

1	<b>Contents</b>	
2	<b>A NESYPR: Neurosymbolic Proceduralization</b>	<b>2</b>
3	A.1 Algorithm . . . . .	2
4	A.2 Implementation and Hyperparamter Setting . . . . .	2
5	<b>B Evaluation</b>	<b>4</b>
6	B.1 Experiment Setting . . . . .	4
7	B.1.1 PDDLGym . . . . .	7
8	B.1.2 VirtualHome . . . . .	8
9	B.1.3 ALFWorld . . . . .	9
10	B.1.4 Baselines . . . . .	11
11	B.1.5 Metrics . . . . .	15
12	B.2 Additional Experimental Results . . . . .	16
13	B.2.1 Open-loop Continual Embodied Tasks . . . . .	16
14	B.2.2 Closed-loop Continual Embodied Tasks . . . . .	17
15	B.2.3 Analysis on Proceduralization . . . . .	18
16	B.2.4 Analysis on Continual Embodied Task Learning . . . . .	18
17	B.2.5 Evaluation on Inference Efficiency Across Diverse Devices. . . . .	19

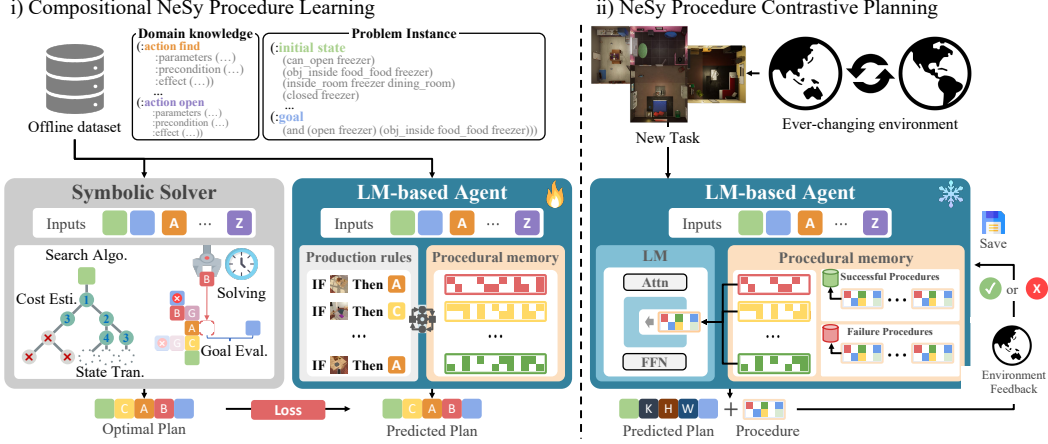


Figure 1: The framework architecture of NESYPR

## A NESYPR: Neurosymbolic Proceduralization

To equip agents with structured and adaptive reasoning for diverse embodied tasks, NESYPR learns to encode production rules and compose procedures, compressed representations derived from the declarative knowledge of symbolic tools. We refer to this end-to-end learning and utilization process as *neurosymbolic proceduralization*. As illustrated in Figure 1, neurosymbolic proceduralization operates in two phases: i) a training phase, *compositional NeSy procedure learning*, where procedural knowledge is structured within procedural memory using plans generated by a symbolic tool, and ii) a test phase, *NeSy procedure contrastive planning*, where the agent autonomously adapts to new tasks by contrastively reconstructing procedural representations without relying on symbolic tools.

### A.1 Algorithm

During phase i), the agent trains on offline data comprising symbolically defined problem instances (observations and goals) and associated domain knowledge (action rules). The declarative knowledge used by symbolic tools for problem-solving—such as search algorithms, state transitions, cost estimation, and goal evaluation—is internalized as production rules. The agent composes these rules into task-solving procedures in procedural memory, which it then exploits to generate plans. The algorithm for phase i) is provided in Algorithm 1.

In phase ii), using the procedural memory established during phase i), the agent performs structured reasoning without access to external symbolic tools (i.e., declarative memory). It further engages in adaptive reasoning by contrastively reconstructing procedures from prior ones labeled as successes or failures via environmental feedback. The agent continually reinforces plans aligned with valid procedures while suppressing those associated with invalid ones. The algorithm for phase ii) is provided in Algorithm 2.

Accordingly, NESYPR enables LM-based agents to reason robustly across tasks and adapt efficiently to ever-changing environments.

### A.2 Implementation and Hyperparameter Setting

We instantiate NESYPR on a set of backbone LMs and equip it with a memory-augmented module that operates during both training and inference. During the *Compositional NeSy Procedure Learning* phase, symbolic reference plans are produced by the symbolic planner [1] and encoded into a procedure-book  $\mathcal{C}$  within procedural memory. The procedure-book  $\mathcal{C}$  is composed of  $K$  procedure-units of dimension  $d$ . We set  $(K, d) = (256, 32)$  for PDDL Gym and  $(224, 28)$  for VirtualHome and ALFWorld. NESYPR is trained with AdamW ( $\text{lr} = 2 \times 10^{-4}$ ,  $\text{batch} = 1$ ) for 50 epochs on PDDL Gym and 20 epochs on VirtualHome and ALFWorld.

At inference time, during NeSy Procedure Contrastive Planning, the agent retrieves procedures from the positive ( $\mathcal{M}^+$ ) and negative ( $\mathcal{M}^-$ ) procedure banks based on cosine similarity, using a

---

**Algorithm 1** Compositional NeSy Procedure Learning

---

**Initialize** working memory  $M \leftarrow [e_1, e_2, \dots, e_S]$ **Initialize** procedure-book  $\mathcal{C} \leftarrow \{c_1, c_2, \dots, c_K\}$ Memory-augmented LM policy  $\pi_{\text{LM}}$ Offline dataset  $D$ Symbolic planner  $\pi_{\text{tool}}$ 

```
1: for all  $(o, g, \text{DK}) \in D$  do
2:    $a \leftarrow \pi_{\text{tool}}(o, g, \text{DK})$   $\triangleright$  Calculate action plan  $a$  based on domain knowledge DK
   /* LM forward pass */
3:    $H_0 \leftarrow \text{EmbeddingLayer}(o, g, \text{DK})$ 
4:   for  $l = 1$  to  $L$  do
     /* DecoderBlockl */
5:      $E_{\text{self}} \leftarrow \text{SelfAttention}(H_{l-1})$ 
6:      $E_{\text{work}} \leftarrow \text{CrossAttention}(E_{\text{self}}, M)$   $\triangleright$  cf. Eq. (3)
7:     Update  $M$  by gated merge of  $E_{\text{work}}$  and  $M$   $\triangleright$  cf. Eq. (4)
8:      $R \leftarrow \text{VQ}(M, \mathcal{C})$   $\triangleright$  cf. Eq. (5), (6)
9:      $H_l \leftarrow \text{FFN}(E_{\text{work}} + \text{Gate}(R))$   $\triangleright$  cf. Eq. (7)
10:  end for
   /* Loss computation and model update */
11:   $\mathcal{L} = -\sum \log \pi_{\text{LM}}(a \mid o, g, M) + \sum \mathcal{L}_{\text{VQ}}$   $\triangleright$  cf. Eq. (8), (9)
12:  Update  $\pi_{\text{LM}}$  parameter  $\theta$  via loss  $\mathcal{L}$ 
13:  Update  $\mathcal{C}$  via EMA
14: end for
```

---

reconstruction threshold of  $v = 0.95$ . Contrastive decoding is applied with a truncation threshold  $\vartheta = 0.1$ , promoting selection of valid actions while suppressing failure patterns. Generation is performed using deterministic decoding with top- $p = 1.0$  and temperature 0.0.

All experiments run on Python 3.10, PyTorch 2.3.0, and Transformers 4.47.1 with an Intel i9-10980XE CPU and a single NVIDIA RTX A6000 GPU. A concise summary of the hyperparameter settings is provided in Table 1.

Table 1: Hyperparameter settings for NESYPR

Setting	PDDL Gym	VirtualHome	ALFWorld
Procedure-book size $K$	256	224	224
Procedure-unit dim. $d$	32	28	28
Training epochs	50	20	20
Optimizer / lr	AdamW / $2 \times 10^{-4}$		
Batch size	1		
EMA decay ( $\mathcal{C}$ )	0.99		
Reconstruction threshold $v$	0.95		
Contrastive trunc. $\vartheta$	0.10		
Decoding top- $p$ / temp.	1.0 / 0.0		

---

**Algorithm 2** NeSy Procedure Contrastive Planning

---

**Initialize** working memory  $M \leftarrow [e_1, e_2, \dots, e_S]$   
**Initialize** procedure banks  $\mathcal{M}^+ \leftarrow \{\}, \mathcal{M}^- \leftarrow \{\}$   
pretrained procedure-book  $\mathcal{C}$   
pretrained LM policy  $\pi_{\text{LM}}$   
Environment  $env$

```
1: for all  $\tau \in \mathcal{T}$  do
2:    $t \leftarrow 0$ 
3:    $(o_t, g, \text{DK}) \leftarrow env.\text{reset}(\tau)$ 
4:    $\text{done} \leftarrow \text{False}$ 
5:   while not  $\text{done}$  do
6:     /* LM forward pass */
7:      $H_0 \leftarrow \text{EmbeddingLayer}(o_t, g, \text{DK})$ 
8:     for  $l = 1$  to  $L$  do
9:        $(H_l, M) \leftarrow \text{DecoderBlock}_l(H_{l-1}, M)$   $\triangleright$  with reconstruction of  $R$ , cf. Eq. (10)
10:    end for
11:     $i \leftarrow |H_0| + 1$   $\triangleright$  context length +1
12:     $a_t \leftarrow []$ 
13:    while not  $\text{EOS}(x_{<i})$  do
14:      /* Contrastive decoding */
15:      Obtain  $\mathcal{V}_{\text{head}}$  (top- $\vartheta$  head set w.r.t.  $p^+$ )  $\triangleright$  cf. Eq. (11)
16:      for all  $x \in \mathcal{V}_{\text{head}}$  do
17:        Compute contrastive score  $S(x)$  using  $p^+, p^-$   $\triangleright$  cf. Eq. (12)
18:      end for
19:      Sample  $x_i \sim p_{\text{CP}}(\cdot \mid x_{<i})$   $\triangleright$  cf. Eq. (13)
20:       $a_t \leftarrow [a_t, x_i]$   $\triangleright$  append token  $x_i$ 
21:       $i \leftarrow i + 1$ 
22:    end while
23:     $(o_{t+1}, \text{done}) \leftarrow env.\text{step}(a_t)$ 
24:     $t \leftarrow t + 1$ 
25:  end while
26:  if task success then
27:     $\mathcal{M}^+ \leftarrow \mathcal{M}^+ \cup R^+$ 
28:  else
29:     $\mathcal{M}^- \leftarrow \mathcal{M}^- \cup R^-$ 
30:  end if
31: end for
```

---

## 58 B Evaluation

### 59 B.1 Experiment Setting

60 We evaluate NESYPR on standard embodied benchmarks: three PDDL Gym [2] domains (e.g.,  
61 Minecraft, Rearrangement, GlibRearrangement), VirtualHome [3], and ALFWorld [4]. Each bench-  
62 mark is converted to a continual task setting in which the agent solves a sequence of tasks that require  
63 multi-step planning and continual adaptation. Tasks are provided in symbolic form, consisting of  
64 the current observation, the goal, and domain knowledge that specifies the action set and transition  
65 dynamics. Input examples are shown below.

#### Symbolic Input: problem instance

```
(:observation
  obj_next_to(kitchen_counter:object,dish_soap:object)
  obj_next_to(plate:object,kitchen_counter:object)
  obj_next_to(microwave:object,kitchen_counter:object)
  :
  )
```

66

```

    plugged_out(dishwasher:object)
    off(dishwasher:object)
    clean(dishwasher:object)
  )

  (:goal
    (and
      (closed dishwasher)
      (on dishwasher)
      (obj_ontop dish_soap dishwasher)
      (obj_ontop plate dishwasher)
    )
  )
)

```

67

### Symbolic Input: domain knowledge

```

(:action walk_towards
  :parameters (?char - character ?obj - object)
  :precondition (and
    (not (sitting ?char))
    (not (lying ?char))
  )
  :effect (and
    (next_to ?char ?obj)
    (forall (?far_obj - object)
      (when (not (obj_next_to ?far_obj ?obj)) (not (next_to ?char ?far_obj)))
    )
    (forall (?close_obj - object)
      (when (obj_next_to ?close_obj ?obj) (next_to ?char ?close_obj))
    )
  )
)

:
(:action plug_in
  :parameters (?char - character ?obj - object)
  :precondition (or
    (and
      (next_to ?char ?obj)
      (has_plug ?obj)
      (plugged_out ?obj)
    )
    (and
      (next_to ?char ?obj)
      (has_switch ?obj)
      (plugged_out ?obj)
    )
  )
  :effect (and
    (plugged_in ?obj)
    (not (plugged_out ?obj))
  )
)

(:predicates
  (closed ?obj - object)
  (open ?obj - object)
)

```

68

```
(on ?obj - object)
:  
(surfaces ?obj - object)  
(sittable ?obj - object)  
(licable ?obj - object)  
)
```

### 70 B.1.1 PDDL Gym

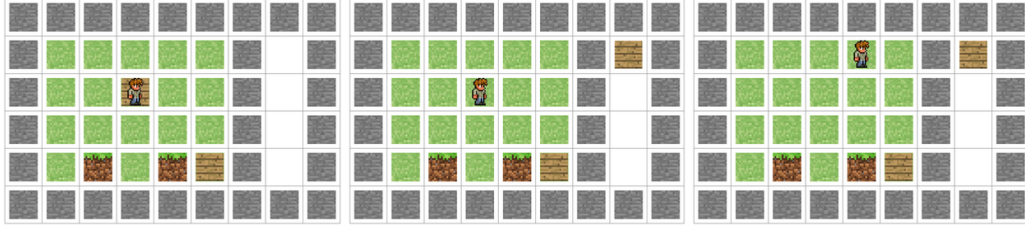
71 PDDL Gym [2] is an embodied task benchmark that converts classical-planning problems expressed  
 72 in PDDL [5] into OpenAI Gym environments [6], providing a unified interface across more than  
 73 twenty relational domains, such as Blocksworld, Sokoban, Minecraft-Crafting, and various object-  
 74 rearrangement tasks. In each environment, observations are represented as sets of first-order predicates,  
 75 and actions are defined by parameterized operator schemas. Solving tasks in such settings requires  
 76 object-centric reasoning, long-horizon planning, and symbol grounding—core capabilities for em-  
 77 bodied agents. Because every problem instance is specified declaratively, it can be exported directly  
 78 to symbolic planners or grounded in low-level simulators, making it easy to compare planning and  
 79 learning methods on exactly the same set of tasks. As a result, PDDL Gym has become a standard  
 80 benchmark for neurosymbolic planning, relational reinforcement learning, and LM-based embodied  
 81 decision making, enabling side-by-side comparisons of symbolic planners, policy learners, and hybrid  
 82 approaches within exactly the same suite of environments.

83 We conduct our experiments on the Minecraft, Rearrangement, and GlibRearrangement domains  
 84 from PDDL Gym. The Minecraft is a domain focused on object manipulation and crafting, where the  
 85 agent must gather raw materials and perform a sequence of actions to synthesize target items. The  
 86 Rearrangement involves moving specific objects to designated locations, requiring spatial reasoning  
 87 and sequential planning in structured environments. The GlibRearrangement extends Rearrangement  
 88 by introducing additional relational constraints, such as requiring the agent to hold certain objects,  
 89 thereby increasing task complexity. While the overall goal remains to reposition specific objects,  
 90 the agent must now account for more intricate action preconditions and inter-object relations during  
 91 planning. These domains in PDDL Gym include 430, 420, and 120 problem instances, respectively,  
 92 most of which can be verified using symbolic planners [1, 7], making them suitable for our evaluation.  
 93 We construct nine distinct task sequences in PDDL Gym for continual tasks by randomly composing  
 94 tasks under a shared evaluation setting. Examples of task and action types for the Minecraft domain  
 95 are summarized in Table 2, with corresponding visualizations shown in Figure 2.

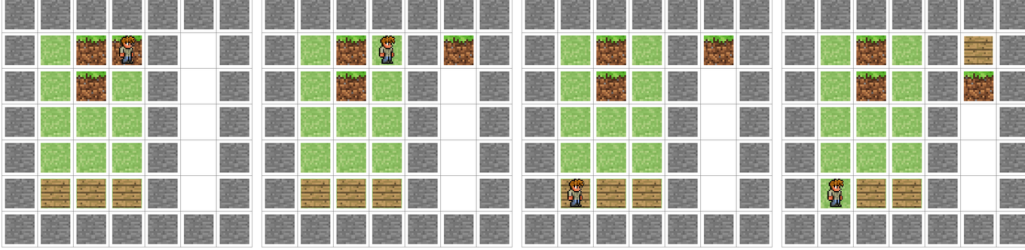
Table 2: Examples of tasks and actions in the Minecraft domain

	Type	Example
Tasks	Move & Equip	(:goal (and (equipped new-2 agent) (equipped grass-0 agent) ))
	Collect & Move	(:goal (and (equipped grass-2 agent) (agentat loc-3-1) ))
	Craft & Equip	(:goal (and (equipped new-2 agent) (isplanks new-2) ))
	Move & Inventory	(:goal (and (inventory new-0) (agentat loc-0-3) ))
	Equip & Inventory	(:goal (and (equipped new-2 agent) (inventory grass-2) ))
	Craft & Inventory	(:goal (and (inventory grass-0) (isplanks new-0) ))
Actions	(recall ?var0 - moveable ?var1 - agent)	recall (grass-0:moveable, agent:agent)
	(move ?var0 - static ?var1 - static)	move (loc-2-0:static, loc-2-1:static)
	(craftPlank ?var0 - moveable ?var1 - agent ?var2 - moveable)	craftplank (new-0:moveable, agent:agent, log-2:moveable)
	(equip ?var0 - moveable ?var1 - agent)	equip (log-2:moveable, agent:agent)
	(pick ?var0 - moveable ?var1 - static)	pick (grass-0:moveable, loc-0-1:static)

96 **Datasets.** We split the full set of problem instances for each domain into separate train and test  
 97 datasets. The train sets consist of 29 instances for the Minecraft, 20 for the Rearrangement, and 40 for  
 98 the GlibRearrangement. The corresponding test sets contain 389, 400, and 80 instances, respectively,  
 99 with no overlap with the training data. For training, we use a relatively small number of problem  
 100 instances, each paired with a reference plan generated by a symbolic planner. For evaluation, we filter  
 101 the test sets to include only solvable instances, which are verified in advance, and do not provide  
 102 reference plans at test time.



(a) Example of “(:goal (and (inventory new-0) (agentat loc-0-3) ))”



(b) Example of “(:goal (and (equipped grass-2 agent) (inventory log-3) ))”

Figure 2: Illustrative examples of tasks in the Minecraft domain

### 103 B.1.2 VirtualHome

104 VirtualHome [8] is an embodied simulation platform designed to replicate real-life household activ-  
 105 ities. In this environment, agents must interact with their surroundings using high-level actions to  
 106 accomplish diverse and complex tasks, such as "Make coffee" or "Wash dishes with the dishwasher",  
 107 making VirtualHome a well-suited benchmark for evaluating the adaptive and structured reasoning  
 108 capabilities of LM-based agents.

109 For our experiments, we build upon the open-source Embodied Agent Interface[3], which provides a  
 110 domain file and problem files for 338 distinct tasks in VirtualHome. We implement the environment by  
 111 simulating the files using TextWorld[9], resulting in VirtualHome environment where the observations  
 112 and actions are represented in PDDL format. To evaluate agents in continual task settings, we  
 113 follow [10], constructing two configurations: behavior-incremental and environment-incremental. In  
 114 each configuration, tasks are grouped into four semantically similar sets based on their respective  
 115 incremental properties. Agents are then evaluated across four distinctly ordered sequences of these  
 116 task sets to assess their robustness to task order variations. These sequences are constructed for both  
 117 seen and unseen settings. In the seen setting, tasks share the same types as those in the training set,  
 118 differing only in object placement. In contrast, the unseen setting consists of entirely novel task types,  
 119 requiring the agent to adaptively reason and generalize based on previously acquired knowledge.  
 120 Details of the tasks and actions are presented in Table 3, and visualizations of various indoor scenes  
 121 are depicted in Figure 3.

122 **Datasets.** We split the full set of problem instances into separate train and test sets, with the test set  
 123 further divided into seen and unseen subsets. The train set contains 77 instances, each paired with  
 124 an optimal plan generated by a symbolic planner [1]. The seen set includes 112 instances that share  
 125 the same goals as those in the train set but differ in object placement and inter-object relations. The  
 126 unseen subset consists of 52 entirely novel tasks that do not appear in either the train or seen sets. At  
 127 test time, agents are given only the current observation and goal, with no access to symbolic tools or  
 128 reference plans.



Table 3: Examples of tasks and actions in VirtualHome

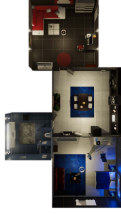
	Type	Example
Tasks	Plug & Switch	(:goal (and (closed cd_player) (plugged_in cd_player) (on cd_player) ))
	Hold	(:goal (and (holds_lh character tooth_paste) (holds_rh character toothbrush) ))
	Sit/Lie	(:goal (and (lying character) (ontop character bed) ))
	Be next to	(:goal (and (next_to character shower) ))
	Object on	(:goal (and (obj_ontop plate table) ))
	Object in	(:goal (and (open freezer) (plugged_in freezer) (obj_inside food_food freezer) ))
Actions	(walk_towards ?char - character ?obj - object)	walk_towards (character:character, computer:object)
	(walk_into ?char - character ?room - object)	walk_into (character:character, bathroom:object)
	(sit ?char - character ?obj - object)	sit (character:character, couch:object)
	(standup ?char - character)	standup (character:character)
	(grab ?char - character ?obj - object)	grab (character:character, plate:object)
	(open ?char - character ?obj - object)	open (character:character, dishwasher:object)
	(close ?char - character ?obj - object)	close (character:character, dishwasher:object)
	(put_on ?char - character ?obj1 - object ?obj2 - object)	put_on (character:character, ground_coffee:object, coffee_maker:object)
	(put_inside ?char - character ?obj1 - object ?obj2 - object)	put_inside (character:character, plate:object, kitchen_cabinet:object)
	(switch_on ?char - character ?obj - object)	switch_on (character:character, computer:object)
	(turn_to ?char - character ?obj - object)	turn_to (character:character, toilet:object)
	(lie ?char - character ?obj - object)	lie (character:character, bed:object)
	(plug_in ?char - character ?obj - object)	plug_in (character:character, dishwasher:object)



(a) Example of scene 0



(b) Example of scene 1



(c) Example of scene 2



(d) Example of scene 3



Figure 3: Illustrative examples of scenes in VirtualHome

### 129 B.1.3 ALFWorld

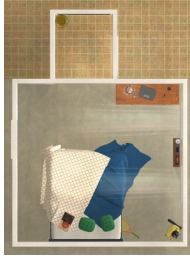
130 ALFWorld[4] is TextWorld-based embodied benchmark built upon the tasks and interaction capa-  
 131 bilities of ALFRED benchmark[11]. This household environment is designed to train and evaluate  
 132 agents on action planning under long and compositional task instructions, making it a challenging  
 133 benchmark for embodied agents.

134 In ALFWorld, a total of 3,554 task instances are available, each provided with its respective problem  
 135 file. To enable evaluation on ALFWorld under symbolic settings, we modify the environment to  
 136 provide observations and accept actions in PDDL format, similar to our setup in VirtualHome.  
 137 All other aspects of the evaluation framework remain consistent with those used in VirtualHome  
 138 experiments, including the presence of behavior- and environment-incremental configurations, as  
 139 well as the division of test set into seen and unseen sets. Details of the tasks and actions are presented  
 140 in Table 4.

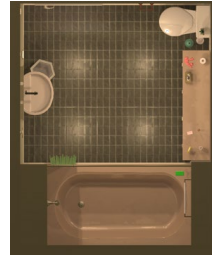
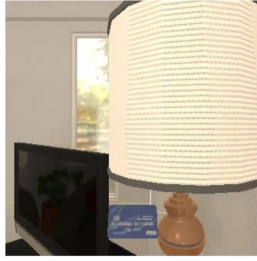
141 **Datasets.** Similar to VirtualHome, we partition the full set of problem instances into separate train  
 142 and test sets, with the test set further divided into seen and unseen sets. The train set comprises 549  
 143 instances, each paired with an optimal plan generated by a symbolic planner [1]. The seen set includes  
 144 1,509 instances that share goals with the train set but vary in object placement and inter-object  
 145 relations. The unseen set contains 1,369 entirely novel tasks not present in either the train or seen

Table 4: Examples of tasks and actions in ALFWorld

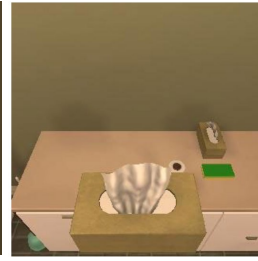
Type	Example
Heat	(:goal (and (exists (?r - receptacle) (exists (?o - object) (and (heatable ?o) (objecttype ?o eggtype) (receptacletype ?r garbagecans) (ishot ?o) (inreceptacle ?o ?r) )))))
Cool	(:goal (and (exists (?r - receptacle) (exists (?o - object) (and (coolable ?o) (objecttype ?o lettuce) (receptacletype ?r countertop) (iscold ?o) (inreceptacle ?o ?r) )))))
Clean	(:goal (and (exists (?r - receptacle) (exists (?o - object) (and (cleanable ?o) (objecttype ?o spatula) (receptacletype ?r diningtable) (isclean ?o) (inreceptacle ?o ?r) )))))
Pick & Place	(:goal (and (exists (?r - receptacle) (exists (?o - object) (and (inreceptacle ?o ?r) (objecttype ?o soapbottle) (receptacletype ?r countertop) )))))
Pick2 & Place	(:goal (and (exists (?r - receptacle) (exists (?o1 - object) (and (objecttype ?o1 vase) (receptacletype ?r desk) (inreceptacle ?o1 ?r) (exists (?o2 - object) (and (not (= ?o1 ?o2)) (objecttype ?o2 vase) (receptacletype ?r desk) (inreceptacle ?o2 ?r) ))))))
Examine	(:goal (and (exists (?ot - object ?r - receptacle ?a - agent ?l - location) (and (objecttype ?ot desk) (toggleable ?ot) (istoggled ?ot) (receptacleatlocation ?r ?l) (atlocation ?a ?l) (inreceptacle ?ot ?r))) (exists (?o - object ?a - agent) (and (objecttype ?o book) (holds ?a ?o) ))))
Actions	<p>(gotolocation ?a - agent ?lstart - location ?lend - location ?r - receptacle)</p> <p>(pickupobject ?a - agent ?l - location ?o - object ?r - receptacle)</p> <p>(openobject ?a - agent ?l - location ?r - receptacle)</p> <p>(heatobject ?a - agent ?l - location ?r - receptacle ?o - object)</p> <p>(coolobject ?a - agent ?l - location ?r - receptacle ?o - object)</p> <p>(cleanobject ?a - agent ?l - location ?r - receptacle ?o - object)</p> <p>(toggleobject ?a - agent ?l - location ?o - object ?r - receptacle)</p>



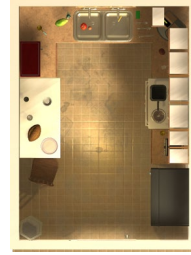
(a) Example of bedroom type scene



(b) Example of bathroom type scene



(c) Example of livingroom type scene



(d) Example of kitchen type scene



Figure 4: Illustrative examples of scenes ALFWorld

146 sets. At test time, agents receive only the current observation and goal, without access to symbolic  
 147 tools or reference plans.

#### 148 B.1.4 Baselines

149 For comparison, we organize the baselines into four categories:

- 150 • Single-step planning approaches generate the entire action sequence in a single, unified inference  
151 step, without intermediate reasoning or iterative refinement.
  - 152 – ZSP [12] leverages pretrained LMs to translate high-level natural language instructions into  
153 executable action sequences for embodied agents in a zero-shot manner.
  - 154 – RAP [13] enables LM-based agents to improve decision-making by retrieving and leveraging  
155 past experiences stored in contextual memory.
  - 156 – LLM-Planner [14] leverages the few-shot in-context learning capabilities of LMs to aid action  
157 planning. By retrieving demonstrations relevant to the current observation throughout the episode,  
158 it enables more grounded and context-aware planning.
- 159 • Agentic workflows perform multi-step reasoning through multiple LM calls, allowing for self-  
160 refinement and iterative decision-making.
  - 161 – CoT [15] enables LMs to solve complex reasoning tasks by generating intermediate reasoning  
162 steps, leading to improved performance on arithmetic, commonsense, and symbolic reasoning  
163 benchmarks.
  - 164 – ToT [16] empowers LMs to perform deliberate problem-solving by exploring and evaluating  
165 multiple reasoning paths, significantly improving performance on complex tasks.
  - 166 – GoT [17] enhances LMs reasoning by structuring intermediate thoughts as a graph, allowing for  
167 advanced operations like feedback loops and aggregation, leading to improved performance on  
168 complex tasks.
  - 169 – ReAct [18] interleaves reasoning steps between the interactions with the environment, enabling  
170 more adaptive decision-making based on rationales generated from interpreting the current  
171 observation.
  - 172 – Reflexion [19] incorporates verbal feedback from failed experiences to guide subsequent decision-  
173 making, enabling the model to learn from experiences without parameter updates.
- 174 • Memory-augmented LM
  - 175 – LongMem [20] enhances LMs by incorporating a decoupled memory module, enabling the  
176 models to store and retrieve extended context information, thereby improving performance on  
177 tasks requiring long-term dependencies.
  - 178 – LMM [21] augments the Transformer [22] architecture with an auxiliary memory module to  
179 improve multi-step reasoning and long-term context modeling.
  - 180 – Optimus-2 [23] introduces a memory-augmented agent for embodied task planning in open-world  
181 environment, enabling long-horizon action generation by conditioning on goal, observation, and  
182 action history.
  - 183 – DT-Mem [24] enhances generalization and adaptability of reinforcement learning agents by  
184 storing and reusing knowledge across tasks through an internal working memory module.
  - 185 – BUTLER [25] fine-tunes a pretrained LMs using a small number of expert demonstrations and  
186 limited environment interactions to enable effective action planning in text-based environments.  
187 Although it does not incorporate memory mechanisms, it serves as a learning-based baseline for  
188 evaluating memory-augmented LM-based embodied agents.
- 189 • Proceduralization
  - 190 – BoT [26] stores abstract thought templates in a meta-buffer and dynamically retrieves and  
191 instantiates them to guide procedural LM reasoning.
  - 192 – LRM [27] internalizes CoT reasoning through reinforcement learning, and also offers compact  
193 student variants distilled from larger teacher models.
  - 194 – PlaSma [28] equips smaller LMs with procedural knowledge and planning capabilities by  
195 distilling knowledge from larger LMs and employing a verifier-guided decoding algorithm,  
196 enabling them to achieve performance comparable to their larger counterparts.

197 **ZSP.** This work investigates whether a pretrained LMs can convert high-level natural language  
198 instructions into executable action sequences without additional training. The entire action plan is  
199 generated in a single inference step and remains fixed, without incorporating feedback from the  
200 environment or adapting to changes in state.

We refer to the publicly available implementation <sup>1</sup>. We follow the default prompt templates described in the paper and codes, with minor modifications to adapt to the observation-goal input format in our benchmark environments. For the planning LM, we use LLaMA3.2-1B [29] with a temperature of 0.0 and a maximum output length of 128 tokens for PDDL Gym. For the translation LM, we use sentence embedding model (i.e., all-mpnet-base-v2) from Sentence-Former [30] to retrieve the top- $k$  most similar instructions along with their associated symbolic actions. As shown in Table 6, input prompts include five in-context examples sampled from the training set. Generation is performed using deterministic decoding with top- $p = 1.0$ .

Table 5: Hyperparameter settings for ZSP

Hyperparameters	Value
In-context samples	5
Decoding	Greedy
Maximum new tokens	128

**RAP.** RAP addresses the challenge of enabling LM agents to utilize past experiences in current decision-making processes, a behavior innate to humans. The framework stores past experiences in a memory module and retrieves relevant information based on the similarity to the current context. This retrieved information is then used to inform the agent’s planning through in-context learning.

Here, we extend ZSP by incorporating the contextual memory concept from RAP. Specifically, we enable past successful experiences to be dynamically composed as in-context examples based on the current task context. This allows the agent to adaptively retrieve and organize relevant demonstrations as the memory grows. We build on the publicly available implementation of RAP <sup>2</sup>. All other components, including planning and retrieval, remain consistent with ZSP.

**LLM-Planner.** LLM-Planner augments ZSP with few-shot demonstrations, leveraging the in-context learning capabilities of LMs to better ground the agent in the environment. It also incorporates a replanning mechanism that enables the agent to recover from infeasible plans.

We implement referring to the official implementation <sup>3</sup>. At each timestep, the top- $k$  most relevant demonstrations are dynamically retrieved from train set based on the current observation and goal. Candidates are first filtered using cosine similarity with the goal, followed by selection based on Jaccard similarity over current observation predicates. For replanning, LLM-Planner is implemented to reset the action history when three consecutive non-executable actions are generated. The hyperparameter settings for LLM-Planner are provided in Table 6. By default, we use Qwen2.5-0.5B [31] for both VirtualHome and ALFWorld.

Table 6: Hyperparameter settings for LLM-Planner

Hyperparameters	Value
In-context samples	3
Decoding	Greedy
Maximum new tokens	120

**CoT.** CoT prompting improves the reasoning ability of LMs by presenting step-by-step exemplars that guide the model through intermediate reasoning steps, leading to better performance on complex, multi-step tasks.

Our implementation of CoT is used as a baseline for PDDL Gym, built upon the RAP framework and extended with additional rationales to guide symbolic reasoning. We base our implementation on the publicly available GoT repository <sup>4</sup> which includes CoT as part of its baseline methods. Specifically, we augment the train set with step-by-step rationales that reflect key features of the underlying MDP structure. The rationale design follows the approach proposed in [32], using the problem instances

<sup>1</sup><https://github.com/huangw118/language-planner>

<sup>2</sup><https://github.com/PanasonicConnect/rap>

<sup>3</sup><https://github.com/OSU-NLP-Group/LLM-Planner>

<sup>4</sup><https://github.com/spcl/graph-of-thoughts>

236 and corresponding plans in the train set to provide structured, interpretable reasoning traces. All other  
237 hyperparameters follow the same configuration as used in ZSP and RAP.

238 **ToT.** ToT prompting extends the chain-of-thought (CoT) framework by enabling exploration of  
239 multiple reasoning paths instead of committing to a single linear sequence. At each step, the model  
240 generates and evaluates multiple candidate thoughts, forming a tree structure that allows for backtrack-  
241 ing and selection of the most promising reasoning trajectory. This approach improves performance on  
242 tasks that benefit from deliberation, such as mathematical reasoning and complex decision making.

243 Our implementation of ToT builds on the same additionally designed rationales used in CoT, and  
244 leverages the ToT implementation provided in the GoT repository. To enable tree-structured search,  
245 we adapt the branching, self-evaluation, and selection prompts to align with the characteristics of  
246 PDDL Gym tasks, setting the maximum branch factor to 5. All other hyperparameters follow the same  
247 configuration as used in ZSP and RAP.

248 **GoT.** GoT generalizes the tree-based reasoning structure of ToT by allowing thoughts to be organized  
249 as arbitrary graphs rather than strict trees. This enables the model to support richer reasoning patterns  
250 such as merging, revisiting, and cross-referencing intermediate thoughts. Unlike CoT and ToT, which  
251 follow fixed linear or hierarchical paths, GoT allows for flexible traversal and aggregation of reasoning  
252 substructures, making it well-suited for tasks that require integration of multiple partial solutions or  
253 iterative refinement.

254 Our implementation of GoT is based on the official repository, with additional modifications to the  
255 aggregation and refinement prompts to better suit the structure of PDDL Gym tasks. The branch factor  
256 is set to a maximum of 5. All other hyperparameters follow the same configuration as used in ZSP  
257 and RAP.

258 **ReAct.** ReAct interleaves reasoning steps between observation inputs and decision-making, enabling  
259 the agent to adaptively solve tasks in the environment. The generated reasoning may involve goal  
260 analysis or interpretation of the current observation, thereby aiding the production of appropriate  
261 actions.

262 We refer to official repository <sup>5</sup> for implementation. The few-shot demonstrations annotated with  
263 reasoning steps are provided as in-context examples to guide the LM’s behavior. These annotations  
264 are generated by prompting the LLaMa3.1-8B model to produce rationales based on the current  
265 observation and goal. Demonstration retrieval follows the same procedure as in LLM-Planner. During  
266 inference, the model generates the reasoning step first, followed by the corresponding action. ReAct  
267 adopts the same hyperparameter settings as LLM-Planner, as shown in Table 6. By default, we use  
268 Qwen2.5-0.5B for both VirtualHome and ALFWorld.

269 **Reflexion.** Reflexion is an LM-based agent, similar to ReAct, that alternates between reasoning and  
270 acting. It additionally introduces a self-reflection mechanism, generating feedback by analyzing failed  
271 trajectories and using this feedback to better guide subsequent attempts.

272 The original framework incorporates extracted feedback by appending it into the prompt, encouraging  
273 the LM to consider it during generation. In our setting, we incorporate feedback by adding the failed  
274 action to the "bad words" list, thereby directly preventing the model from generating that action in  
275 subsequent attempts. Reflexion adopts the same hyperparameter settings as LLM-Planner, as shown  
276 in Table 4. By default, we use Qwen2.5-0.5B for both VirtualHome and ALFWorld.

277 **LongMem.** LongMem is a composite model consisting of a frozen LM backbone and a trainable  
278 memory retrieval network, Residual Sidenet. Once trained, it automatically incorporates previous  
279 information via the key-value cache of earlier chunks into the current generation, enabling LM to  
280 effectively memorize and model long-term dependencies.

281 To adapt LongMem for benchmarks such as PDDL Gym, VirtualHome, and ALFWorld, we apply  
282 several modifications and hyperparameter adjustments based on the official implementation <sup>6</sup>. The  
283 memory module is configured to store key-value pairs for up to 400 problem instances, with each  
284 input sequence stored in 64-token chunks. During retrieval, the model selects the top 6 most relevant  
285 chunks based on similarity to the current query. By default, we use LLaMA3.2-1B for PDDL Gym  
286 and Qwen2.5-0.5B for VirtualHome and ALFWorld.

---

<sup>5</sup><https://github.com/ysymyth/ReAct>

<sup>6</sup><https://github.com/Victorwz/LongMem>

**LMM.** LMM augments a conventional decoder-only transformer with an auxiliary memory flow, enabling it to maintain long-term dependencies throughout generation. A dedicated memory module allows for preserving important information by dynamically interacting with input embeddings, while the memory itself continuously updating through a gating mechanism.

Based on the official implementation <sup>7</sup>, we make slight adaptations to use LMM as an LM-based agent. The memory capacity and slot size are set to match the hidden size of the backbone LM. Additionally, both the number of attention heads and the hidden size used in the memory-augmented attention mechanism follow the same configuration as the backbone model. By default, we use LLaMA3.2-1B for PDDL Gym and Qwen2.5-0.5B for VirtualHome and ALFWorld.

**Optimus-2.** Optimus-2 is an embodied agent model designed to tackle various open-world tasks, with the focus on Minecraft. To effectively model the complex relationships among observations, actions, and language, Optimus-2 proposes action-guided behavior encoder, which integrates these three elements along with behavior history memory to produce behavior tokens.

Optimus-2 is used as a closed-loop task planning baseline for VirtualHome and ALFWorld. Since the official code is not available on the GitHub repository <sup>8</sup>, we implemented it based on the descriptions provided in the original paper. We use Qwen2.5-0.5B as the backbone LM. The memory module is configured with a maximum history length of 1,000, and both the attention dimensionality and number of attention heads in the memory mechanism are set to match those of the backbone LM.

**DT-Mem.** DT-Mem addresses the forgetting problem in Decision Transformers, which arises from storing task-specific skills in implicit memory. Inspired by human working memory, it introduces a distributed memory architecture that allows for explicit storage and retrieval of multiple skills, effectively mitigating forgetting.

For the VirtualHome and ALFWorld baselines, we employ Qwen2.5-0.5B as the backbone LM, providing it with a concatenation of the current observation, the goal description, and domain knowledge. Only the memory module is borrowed from the public DT-Mem implementation <sup>9</sup>, configured with 128 memory slots whose dimensionality equals the model’s hidden size.

**BUTLER.** BUTLER is a LM-based agent that generates high-level textual actions based on current observations and a given goal. Built upon a pre-trained LM, it is fine-tuned with a small set of expert demonstrations to better adapt to the target environment. BUTLER serves as the baseline for other models that also involve fine-tuning on demonstrations.

We adopt a LoRA-based fine-tuning approach [33] for training. For the PDDL Gym baseline, we use LLaMA3.2-1B as the backbone LM.

**BoT.** Buffer of Thoughts (BoT) utilizes thought templates, which are guidelines for high-level reasoning, and uses them as scaffold for reasoning. This improves both the efficiency and accuracy of the LM’s reasoning process.

BoT begins by distilling key information from the problem into a well-structured list. It then retrieves the most relevant thought template from the meta-buffer and instantiates it with the key information to perform reasoning. Once reasoning is complete, the instantiated reasoning is distilled back into a new thought template and added to the meta-buffer. For implementation, we refer to official code repository <sup>10</sup>. We use GPT-4.1, LLaMa3.1-8B, and LLaMa3.1-70B as base LMs. For problem distillation, however, we use GPT-4.1 across all models to ensure better performance. The hyperparameter settings for BoT are provided in Table 7.

Table 7: Hyperparameter settings for BoT

Hyperparameters	Value
Retrieval Threshold	0.6
Maximum new tokens	3,000

<sup>7</sup><https://github.com/convergence-ai/lm2>

<sup>8</sup><https://github.com/JiuTian-VL/Optimus-2>

<sup>9</sup>[https://github.com/luciferkonn/DT\\_Mem](https://github.com/luciferkonn/DT_Mem)

<sup>10</sup><https://github.com/YangLing0818/buffer-of-thought-llm>

329 **LRM.** LRM is a class of LMs in which the ability to generate reasoning chains is internalized within  
 330 the model parameters, enabling more adaptive and diverse reasoning than the in-context learning  
 331 approach such as CoT.

332 These models are often trained via reinforcement learning, as in DeepSeek-R1, and can also serve as  
 333 teacher models for training smaller variants through distillation. In our work, we use two distilled  
 334 models—DeepSeek-R1-Distill-LLaMa-8B and 70B—as well as one proprietary model, o3-mini, for  
 335 proceduralization baselines. The hyperparameter settings for the distilled models are provided in  
 336 Table 8, while default arguments are used for o3-mini.

Table 8: Hyperparameter settings for LRM

Hyperparameters	Value
Temperature	0.6
Maximum new tokens	3,000

337 **PlaSma.** PlaSma distills procedural knowledge from a large LM into a small LM by training it to  
 338 generate both standard, and counterfactual plans under given constraints. To further enhance planning  
 339 quality, a step-wise verifier is utilized during decoding process, guiding the small LM to produce  
 340 more coherent plans.

341 As a proceduralization baseline, we employ GPT-4o [34] as the teacher LLM to synthesize both  
 342 positive augmented problem instances and counterfactual variants. We fine-tune a RoBERTa-Large  
 343 verifier [35] on the synthesized samples and embed it in a verifier-guided, step-wise beam-search  
 344 decoder implemented with the public PlaSma code <sup>11</sup>. As backbone models, we evaluate LLaMa3.2-  
 345 1B, LLaMa3.2-3B, and LLaMa3.1-8B.

#### 346 B.1.5 Metrics

347 We evaluate performance using four standard metrics, following [12, 36, 37].

- 348 • **Cumulative Task Success Rate (CSR)** measures the proportion of tasks in which all required  
 349 sub-goals are successfully completed, indicating overall task-level performance.
- 350 • **Cumulative Goal-Conditioned Success Rate (CGC)** computes the fraction of individual sub-goals  
 351 achieved across all tasks, capturing the agent’s partial progress even when full task completion is  
 352 not attained.
- 353 • **Executability (Exe)** evaluates whether each action selected by the agent is executable within the  
 354 environment, reflecting the syntactic and semantic correctness of generated actions.
- 355 • **Success rate weighted by Path Length (SPL)** accounts for both task success and the efficiency of  
 356 the action sequence, rewarding shorter and more optimal plans that still achieve the goal.

---

<sup>11</sup><https://github.com/allenai/PlaSma>

Table 9: Detailed performance on open-loop continual embodied tasks across each PDDL Gym domain. These results correspond to Table 1 in the main paper (Section 4.2).

METHOD	PARAMS	TRAIN				TEST			
		CSR (↑)	CGC (↑)	EXE (↑)	SPL (↑)	CSR (↑)	CGC (↑)	EXE (↑)	SPL (↑)
DOMAIN: MINECRAFT									
ZSP	1.7B (0.0%)	76.4±5.5	81.7±4.8	100.0±0.0	0.8±0.1	16.4±1.5	35.9±2.0	100.0±0.1	0.1±0.0
RAP	1.7B (0.0%)	76.4±5.5	81.7±4.8	100.0±0.0	0.8±0.1	16.8±0.9	18.0±1.9	100.0±0.1	0.1±0.0
CoT	1.7B (0.0%)	83.3±6.7	83.3±9.3	100.0±0.0	0.8±0.1	17.0±0.5	25.7±1.8	100.0±0.0	0.1±0.0
ToT	1.7B (0.0%)	85.1±4.2	85.7±3.7	100.0±0.0	0.9±0.0	18.7±1.0	32.2±2.6	100.0±0.0	0.1±0.0
GoT	1.7B (0.0%)	89.1±5.1	94.6±7.7	100.0±0.0	0.9±0.1	18.9±0.5	25.9±1.2	100.0±0.0	0.1±0.0
BUTLER	1.2B (0.6%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	51.4±1.9	56.7±2.6	99.7±0.3	0.4±0.0
LONGMEM	1.6B (24.6%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	53.3±3.7	56.3±4.1	99.8±0.1	0.5±0.0
LMM	1.3B (6.3%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	47.9±8.4	56.5±4.6	99.9±0.1	0.4±0.0
NeSyPr	1.3B (6.3%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	65.2±1.4	68.9±2.4	100.0±0.1	0.6±0.0
DOMAIN: REARRANGEMENT									
ZSP	1.7B (0.0%)	94.2±3.8	97.6±1.2	100.0±0.0	0.9±0.0	15.3±2.1	22.4±3.4	100.0±0.0	0.1±0.0
RAP	1.7B (0.0%)	94.2±3.8	97.6±1.2	100.0±0.0	0.9±0.0	18.3±1.7	29.0±2.6	100.0±0.0	0.1±0.0
CoT	1.7B (0.0%)	95.0±4.5	98.5±2.5	100.0±0.0	1.0±0.0	19.2±0.9	29.9±1.5	100.0±0.0	0.1±0.0
ToT	1.7B (0.0%)	95.0±6.3	99.5±1.2	100.0±0.0	1.0±0.1	20.2±1.2	33.9±1.4	100.0±0.0	0.2±0.0
GoT	1.7B (0.0%)	95.8±4.9	100.0±0.0	100.0±0.0	1.0±0.0	23.0±1.5	37.1±1.4	100.0±0.0	0.2±0.0
BUTLER	1.2B (0.6%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	56.5±2.0	69.8±2.9	100.0±0.0	0.5±0.0
LONGMEM	1.6B (24.6%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	58.2±1.9	69.9±1.4	100.0±0.0	0.5±0.0
LMM	1.3B (6.3%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	57.4±4.0	69.6±8.9	100.0±0.0	0.5±0.1
NeSyPr	1.3B (6.3%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	73.5±3.0	80.8±1.0	100.0±0.0	0.7±0.0
DOMAIN: GLIBREARRANGEMENT									
ZSP	1.7B (0.0%)	68.0±2.9	75.1±2.9	100.0±0.0	0.7±0.0	14.0±1.4	27.9±7.3	100.0±0.0	0.1±0.0
RAP	1.7B (0.0%)	77.8±2.9	89.2±3.8	100.0±0.0	0.8±0.0	16.8±1.6	21.0±2.6	100.0±0.0	0.1±0.0
CoT	1.7B (0.0%)	80.9±5.0	89.5±3.9	100.0±0.0	0.8±0.1	17.8±1.6	21.4±1.6	100.0±0.0	0.1±0.0
ToT	1.7B (0.0%)	93.9±2.6	98.8±1.5	100.0±0.0	0.9±0.0	19.7±0.5	28.2±1.3	100.0±0.0	0.1±0.0
GoT	1.7B (0.0%)	95.1±1.5	95.1±2.7	100.0±0.0	1.0±0.0	20.5±1.4	28.4±1.9	100.0±0.0	0.2±0.0
BUTLER	1.2B (0.6%)	99.2±1.3	99.7±0.5	100.0±0.0	1.0±0.0	46.5±2.9	57.2±3.2	100.0±0.0	0.3±0.0
LONGMEM	1.6B (24.6%)	99.6±1.0	99.8±0.4	100.0±0.0	1.0±0.0	47.7±5.3	53.6±3.4	100.0±0.0	0.3±0.0
LMM	1.3B (6.3%)	99.6±1.0	99.8±0.4	100.0±0.0	1.0±0.0	48.3±1.9	60.3±0.0	100.0±0.0	0.3±0.0
NeSyPr	1.3B (6.3%)	100.0±0.0	100.0±0.0	100.0±0.0	1.0±0.0	58.7±1.8	72.4±1.2	100.0±0.0	0.5±0.0

## B.2 Additional Experimental Results

### B.2.1 Open-loop Continual Embodied Tasks

To evaluate the generalization performance of NESyPR on open-loop continual task planning, we conduct experiments in Table 9 using multiple domains from PDDL Gym. In this setting, the agent generates a complete action sequence without intermediate observations and receives only binary task feedback (success or failure) before proceeding to the next task. NESyPR consistently outperforms the strongest baseline in each domain, LongMem in Minecraft and Rearrangement, and LMM in Glibrearrangement. It achieves average improvements of 12.5% in CSR, 11.9% in CGC, and 0.16 in SPL, demonstrating superior structured and adaptive reasoning capabilities. The single-step planning baselines such as ZSP and RAP, which rely on in-context retrieval-augmented generation [38], show limited reasoning capacity on unseen tasks. The agentic workflow baselines such as CoT, ToT, and GoT, which use reasoning-guidance prompts crafted from the train set [32], perform slightly better, but remain far from achieving reliable task success. The memory-augmented LMs such as LongMem and LMM outperform the fine-tuning baseline BUTLER, yet still show 13.2% lower CSR on average compared to NESyPR.



Table 10: Detailed performance on closed-loop continual embodied tasks in VirtualHome and ALFWorld. These results correspond to Table 2 in the main paper (Section 4.2).

(a) Performance across different evaluation categories in VirtualHome									
METHOD	TRAIN			SEEN			UNSEEN		
	CSR (↑)	CGC (↑)	SPL (↑)	CSR (↑)	CGC (↑)	SPL (↑)	CSR (↑)	CGC (↑)	SPL (↑)
<b>CONFIGURATION: BEHAVIOR-INCREMENTAL</b>									
LLM-PLANNER	61.0±3.5	66.4±2.9	0.4±0.0	45.5±1.9	47.4±2.0	0.3±0.0	28.8±2.2	30.0±2.1	0.2±0.0
REACT	63.6±1.1	70.0±0.6	0.4±0.0	54.7±1.3	56.4±0.8	0.4±0.0	32.7±3.5	34.3±3.6	0.3±0.0
REFLEXION	60.8±5.9	68.8±5.5	0.4±0.0	57.2±3.8	61.6±5.4	0.4±0.0	33.7±2.5	35.3±2.1	0.3±0.0
LONGMEM	80.5±2.9	86.2±2.5	0.8±0.0	63.3±5.6	68.3±6.3	0.6±0.1	45.7±7.3	52.2±8.9	0.4±0.1
LMM	80.2±4.2	85.2±2.9	0.8±0.0	53.6±5.9	57.6±5.2	0.5±0.1	38.8±4.4	43.3±4.4	0.3±0.0
DT-MEM	77.3±3.8	80.8±4.8	0.7±0.0	69.3±5.7	71.9±5.9	0.7±0.1	48.7±5.9	52.6±6.0	0.5±0.1
OPTIMUS-2	79.3±5.4	83.9±4.0	0.8±0.1	70.4±4.4	74.0±3.4	0.7±0.1	44.0±6.1	50.9±7.3	0.4±0.1
NEsyPR	<b>89.8±1.9</b>	<b>92.3±1.1</b>	<b>0.9±0.0</b>	<b>78.9±4.5</b>	<b>81.7±2.5</b>	<b>0.8±0.0</b>	<b>61.1±2.2</b>	<b>69.3±2.6</b>	<b>0.6±0.0</b>
<b>CONFIGURATION: ENVIRONMENT-INCREMENTAL</b>									
LLM-PLANNER	61.7±7.5	66.5±6.8	0.4±0.0	45.5±1.9	47.7±2.6	0.3±0.0	32.7±2.7	33.7±2.9	0.2±0.0
REACT	64.3±3.4	69.9±2.5	0.4±0.0	52.2±2.7	55.6±2.5	0.4±0.0	35.1±1.0	37.1±1.6	0.3±0.0
REFLEXION	66.2±4.4	72.1±3.6	0.5±0.0	59.6±4.1	62.4±3.3	0.5±0.0	34.1±1.8	35.5±2.4	0.3±0.0
LONGMEM	80.2±3.2	86.2±2.4	0.8±0.0	68.9±5.9	72.9±6.3	0.7±0.1	45.2±3.3	51.8±4.0	0.4±0.0
LMM	80.2±4.2	85.2±2.9	0.8±0.0	53.6±5.9	57.7±5.2	0.5±0.1	38.8±4.4	43.3±4.4	0.3±0.0
DT-MEM	77.5±4.7	81.0±6.1	0.7±0.0	70.1±6.3	72.7±7.4	0.7±0.1	51.6±6.0	56.6±6.9	0.5±0.1
OPTIMUS-2	79.3±5.4	83.8±4.0	0.8±0.1	70.4±4.4	74.0±3.4	0.7±0.0	44.0±6.1	50.9±7.3	0.4±0.1
NEsyPR	<b>90.1±1.0</b>	<b>92.7±0.6</b>	<b>0.9±0.0</b>	<b>77.0±2.5</b>	<b>80.7±2.4</b>	<b>0.8±0.0</b>	<b>58.2±3.0</b>	<b>66.1±3.8</b>	<b>0.6±0.0</b>

(b) Performance across different evaluation categories in ALFWorld									
METHOD	TRAIN			SEEN			UNSEEN		
	CSR (↑)	CGC (↑)	SPL (↑)	CSR (↑)	CGC (↑)	SPL (↑)	CSR (↑)	CGC (↑)	SPL (↑)
<b>CONFIGURATION: BEHAVIOR-INCREMENTAL</b>									
LLM-PLANNER	52.4±2.6	68.1±2.0	0.5±0.0	14.0±0.8	21.7±0.7	0.1±0.0	3.3±0.4	11.7±0.5	0.0±0.0
REACT	46.3±1.5	65.3±1.1	0.4±0.0	12.8±0.5	21.4±0.6	0.1±0.0	2.9±0.2	11.6±0.2	0.0±0.0
REFLEXION	44.3±0.7	64.1±0.7	0.4±0.0	13.0±0.5	21.6±0.8	0.1±0.0	2.9±0.4	11.8±0.4	0.0±0.0
LONGMEM	50.1±3.1	58.6±2.4	0.5±0.0	48.6±0.6	57.3±0.7	0.5±0.0	45.2±0.8	54.5±0.8	0.5±0.0
LMM	63.8±1.5	71.5±1.4	0.6±0.0	42.6±2.4	46.9±2.0	0.4±0.0	38.7±2.1	45.2±2.5	0.4±0.0
DT-MEM	57.7±2.2	61.7±2.7	0.6±0.0	41.3±2.9	48.0±3.3	0.4±0.0	38.0±3.8	44.1±4.2	0.4±0.0
OPTIMUS-2	59.5±1.5	67.4±1.3	0.6±0.0	52.8±0.9	61.6±0.7	0.5±0.0	49.1±0.7	58.7±0.6	0.5±0.0
NEsyPR	<b>69.6±2.7</b>	<b>76.2±2.1</b>	<b>0.7±0.0</b>	<b>61.1±1.1</b>	<b>68.6±1.3</b>	<b>0.6±0.0</b>	<b>59.7±1.4</b>	<b>67.9±1.3</b>	<b>0.6±0.0</b>
<b>CONFIGURATION: ENVIRONMENT-INCREMENTAL</b>									
LLM-PLANNER	39.1±2.0	52.4±0.9	0.3±0.0	8.8±0.4	17.0±0.3	0.1±0.0	2.7±0.1	10.4±0.5	0.0±0.0
REACT	33.0±1.6	49.2±0.7	0.3±0.0	9.2±0.5	18.0±0.5	0.1±0.0	2.6±0.6	10.5±0.8	0.0±0.0
REFLEXION	31.3±0.9	48.2±0.5	0.3±0.0	8.8±0.4	17.3±0.1	0.1±0.0	2.6±0.4	10.5±0.1	0.0±0.0
LONGMEM	40.6±1.1	51.3±0.9	0.4±0.0	39.4±2.4	51.6±1.9	0.4±0.0	32.6±1.4	45.7±0.7	0.3±0.0
LMM	48.0±0.4	52.8±1.4	0.5±0.0	32.0±2.1	41.4±0.5	0.3±0.0	29.0±1.4	39.3±2.8	0.3±0.0
DT-MEM	42.4±1.8	52.9±1.9	0.4±0.0	38.7±11.7	47.8±14.7	0.4±0.1	30.0±8.2	40.5±10.1	0.3±0.1
OPTIMUS-2	44.6±0.7	56.9±0.5	0.4±0.0	42.7±0.9	55.2±1.2	0.4±0.0	33.1±0.2	46.9±0.7	0.3±0.0
NEsyPR	<b>52.2±0.9</b>	<b>62.4±0.9</b>	<b>0.5±0.0</b>	<b>51.1±2.1</b>	<b>63.5±1.8</b>	<b>0.5±0.0</b>	<b>41.1±0.7</b>	<b>55.3±1.0</b>	<b>0.4±0.0</b>

## B.2.2 Closed-loop Continual Embodied Tasks

To further evaluate the generalization performance of NESYPR alongside its adaptability in dynamic settings, we conduct experiments under a closed-loop continual task planning setup across VirtualHome and ALFWorld. The test sets are explicitly divided into seen and unseen sets, enabling a detailed assessment of the agent’s structured and adaptive reasoning capabilities, as shown in Table 10. Unlike open-loop settings, the agent selects actions sequentially in response to intermediate observations. In VirtualHome experiment (Table 10(a)), NESYPR similarly outperforms the strongest baseline, DT-Mem. It shows average improvements of 12.6% in CSR and 11.6% in CGC on the train set, 8.3% and 8.9% on the seen set, and 9.5% and 13.1% on the unseen set, respectively. In ALFWorld experiment (Table 10(b)), NESYPR outperforms the strongest baseline, Optimus-2, across all incremental settings. It achieves average gains of 8.9% in CSR and 7.2% in CGC on the train set, 8.4% and 7.7% on the seen set, and 9.3% and 8.8% on the unseen set, respectively. Across both benchmarks, the performance gains on the unseen set are consistently greater than those on the seen sets. Combined with an average improvement of 0.12 in SPL, these results indicate that NESYPR performs effective symbolic reasoning. Specifically, both LLM-Planner and the agentic workflows such as ReAct and Reflexion show limited capabilities for symbolic reasoning across all evaluation scenarios. While Reflexion leverages past experiences via feedback, it appears to lack the robust

reasoning capabilities required in dynamic and complex tasks. Memory-augmented models tailored for embodied agents, such as DT-Mem and Optimus-2, outperform general-purpose variants like LongMem and LMM. However, NESYPR achieves higher performance, surpassing both DT-Mem and Optimus-2 by an average of 11.2% in CSR and 11.6% in CGC, underscoring the effectiveness of neurosymbolic proceduralization.

Table 11: Detailed analysis of proceduralization, corresponding to Table 3 in the main paper (Section 4.3). LATENCY denotes the agent’s plan generation time in seconds. IN/OUT TOKENS refer to the number of input and generated tokens, respectively.

METHOD	LM	TASK PERFORMANCE			REASONING LOAD			FLOPs
		CSR (↑)	CGC (↑)	SPL (↑)	LATENCY (↓)	IN TOKENS (↓)	OUT TOKENS (↓)	
BoT	LLAMA3.1-8B	53.0±0.5	63.5±0.4	0.3±0.0	59.5±1.9	8007.9±103.9	1315.4±28.1	86.8+ $\alpha$ TFLOPs
	LLAMA3.1-70B	81.9±0.4	85.1±0.3	0.6±0.0	75.1±3.8	7651.0±127.7	794.1±33.4	-
	GPT4.1	92.1±0.3	93.6±0.2	0.7±0.0	22.2±2.8	7986.1±144.2	1202.2±197.2	-
LRM	DEEPSEEK-R1-8B	11.5±0.3	15.6±0.3	0.1±0.0	111.0±3.3	3198.5±15.8	2187.6±69.0	55.4 TFLOPs
	DEEPSEEK-R1-70B	26.5±0.4	27.5±0.4	0.2±0.0	209.4±9.2	3198.5±15.8	1679.3±87.5	-
	O3-MINI	78.9±0.4	80.8±0.4	0.5±0.0	18.6±1.7	3214.6±15.7	2113.9±63.2	-
PLASMA	LLAMA3.2-1B	67.4±0.5	71.9±0.4	0.7±0.0	2.7±0.5	3221.8±45.3	32.7±4.6	-
	LLAMA3.2-3B	70.7±0.4	75.7±0.3	0.7±0.0	7.2±0.7	3247.7±17.2	29.5±5.5	-
	LLAMA3.1-8B	80.5±0.5	89.2±2.3	0.8±0.0	18.4±5.8	3371.0±13.5	122.4±41.6	236.9 TFLOPs
NESYPR	LLAMA3.2-1B	73.2±0.4	76.0±0.4	0.7±0.0	1.2±0.3	3168.5±0.0	30.1±5.3	-
	LLAMA3.2-3B	83.6±2.0	88.8±2.0	0.8±0.0	3.5±0.3	3169.5±0.0	43.6±5.3	-
	LLAMA3.1-8B	89.0±2.0	93.5±1.8	0.9±0.0	5.2±0.7	3155.5±0.0	41.9±6.0	98.3 TFLOPs

### B.2.3 Analysis on Proceduralization

Table 11 presents a comparative analysis of our neurosymbolic proceduralization method to existing proceduralization methods, evaluated in terms of task performance and reasoning efficiency, with a particular focus on enabling timely reasoning through single-step inference. To ensure a fair comparison, we additionally include a unified setting in which all methods are evaluated under identical inference conditions using the same LLAMA3.1-8B backbone, highlighted in gray background in the table. Under these conditions, NESYPR achieves the lowest average plan generation latency of 5.2 seconds, the highest task success rate of 89.0%, and the fewest input and output tokens—averaging 3,155.5 and 41.9, respectively. BoT and LRM exhibit latencies that are 54.3 and 105.8 seconds longer than NESYPR, respectively, along with 6,125.9 and 2,188.7 more total tokens consumed, due to their reliance on multi-step reasoning. PlaSma, which distills procedural knowledge from larger to smaller LMs, achieves competitive results with efficient inference, reaching a CSR of 80.5% using an 8B LM. Yet, NESYPR outperforms it with a higher CSR of 83.6% while operating with only a 3B LM. While NESYPR incurs slightly higher FLOPs than simpler baselines such as LRM, this overhead stems from its memory-augmented module and contrastive planning mechanisms, which are designed to enhance robustness and correctness in reasoning. It is important to note that in our implementation, the problem distillation step of BoT is performed by GPT-4.1 regardless of the base LM. Consequently, this additional step is excluded from FLOPs measurement, and we denote it as “+ $\alpha$ ” to reflect the unaccounted cost—potentially underestimating the actual computational burden of BoT. In contrast to prompt-based or distillation-based approaches, NESYPR composes and validates procedures during inference, trading off minimal compute overhead for significantly higher planning accuracy and consistency. Despite this moderate FLOPs usage, NESYPR achieves the best trade-off across success rate, latency, and token efficiency. This demonstrates that structured reasoning can remain performant without relying on excessively large models or token-intensive decoding.

### B.2.4 Analysis on Continual Embodied Task Learning

Figure 5 shows how CSR evolves over 10 continual learning phases, where new tasks are introduced at each phase and evaluation is consistently performed on the full task set defined in Phase 10 [39, 40, 41]. We test whether effective continual learning can be achieved by incrementally expanding the procedure-book as new procedural knowledge is acquired. This setting is denoted as NESYPR-Incremental. We compare four baselines. NESYPR-Oracle represents the upper-bound performance, trained on the entire task set with a sufficiently large procedure-book in a single learning phase. DT-Mem is the memory-augmented LM baseline explained in Section B.1.4, adapted to the continual learning setting. NESYPR-Static reuses a fixed-size procedure-book throughout training; we include both small and large variants to observe the effect of capacity. NESYPR-Incremental increases the

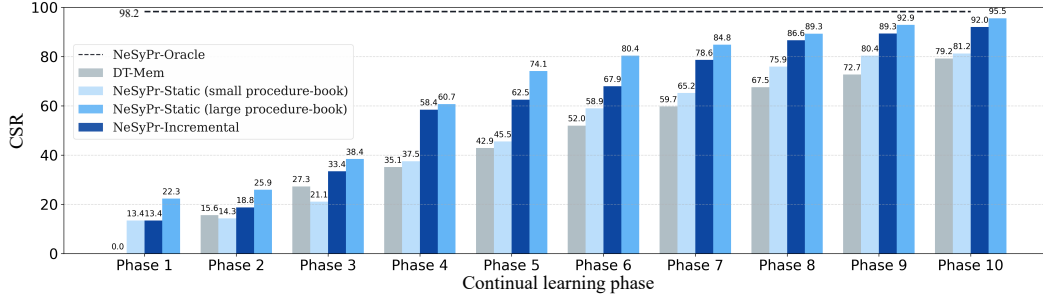


Figure 5: Analysis on continual embodied task learning scenarios

procedure-book size at each phase while continuing neurosymbolic proceduralization on the new train data. It shows steady improvement in CSR and reaches performance comparable to NESYPR-Static with a large procedure-book. This indicates that the agent can effectively accumulate procedural knowledge over time by gradually expanding the memory without needing to retrain from scratch.

Table 12: Evaluation of inference efficiency and task performance of NESYPR across a range of devices, including an embedded device. Latency indicates the time taken for the agent to generate a plan, and memory usage captures the minimum and maximum values observed during continuous logging throughout inference. On GPU-based devices, this reflects GPU memory only, while on Jetson Orin, which lacks a discrete GPU, it reflects overall system memory usage.

Device	Latency (sec)			Memory Usage (GB)		Train			Seen			Unseen		
	Min.	Max.	Avg.	Min.	Max.	SR	GC	SPL	SR	GC	SPL	SR	GC	SPL
RTX A6000	0.5	5.1	1.5	0.0	1.9	97.4	97.7	1.0	88.4	89.6	0.9	61.5	67.9	0.6
RTX 4090	0.5	4.5	1.0	0.0	1.9	96.1	98.7	1.0	88.4	89.7	0.9	61.5	68.5	0.6
RTX 3090	0.5	4.8	1.2	0.0	1.9	92.2	94.6	0.9	86.6	87.8	0.9	61.5	68.5	0.6
RTX 3050	0.6	7.7	1.6	0.0	1.9	94.8	96.9	0.9	84.8	86.5	0.8	61.5	67.9	0.6
Jetson Orin	2.0	11.4	5.0	4.3	8.6	93.5	96.1	0.9	87.5	88.7	0.9	59.6	66.6	0.6

### B.2.5 Evaluation on Inference Efficiency Across Diverse Devices.

Table 12 presents the inference efficiency and task performance of NESYPR across five different devices, ranging from high-end (RTX A6000, 4090) to off-the-shelf GPUs (RTX 3090, 3050) and embedded (Jetson Orin) devices. Despite substantial differences in computational resources, NESYPR demonstrates stable performance across the train, seen, and unseen sets. Across all tested devices, the task performance of NESYPR remains relatively stable, exhibiting less than a 5.5% absolute variation in SR on the train set. High-end GPUs such as the RTX A6000 and 4090 achieve peak SRs exceeding 96%, while even lower-tier devices like the RTX 3090 maintain performance above 92%, demonstrating consistent task execution across diverse hardware configurations. Although the agent stores and reuses procedures during inference, it incurs minimal overhead in both latency and memory usage. On Jetson Orin, the average inference time is approximately 5 seconds—still reasonably fast—demonstrating the potential for deployment on embedded devices. However, the considerable variance between minimum and maximum latency indicates that further optimization may be necessary to ensure more stable performance under highly resource-constrained conditions.

## References

- [1] Malte Helmert. “The fast downward planning system”. In: *Journal of Artificial Intelligence Research* (2006).
- [2] Tom Silver and Rohan Chitnis. “PDDL Gym: Gym environments from PDDL problems”. In: *arXiv preprint arXiv:2002.06432* (2020).
- [3] Manling Li et al. “Embodied agent interface: Benchmarking llms for embodied decision making”. In: *Advances in Neural Information Processing Systems* (2024).
- [4] Mohit Shridhar et al. “Alfworld: Aligning text and embodied environments for interactive learning”. In: *arXiv preprint arXiv:2010.03768* (2020).
- [5] Constructions Aeronautiques et al. “Pddl the planning domain definition language”. In: *Technical Report, Tech. Rep.* (1998).
- [6] Greg Brockman et al. “Openai gym”. In: *arXiv preprint arXiv:1606.01540* (2016).
- [7] Jörg Hoffmann. “FF: The fast-forward planning system”. In: *AI magazine* (2001).
- [8] Xavier Puig et al. “VirtualHome: Simulating household activities via programs”. In: *Proceedings of the 29th IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018.
- [9] Marc-Alexandre Côté et al. “Textworld: A learning environment for text-based games”. In: *Computer Games: 7th Workshop, CGW 2018, Held in Conjunction with the 27th International Conference on Artificial Intelligence, IJCAI 2018, Stockholm, Sweden, July 13, 2018, Revised Selected Papers 7*. 2019.
- [10] Byeonghwi Kim, Minhyuk Seo, and Jonghyun Choi. “Online Continual Learning for Interactive Instruction Following Agents”. In: *ICLR*. 2024.
- [11] Mohit Shridhar et al. “Alfred: A benchmark for interpreting grounded instructions for everyday tasks”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020.
- [12] Wenlong Huang et al. “Language models as zero-shot planners: Extracting actionable knowledge for embodied agents”. In: *Proceedings of the 39th International Conference on Machine Learning*. 2022.
- [13] Tomoyuki Kagaya et al. “Rap: Retrieval-augmented planning with contextual memory for multimodal llm agents”. In: *arXiv preprint arXiv:2402.03610* (2024).
- [14] Chan Hee Song et al. “LLM-planner: Few-shot grounded planning for embodied agents with large language models”. In: *Proceedings of the 19th IEEE/CVF International Conference on Computer Vision*. 2023.
- [15] Jason Wei et al. “Chain-of-thought prompting elicits reasoning in large language models”. In: *Proceedings of the 36th Advances in Neural Information Processing Systems*. 2022.
- [16] Shunyu Yao et al. “Tree of thoughts: Deliberate problem solving with large language models”. In: *Advances in Neural Information Processing Systems* (2024).
- [17] Maciej Besta et al. “Graph of thoughts: Solving elaborate problems with large language models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. 2024.
- [18] Shunyu Yao et al. “ReAct: Synergizing Reasoning and Acting in Language Models”. In: *The Eleventh International Conference on Learning Representations*. 2023.
- [19] Noah Shinn et al. “Reflexion: Language agents with verbal reinforcement learning”. In: *Advances in Neural Information Processing Systems* (2024).
- [20] Weizhi Wang et al. “Augmenting language models with long-term memory”. In: *Advances in Neural Information Processing Systems* (2023).
- [21] Jikun Kang et al. “LM2: Large Memory Models”. In: *arXiv preprint arXiv:2502.06049* (2025).
- [22] Ashish Vaswani et al. “Attention is all you need”. In: *Proceedings of the 31st Advances in Neural Information Processing Systems* (2017).
- [23] Zaijing Li et al. “Optimus-2: Multimodal minecraft agent with goal-observation-action conditioned policy”. In: *arXiv preprint arXiv:2502.19902* (2025).
- [24] Jikun Kang et al. “Think before you act: Decision transformers with working memory”. In: *arXiv preprint arXiv:2305.16338* (2023).
- [25] Vincent Micheli and François Fleuret. “Language models are few-shot butlers”. In: *arXiv preprint arXiv:2104.07972* (2021).

- 500 [26] Ling Yang et al. “Buffer of thoughts: Thought-augmented reasoning with large language  
501 models”. In: *Advances in Neural Information Processing Systems* (2024).
- 502 [27] Ahmed El-Kishky et al. “Competitive programming with large reasoning models”. In: *arXiv  
503 preprint arXiv:2502.06807* (2025).
- 504 [28] Faeze Brahman et al. “PlaSma: Procedural Knowledge Models for Language-based Planning  
505 and Re-Planning”. In: *The Twelfth International Conference on Learning Representations*.  
506 2024.
- 507 [29] Meta. *Llama 3.2 1B Language Model*. 2024. URL: [https://huggingface.co/meta-  
508 llama/Llama-3.2-1B](https://huggingface.co/meta-llama/Llama-3.2-1B).
- 509 [30] Nils Reimers and Iryna Gurevych. “Sentence-BERT: Sentence Embeddings using Siamese  
510 BERT-Networks”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural  
511 Language Processing*. 2019.
- 512 [31] Qwen Team. *Qwen2.5: A Party of Foundation Models*. 2024. URL: [https://qwenlm.github.  
513 io/blog/qwen2.5/](https://qwenlm.github.io/blog/qwen2.5/).
- 514 [32] Wonje Choi et al. “Embodied CoT Distillation From LLM To Off-the-shelf Agents”. In:  
515 *Proceedings of the 41st International Conference on Machine Learning*. 2024.
- 516 [33] Edward J Hu et al. “Lora: Low-rank adaptation of large language models.” In: *ICLR* (2022).
- 517 [34] Aaron Hurst et al. “Gpt-4o system card”. In: *arXiv preprint arXiv:2410.21276* (2024).
- 518 [35] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint  
519 arXiv:1907.11692* (2019).
- 520 [36] Mohit Shridhar et al. “ALFRED: A Benchmark for Interpreting Grounded Instructions for  
521 Everyday Tasks”. In: *The IEEE Conference on Computer Vision and Pattern Recognition  
522 (CVPR)*. 2020.
- 523 [37] Peter Anderson et al. “On evaluation of embodied navigation agents”. In: *arXiv preprint  
524 arXiv:1807.06757* (2018).
- 525 [38] Ori Ram et al. “In-context retrieval-augmented language models”. In: *Transactions of the  
526 Association for Computational Linguistics* (2023).
- 527 [39] Wonje Choi et al. “NeSyC: A Neuro-symbolic Continual Learner For Complex Embodied  
528 Tasks In Open Domains”. In: *arXiv preprint arXiv:2503.00870* (2025).
- 529 [40] Daehee Lee et al. “Incremental learning of retrievable skills for efficient continual task adapta-  
530 tion”. In: *Advances in Neural Information Processing Systems* (2024).
- 531 [41] Bo Liu et al. “Libero: Benchmarking knowledge transfer for lifelong robot learning”. In:  
532 *Advances in Neural Information Processing Systems* (2023).