adds `not spilled(loc)` and `clean(loc)` as conditional effects that apply when `spilled(loc)` holds before wiping. Through *Knowledge Consolidation*, the candidate update is verified before being incorporated into the symbolic knowledge. The agent then uses the revised knowledge to resume task execution from the current state, wiping the spilled area before continuing to clear the remaining objects.

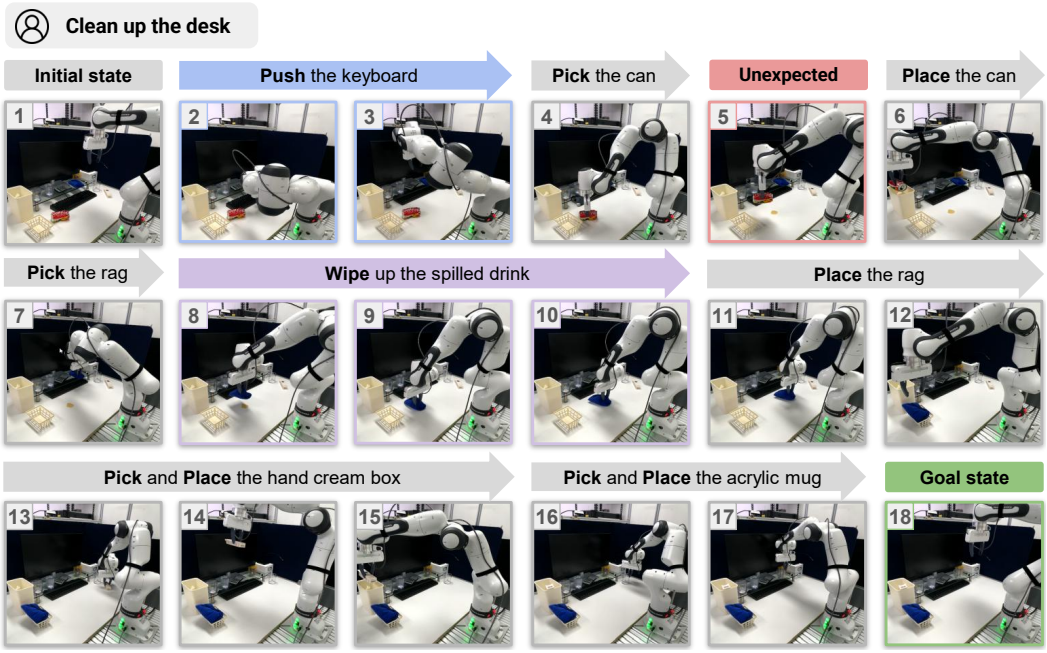


Figure 28: FR3 desk-cleaning execution sequence with recovery from an unexpected spill. Purple arrows (→) indicate spill cleanup using the revised `Wipe` schema following human feedback, while blue arrows (→) indicate keyboard relocation using the previously acquired `Push` schema.

Figure 28 illustrates a single execution sequence in which the agent encounters the spill and subsequently resumes the task using the revised symbolic knowledge. Step 1 shows the initial state, where a keyboard, a soda can, a blue rag, a hand cream box, and an acrylic mug lie on the desk. Steps 2–3 show the robot pushing the keyboard under the monitor stand. Step 4 shows the robot picking up the soda can. Step 5 shows the unexpected outcome, where the remaining contents spill onto the desk while the can is being moved. Step 6 shows the robot placing the soda can in the trash bin. Following the verified knowledge update, Steps 7–12 show the robot picking up the blue rag, wiping the spilled area, and placing the rag in the designated plastic container. Steps 13–15 show the robot picking up the hand cream box and placing it in the trash bin. Steps 16–17 show the robot picking up the acrylic mug and placing it on the monitor stand. Finally, Step 18 shows the goal state, where the task-relevant objects have been cleared from the cleaning area, the spill has been cleaned, and the rag has been placed in the plastic container.