

# 000 001 002 003 004 005 GRACE: A LANGUAGE MODEL FRAMEWORK FOR 006 EXPLAINABLE INVERSE REINFORCEMENT LEARNING 007 008

009 **Anonymous authors**  
010 Paper under double-blind review  
011  
012  
013  
014  
015  
016  
017  
018  
019  
020  
021  
022  
023

## ABSTRACT

024 Inverse Reinforcement Learning (IRL) aims to recover Reward Models from ex-  
025 pert demonstrations, but traditional methods yield "black-box" models that are  
026 difficult to interpret and debug. In this work, we introduce GRACE (Generating  
027 Rewards As CodE), a method for using code Large Language Models (LLMs)  
028 within an evolutionary search to reverse-engineer an interpretable, code-based re-  
029 ward function directly from expert trajectories. The resulting reward function is  
030 executable code that can be inspected and verified. We empirically demonstrate  
031 that GRACE can efficiently learn highly accurate rewards in the multi-task setups  
032 as defined by two benchmarks, BabyAI and AndroidWorld. Further, we demon-  
033 strate that the resulting reward leads to strong policies compared to both com-  
034 petitive Imitation Learning and online RL approaches with groundtruth rewards.  
035 Finally, we show that GRACE is able to build complex reward APIs in multi-task  
036 setups.  
037  
038

## 1 INTRODUCTION

039 The performance of modern Reinforcement Learning (RL) agents is determined by, among other  
040 factors, the quality of their reward function. Traditionally, reward functions are defined manually as  
041 part of the problem specification. In many real-world settings, however, environments are readily  
042 available while reward functions are absent and must be specified. Manually designing rewards  
043 is often impractical, error-prone, and does not scale, particularly in contemporary multi-task RL  
044 scenarios (Wilson et al., 2007; Teh et al., 2017; Parisotto et al., 2016).

045 A natural alternative is to automate reward specification by learning a reward model from data. The  
046 dominant paradigm here is Inverse Reinforcement Learning (IRL), which attempts to infer a reward  
047 model from observations of expert behavior (Ng & Russell, 2000; Christiano et al., 2017; Ziebart  
048 et al., 2008). In the era of Deep RL, approaches such as GAIL (Ho & Ermon, 2016) represent  
049 rewards with deep neural networks. While effective, these reward functions are typically opaque  
050 black boxes, making them difficult to interpret or verify (Molnar, 2020). Moreover, IRL methods  
051 often require substantial amounts of data and often lead to inaccurate rewards (Sapora et al., 2024).

052 An alternative representation that has recently gained traction is using code to express reward mod-  
053 els (Venuto et al., 2024a; Ma et al., 2023). These approaches leverage code-generating Large Lan-  
054 guage Models (LLMs) and human-provided task descriptions or goal states to generate reward pro-  
055 grams (Venuto et al., 2024a). Subsequently, the generated rewards are verified (Venuto et al., 2024a)  
056 or improved using the performance of a trained policy as feedback (Ma et al., 2023). However,  
057 this prior work has not investigated whether it is possible to recover a reward function purely from  
058 human demonstrations in an IRL-style setting, without utilizing any explicit task description or  
059 domain-specific design assumptions.

060 In this work, we address the question of how to efficiently infer rewards-as-code from expert demon-  
061 strations using Large Language Models (LLMs). We propose an optimization procedure inspired by  
062 evolutionary search (Goldberg, 1989; Eiben & Smith, 2003; Salimans et al., 2017; Romera-Paredes  
063 et al., 2024a; Novikov et al., 2025b), in which code LLMs iteratively introspect over demonstrations  
064 to generate and refine programs that serve as reward models. This perspective effectively revisits the  
065 IRL paradigm in the modern context of program synthesis with LLMs.

Our contributions are threefold. We first demonstrate that code LLMs conditioned on expert demonstrations can produce highly accurate reward models. These rewards generalize well to held-out demonstrations and are well-shaped, providing informative intermediate signals rather than merely verifying final success criteria. We further show that the approach is sample-efficient: accurate rewards are obtained from relatively few demonstrations, in contrast to IRL methods based on neural networks that typically require large amounts of training data. More importantly, directly using demonstrations means no domain knowledge or human-in-the-loop guidance is manually specified during reward generation.

Second, we show that the learned rewards enable training of strong policies. We perform our evaluations in two domains: the procedurally generated navigation environment *BabyAI* (Chevalier-Boisvert et al., 2018) and the real-world device control environment *AndroidWorld* (Rawles et al., 2024) demonstrate that GRACE outperforms established IRL approaches such as GAIL (Ho & Ermon, 2016) as well as online RL with ground-truth rewards (Schulman et al., 2017). This highlights both the efficiency of GRACE in learning rewards and its promise for building capable agents across diverse domains.

Finally, by representing rewards as code, GRACE inherits additional advantages. The resulting rewards are interpretable and verifiable by humans, and, when inferred across multiple tasks, naturally form reusable reward APIs that capture common structure and enable efficient multi-task generalization. Our analysis shows that as the evolutionary search progresses, GRACE shifts from creating new functions to heavily reusing effective, high-level modules it has already discovered, demonstrating the emergence of a modular code library.

## 2 RELATED WORKS

**LLMs for Rewards** A common way to provide verification/reward signals in an automated fashion is to utilize Foundation Models. LLM-based feedback has been used directly by Zheng et al. (2023) to score a solution. Additionally, an LLM can be used to critique examples (Zankner et al., 2024). Comparing multiple outputs in a relative manner has been also explored by Wang et al. (2023). Note that such approaches use LLM in a zero shot fashion with additional prompting and potential additional examples. Hence, they can utilize only a small number of demonstrations at best. In addition to zero shot LLM application, it is also common to train reward models, either from human feedback (Ouyang et al., 2022) or from AI feedback (Klissarov et al., 2023; 2024). Note that such approaches require training a reward model that isn't interpretable and often times require a larger number of examples.

**Code as Reward** As LLMs have emerged with powerful program synthesis capabilities (Chen et al., 2021; Austin et al., 2021; Li et al., 2023; Fried et al., 2022; Nijkamp et al., 2022) research has turned towards generating environments for training agents Zala et al. (2024); Faldor et al. (2025) for various domains and complexities. When it comes to rewards in particular, code-based verifiers use a language model to generate executable Python code based on a potentially private interface such as the environment's full state. Because early language models struggled to reliably generate syntactically correct code, the first code-based verifiers (Yu et al., 2023; Venuto et al., 2024b) implemented iterative re-prompting and fault-tolerance strategies. More recent approaches focus on progressively improving a syntactically correct yet suboptimal reward function, particularly by encouraging exploration (Romera-Paredes et al., 2024b; Novikov et al., 2025a). Other approaches such as Zhou et al. (2023); Dainese et al. (2024) use search in conjunction with self-reflection (Madaan et al., 2023) to provide feedback.

**Inverse Reinforcement Learning (IRL)** Early approaches infer a reward function by requiring the expert policy to outperform all alternatives (Ng & Russell, 2000). While related to our formulation, our representation (code) and our optimization strategy (evolutionary search) are fundamentally different. Subsequent works have focused on directly learning policies without explicit reward recovery (Abbeel & Ng, 2004), while incorporating entropy regularization (Ziebart et al., 2008) or leveraging convex formulations (Ratliff et al., 2006). In contrast, GRACE benefits from implicit regularization through its symbolic reward representation, though evolutionary search provides no optimization guarantees. More recently, Imitation Learning (IL) has achieved considerable practical success (Ross et al., 2011), often by training a discriminator to distinguish expert from non-expert trajectories (Ho & Ermon, 2016; Swamy et al., 2021). While such discriminators define implicit rewards, our approach instead operates with explicit reward representations.

108 

### 3 METHOD

109 

#### 3.1 BACKGROUND

110 **Reinforcement Learning** We consider a finite-horizon Markov Decision Process (MDP) (Put-  
 111 eman, 2014) parameterized by  $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, T, r \rangle$  where  $\mathcal{S}$ ,  $\mathcal{A}$  are the state and action spaces,  
 112  $T : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$  is the transition operator, and  $R$  is a reward function. The agent’s behavior is  
 113 described by the policy  $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$ . Starting from a set of initial states  $\mathcal{S}_0 \subset \mathcal{S}$ , the agent takes  
 114 the action  $a \sim \pi(s)$  at  $s$ , receives a reward  $r(s)$  and transitions into state  $s' \sim T(s, a)$ .

115 The performance of the agent is measured with expected cumulative per-timestep rewards, referred  
 116 to as return:  
 117

$$118 \quad J(\pi, r) = \mathbb{E}_{\tau \sim \pi, T} \left[ \sum_{t=1}^H r(s_t) \right] \quad (1)$$

120 where  $\tau$  are trajectory unrolls of horizon  $H$  of the policy  $\pi$  in  $\mathcal{M}$ . An optimal agent can be learned  
 121 by maximizing Equation (1) via gradient descent with respect to the policy, also known as policy  
 122 gradient (Sutton et al., 1999; Schulman et al., 2017).

123 **Inverse Reinforcement Learning** If the reward  $r$  is not specified, it can be learned from demon-  
 124 strations of an expert policy  $\pi_E$ . In particular, the classical IRL objective learns a reward whose  
 125 optimal return is attained by the expert (Ng & Russell, 2000; Syed & Schapire, 2007):  
 126

$$127 \quad \min_{\pi} \max_{R} J(\pi_E, r) - J(\pi, r) \quad (2)$$

128 More recent Imitation Learning (IL) approaches learn a discriminator that distinguishes between  
 129 expert and non-expert demonstrations (Ho & Ermon, 2016; Swamy et al., 2021). The likelihood  
 130 of the agent’s data under the trained discriminator can be implicitly thought of as a reward. These  
 131 approaches utilize gradient based methods to optimize their objectives.

132 **Evolutionary search** As an alternative for cases where the objective is not readily differentiable,  
 133 gradient-free methods can be employed. One such method is evolutionary search, which maintains a  
 134 set of candidate solutions (called a population) and applies variation operators to improve it. These  
 135 operators include mutation, where a hypothesis is partially modified, and recombination, where  
 136 two hypotheses are combined to produce a new one. Each variation is evaluated using a fitness  
 137 function, which measures the quality of a given hypothesis. Starting with an initial population,  
 138 evolutionary search repeatedly applies these variation operators, replacing hypotheses with higher-  
 139 fitness alternatives.

140 In this work, we focus on inferring reward functions, represented as Python code, from a set of  
 141 demonstrations. While this setup is related to IRL, representing rewards as code prevents us from  
 142 applying gradient-based methods commonly used in IRL. For this reason, we adopt evolutionary  
 143 search as our optimization method.

144 

#### 3.2 GRACE

145 We propose GRACE - Generating Rewards As CodE, an interpretable IRL framework that gener-  
 146 ates a reward function as executable Python code. Initially, an LLM analyzes expert and random  
 147 trajectories to **optionally** identify goal states (Phase 1) and generates a preliminary set of reward  
 148 programs. **The step of goal identification is optional and can be skipped in favor of directly querying**  
 149 **the LLM for a reward function which best matches the expert trajectories.** This initial set is then  
 150 iteratively improved through evolutionary search, where the LLM mutates the code based on mis-  
 151 classified examples to maximize a fitness function (Phase 2). Finally, an RL agent is trained using  
 152 the refined reward, and the new trajectories it generates are used to further expand the dataset and  
 153 further improve the reward function (Phase 3). The overall process is illustrated in Figure 1 and  
 154 detailed below and in Algorithm 1

155 **Phase 1: Initialization** The initial reward code generation by GRACE is based on a set of demon-  
 156 stration trajectories  $\mathcal{D}^+$  and a set of random trajectories  $\mathcal{D}^-$ . The former is generated using an expert  
 157 policy or human demonstrations depending on the concrete setup, while the latter is produced by a  
 158 random policy. Note that with a slight abuse of notation we will use  $\mathcal{D}$  to denote interchangeably a  
 159 set of trajectories as well the set of all states from these trajectories.

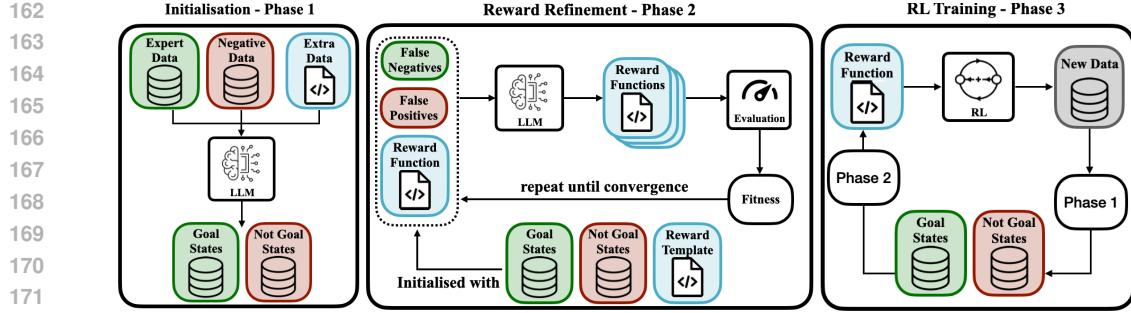


Figure 1: Overview of the GRACE framework. (a) The expert, negative and extra data (if any) is used to identify goal states. By default, all expert states are classified as goal states and all negative states as non-goal states (b) The goal and non-goal states are used to generate reward functions through an evolutionary procedure. The rewards are iteratively refined by feeding the examples misclassified by the reward. (c) An agent is trained with online RL using the converged reward; the data it sees during the training is classified by the LLM into  $\mathcal{D}^+$ ,  $\mathcal{D}^-$  and used to further improve the reward.

The language model is prompted with a random subset of  $\mathcal{D}^+$  and, optionally, extra information available about the environment (e.g. its Python code or tool signature), to produce two artifacts:

**Initial rewards:** The LLM generates an initial set  $\mathcal{R}^{\text{init}}$  of reward functions. Each function  $r \in \mathcal{R}^{\text{init}}$  is represented as Python code:

```
def reward(state: string) -> float:
    <LLM produced code>
```

**(Optional) Goal states:** The LLM analyzes the states from expert demonstrations to identify the subset of goal states  $\mathcal{S}_g \subseteq \mathcal{D}^+$  that solve the task - these are positive samples. All remaining non-goal states  $\mathcal{S}_{ng} = \{\mathcal{D}^+ \setminus \mathcal{S}_g\} \cup \mathcal{D}^-$  are initially treated as negative samples.

designed to assign high values to goal states  $\mathcal{S}_g$  and low values to non goal ones  $\mathcal{S}_{ng}$ . This set of rewards is treated as the population in the subsequent evolution phase.

**Phase 2: Reward Refinement through Evolutionary Search** GRACE uses Evolutionary Search to obtain rewards that best explain the current set of goal and non goal states. This is achieved by *mutating* the current reward population  $\mathcal{R}$  using a code LLM and retaining rewards with high *fitness*.

The *fitness*  $f$  of a reward function  $r$  measures how well this function assigns large values to goal and small values to non-goal states, akin to what would be expected from a meaningful reward:

$$f(r) = \mathbb{E}_{s \sim \mathcal{S}_g}[r(s)] - \mathbb{E}_{s \sim \mathcal{S}_{ng}}[r(s)] \quad (3)$$

In practice, to normalize the fitness computation, we bound the reward signal. Any reward value greater than or equal to a predefined  $r(s) \geq \tau$  is treated as 1, and any value below is treated as 0 for the purpose of this calculation.

The *mutation* operator  $m$  of a reward, that is used to improve the current reward population, is based on an LLM that is prompted to introspect the reward code and address failures. To do so it is provided with several inputs pertaining to the source code of the reward (if available), misclassified states, and additional debugging information:

$$m(r) = \text{LLM}(\text{source}(r), \text{info}, \text{prompt}) \quad (4)$$

In more detail,  $\text{source}(r)$  is the Python code for the reward. The  $\text{info} = (s_g, r(s_g), s_e, \text{debug}(r, s_g))$  is intended to focus the model on failures by honing onto states misclassified by the reward. It consists of a sequence of misclassified states  $s \in S$ , their reward value  $r(s)$ , as well as a debugging info  $\text{debug}(r, s)$  produced by printing intermediate values during the execution of  $r$  on the misclassified state  $s$ . The composition of this feedback is intentionally varied; each prompt contains a different

216 number of examples, presented as either individual states or full trajectories. To help the model dis-  
 217 criminate between true and false positives, prompts containing a false positive are augmented with  
 218 an expert state  $s_e \sim D^+$ .

219 We repeatedly apply the above mutation operation to modify the reward population  $\mathcal{R}$  to improve  
 220 its fitness. In more detail, we repeatedly sample a reward  $r \in \mathcal{R}$  with probability  $\frac{\exp F(r)}{\sum_{r' \in \mathcal{R}_P^i} \exp(F(r'))}$ .  
 221 Subsequently, we apply the mutation and keep the new reward function only if it has a higher fitness  
 222 than other already created rewards. After  $K$  mutations, we return the reward function with highest  
 223 fitness  $r^* = \arg \max_{r \in \mathcal{R}} \{f(r)\}$ . This phase is presented as function EVOSEARCH in Algorithm 1.  
 224

225 **Phase 3: Training Trajectory Expansion via Reinforcement Learning** The optimal reward  $r^*$   
 226 above is obtained by inspecting existing demonstrations. In order to further improve the reward, we  
 227 ought to collect further demonstrations by training a policy  $\pi_{r^*}$  using the current optimal reward  $r^*$ ;  
 228 and use this policy to collect additional data  $\mathcal{D}_{r^*}$ .

229 In more detail, we employ PPO (Schulman et al., 2017) to train a policy in the environment of  
 230 interest. As this process can be expensive, we use a predefined environment interaction budget  $N$   
 231 instead of training to convergence. After obtaining these additional trajectories, we use the same  
 232 process as described in Sec. (3.2, Phase 1) to identify goal  $\mathcal{S}_{g^*}$  and non-goal states  $\mathcal{S}_{ng^*}$ . The new  
 233 trajectories are likely to contain new edge cases and examples of reward hacking, if any. These are  
 234 used to further refine the reward population as described in the preceeding Sec. (3.2, Phase 2.1).  
 235 The process terminates when the RL agent achieves a desired level of performance. This phase is  
 236 presented as function DATAEXPAND in Algorithm 1.  
 237

238 The final algorithm, presented in Algorithm 1, consists of repeatedly performing Evolutionary  
 239 Search over reward population  $\mathcal{R}$  followed by data expansion using RL-trained policy. Each it-  
 240 eration is called a generation.

241 **Additional reward shaping** When the reward function offline performance on  $\mathcal{D}$  doesn't translate  
 242 to good online RL performance, we assume that the reward signal is poorly shaped, and additional  
 243 refinement is required. In these cases, the LLM's info in Eq. 4 is augmented beyond misclassified  
 244 states to include failed trajectory examples from  $\mathcal{D}_{r^*}$ . To achieve this, we instruct the LLM to  
 245 reshape the reward function, using expert trajectories as a reference, so that it provides a signal that  
 246 increases monotonically towards the goal.

247 **Discussion** The above algorithm iterates between policy optimization and reward optimization.  
 248 The objective for the latter is the fitness function from Eq. 3. If one flips the reward on non-goal  
 249 states of positive demonstrations or goal states in learned policy demonstrations, it is straightforward  
 250 to show that GRACE optimizes the canonical IRL objective using Evolutionary Search.

251 **Proposition 1.** Suppose  $m(s) = 1$  iff  $s \in \mathcal{S}_g$ , else  $m(s) = -1$ , then GRACE optimizes,  
 252  $\min_{\pi} \max_r J(\pi_E, m \circ r) - J(\pi, -m \circ r)$ , which is a variation of Eq. (2).

253 The proof can be found in Appendix A.1.

## 254 4 EXPERIMENTS

255 We empirically evaluate GRACE with respect to its ability to generate rewards that lead to effective  
 256 policy learning. Specifically, we aim to address the following questions:

257 **Accuracy and Generalization:** Can GRACE recover correct rewards, and how much supervi-  
 258 sion is required to do so?

259 **Policy Learning Performance:** How does GRACE compare to other IRL methods or to online  
 260 RL trained with ground-truth rewards?

261 **Qualitative Properties:** How well-shaped are the rewards produced by GRACE?

262 **Interpretability and Multi-Task Efficacy:** Does GRACE produce reward APIs that can be  
 263 shared across tasks?

270

**Algorithm 1** GRACE: Generating Rewards As Code

271

272

**Inputs:**

273

 $\mathcal{D}^+$ : expert trajectories  
 $\mathcal{D}^-$ : random trajectories

274

**Parameters:**

275

 $P$ : reward population size  
 $K$ : mutation steps  
 $M$  number of generations  
 $N$ : RL budget

276

**procedure** GRACE( $\mathcal{D}^+, \mathcal{D}^-$ )

277

// Phase 1: Initialization.

278

 $\mathcal{S}_g = \{s \in \mathcal{D}^+ \mid \text{LLM}(s, \text{goal\_prompt})\}$   
 $\mathcal{S}_{ng} = \mathcal{D}^+ \cup \mathcal{D}^- / \mathcal{S}_g$   
 $\mathcal{R} = \{\text{LLM}(S_n, S_{ng}, \text{reward\_prompt})\}$ 

279

// Reward Refinement.

280

**for**  $i = 1 \dots M$  **do**

281

 $\mathcal{R} \leftarrow \text{EVOSEARCH}(\mathcal{R}, \mathcal{S}_g, \mathcal{S}_{ng})$   
     $\mathcal{D}, \mathcal{S}_g^*, \mathcal{S}_{ng}^* \leftarrow \text{DATAEXPANDRL}(\mathcal{R})$   
     $\mathcal{S}_g = \mathcal{S}_g^* \cup \mathcal{S}_g, \mathcal{S}_{ng} = \mathcal{S}_{ng}^* \cup \mathcal{S}_{ng}$ 

282

**end for**

283

**return**  $r^* = \arg \max_{r \in \mathcal{R}} f(r)$ **end procedure**

290

291

292

293

294

295

**4.1 EXPERIMENTAL SETUP**

296

297

To evaluate GRACE, we conduct experiments in two distinct domains: the procedurally generated maze environment *BabyAI* (Chevalier-Boisvert et al., 2018), which tests reasoning and generalization, and the Android-based UI simulator *AndroidWorld* (Rawles et al., 2024), which tests control in high-dimensional action spaces.

302

**BabyAI** Our *BabyAI* evaluation suite comprises 20 levels, including 3 custom levels designed to test zero-shot reasoning on tasks not present in public datasets, thereby mitigating concerns of data contamination. Expert demonstrations are generated using the *BabyAI*-Bot (Farama Foundation et al., 2025), which algorithmically solves BabyAI levels optimally. We extend the bot to support our custom levels as well. For each level, we gather approximately 500 expert trajectories. Another 500 negative trajectories are collected by running a randomly initialized agent in the environment. The training dataset consists of up to 16 trajectories, including both expert and negative examples. All remaining trajectories constitute the test set. For each dataset, we evolve the reward on the train trajectories and report both train and test fitness from Eq. (3).

310

311

The state is represented by a  $(h, w, 3)$  array. The state is fully observable, with the first channel containing information about the object type (with each integer corresponding to a different object, such as box, key, wall, or agent), the second channel contains information about the object's color and the third any extra information (e.g. agent direction, if is the door locked).

314

315

**Android** To assess GRACE in a high-dimensional, real-world setting, we use the *AndroidControl* dataset (Rawles et al., 2023; Li et al., 2024), which provides a rich collection of complex, multi-step human interactions across standard Android applications. The state space includes both raw screen pixels and the corresponding XML view hierarchy.

318

319

320

321

322

323

From this dataset, we curate a subset of trajectories focused on the *Clock* application, where users successfully complete tasks such as "set an alarm for 6AM." These serve as our positive examples. Negative samples are drawn from trajectories in other applications (e.g., *Calculator*, *Calendar*, *Settings*). For each negative trajectory, we randomly assign an instruction from the positive set, ensuring the instruction is clock-related but the trajectory completes a task in an unrelated app. We use 80% of trajectories in the train set and the remaining for the test set.

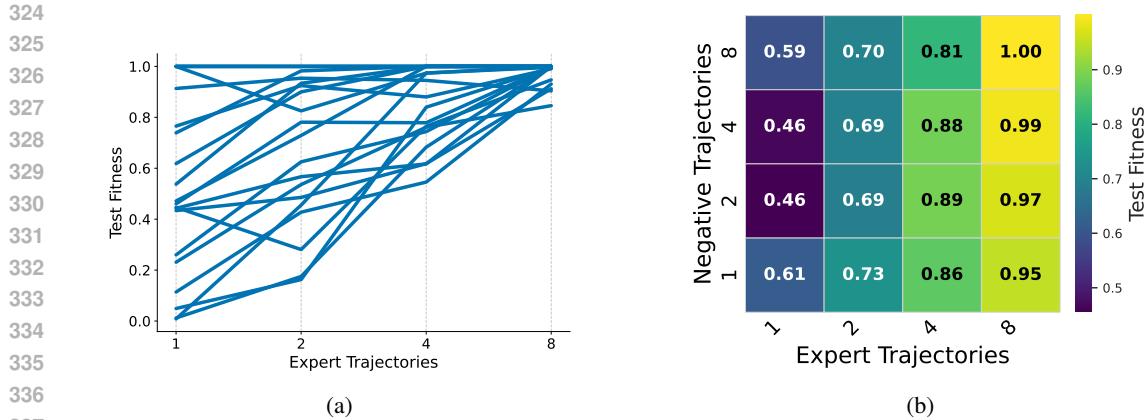


Figure 2: **Fitness vs Number of Expert Trajectories.** The fitness is computed on test dataset after obtaining maximum fitness on training data with corresponding number of expert and negative training trajectories. (a) Performance on all 20 BabyAI tasks. (b) Aggregate fitness across 20 BabyAI tasks.

**MuJoCo** We finally conduct additional experiments on 4 challenging tasks from the classical MuJoCo continuous control suite (Todorov et al., 2012): Hopper, Walker, Ant, Humanoid. These tasks demonstrate that GRACE also excels at reward design in continuous action and state spaces. In these experiments, we don't perform the goal identification step and simply classify all expert states as Goal states and all learner states as Non-Goal states. We run all our MuJoCo experiments using the fully differentiable physics engine Brax (Freeman et al., 2021) to speed up learning. Unlike the BabyAI and Android experiments, in MuJoCo we update the dataset 5 times ( $M = 5$ ) with new trajectories coming from the learner policy. The reward is only updated if the fitness is low on the newly added trajectories.

**GRACE Parameters** All parameters of our approach used across our experiments can be found in Appendix A.6.

## 4.2 ANALYSIS

**GRACE recovers rewards with high accuracy.** We first examine whether GRACE evolutionary search (Phase 1) can successfully recover the underlying task reward from demonstrations alone. We evaluate this in two settings using *BabyAI*: (i) a single-level setting, where the model infers a task-specific reward, and (ii) a more challenging multi-level setting, where GRACE must learn a single, general reward function conditioned on both state and a language goal.

In Figures 2 and 3, we show that the fitness consistently reaches 1.0 across all BabyAI tasks in both single- and multi-level settings, as well as on AndroidControl. A fitness of 1.0 corresponds to assigning higher values to all goal states than to non-goal states.

We further ablate two aspects of the algorithm. First, we analyze sample efficiency by varying the number of expert and negative demonstrations. Results on BabyAI (Figure 2a) show non-trivial performance even with a single demonstration, with gradual improvement and perfect scores achieved using only eight expert trajectories. The number of negative trajectories also plays a role, though to a lesser degree: for example, fitness of 0.95 is achieved with just a single negative trajectory, provided that sufficient expert trajectories are available (Figure 2b).

Finally, we assess the robustness and efficiency of the evolutionary process. As shown in Figure 3, in the multi-task setting GRACE reliably converges to a high-fitness reward function in fewer than 100 generations (i.e., evolutionary search steps), demonstrating the effectiveness of our LLM-driven refinement procedure.

**GRACE outperforms other IRL and online RL:** To validate the quality of the inferred reward model, we compare against two approaches. First, we employ PPO Schulman et al. (2017), as a representative algorithm for online RL, with both GRACE as a reward as well as a groundtruth

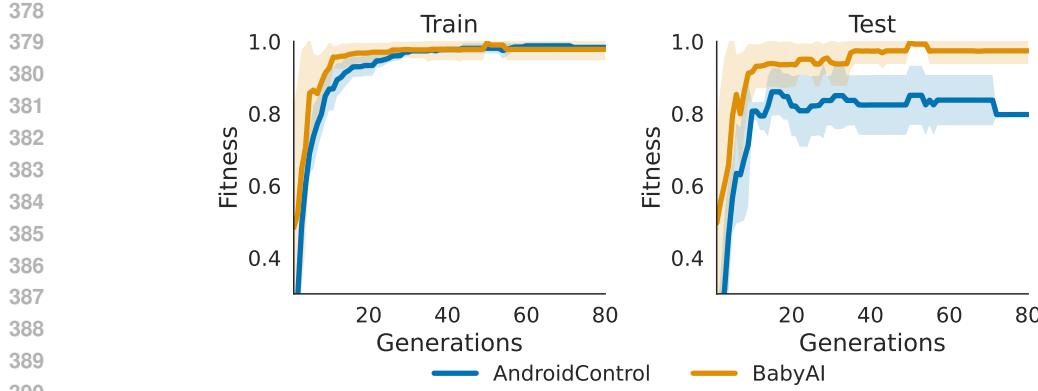


Figure 3: **Fitness vs Number of generations.** Evolution of train and test fitness across evolution generations, as defined by Algorithm 1, for BabyAI (multi-level settings) and AndroidControl (bottom) for "set alarm" task. For BabyAI, we provide 8 expert trajectories and 8 negative trajectories for each task. Shading is standard deviation across 3 seeds. **For these experiments, no online data is added beyond the initial trajectories provided ( $M = 1$ ).**

sparse success reward. Clearly, the latter should serve as an oracle, while it does not benefit from dense rewards.

As an IRL baseline, we compare against GAIL (Ho & Ermon, 2016), that trains a policy whose behavior is indiscriminable from the expert data, as judged by a learned discriminator. GAIL is trained with a large dataset of 2,000 expert trajectories per task, substantially larger than our train data of 8 expert trajectories.

As shown in Table 1 and 2, GRACE consistently matches or outperforms GAIL across all tasks with lesser training data. On several BabyAI tasks, GRACE matches Oracle PPO **with ground-truth rewards**, whereas GAIL completely fails. This demonstrates that the interpretable, code-based rewards from GRACE are practically effective, enabling successful downstream policy learning. To ensure a fair comparison, the agents for the GAIL baseline and GRACE are trained using the same underlying PPO implementation, agent architecture and hyperparameters as the oracle. Performance is measured by the final task success rate after  $1e7$  environment steps. No extra information or environment code is provided in context to GRACE.

Similarly, we use the evolved reward function on the AndroidControl dataset to finetune our agent on the Clock *AndroidWorld* tasks: ClockStopWatchPausedVerify, ClockStopWatchRunning and Clock-TimerEntry. The agent obtains near perfect performance on the Stopwatch tasks zero-shot, but learning on our reward doesn't decrease performance. The training curves for all tasks are reported in Figure 4.

	PPO	GRACE w/ GPT-4o	GRACE w/ Qwen3-Coder-30B	GAIL w/ 10 traj	GAIL w/ 200 traj
<b>Hopper</b>	$2212 \pm 54$	$2143 \pm 80$	$2106 \pm 76$	$1902 \pm 183$	$2056 \pm 92$
<b>Walker</b>	$2675 \pm 292$	$2072 \pm 576$	$2229 \pm 600$	$790 \pm 90$	$1982 \pm 101$
<b>Ant</b>	$6239 \pm 237$	$5707 \pm 210$	$6085 \pm 804$	$3871 \pm 408$	$5521 \pm 674$
<b>Humanoid</b>	$6455 \pm 302$	$5809 \pm 106$	$5921 \pm 301$	$4772 \pm 251$	$6521 \pm 337$

Table 1: **Average returns on 4 MuJoCo (BRAX) continuous control tasks.** Average and standard deviation is reported across 5 different seeds. The total number of required LLM calls to recover a reward for each task averages at 200 for both GPT-4o and Qwen3-Coder-30B.

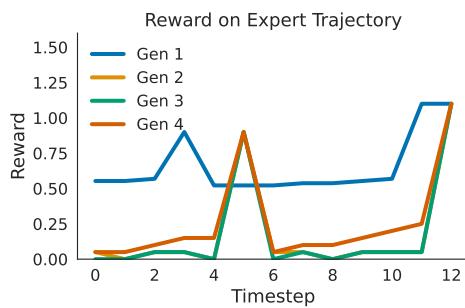
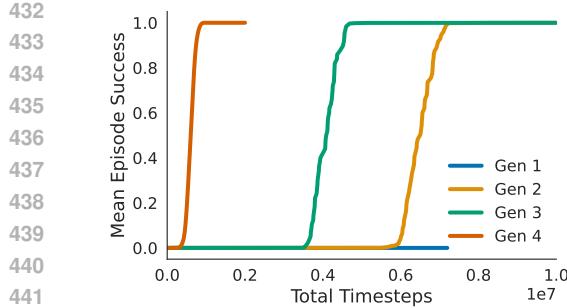


Figure 5: **Shaping** Using the default reward recovered by GRACE occasionally leads to failure in learning the correct behavior due to poor shaping. Through the targeted shaping in Phase 3, we significantly improve final performance and speed of learning.

Task	PPO	GAIL	GRACE
GoToRedBallNoDist	1.00	1.00	1.00
GoToRedBall	1.00	0.35	1.00
PickupDist	0.31	0.15	0.32
PickupLoc	0.21	0.00	0.26
GoToObj	1.00	0.92	1.00
OpenDoorColor	1.00	0.98	1.00
OpenTwoDoors	1.00	0.37	1.00
PlaceBetween (new)	0.09	0.01	0.09
OpenMatchingDoor (new)	0.79	0.20	0.35
Multi-task	0.95	0.31	0.92

Table 2: **Success rates on selected BabyAI environments.** GRACE compared against PPO and GAIL. GRACE uses 8 expert trajectories per task, while GAIL uses 2000.

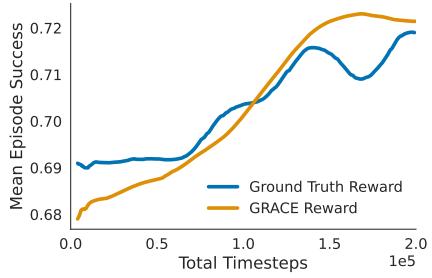


Figure 4: **Training Curves for AndroidWorld Clock Tasks.** Mean episode success over the 3 *AndroidWorld* clock tasks: ClockStopWatchPausedVerify, ClockStopWatchRunning, and ClockTimerEntry.

**GRACE generates well shaped rewards:** We demonstrate GRACE’s ability to produce well-shaped rewards that accelerate learning. For challenging, long-horizon tasks like OpenTwoDoors, a correct but unshaped reward can lead to local optima where the agent gets stuck (Figure 5, "Gen 1"). By explicitly tasking the LLM to introduce shaping terms during Phase 3, GRACE refines the reward to provide a denser learning signal. As shown in Figure 5, this targeted shaping dramatically improves both the final performance and the speed of learning, allowing the agent to solve the task efficiently. This confirms that GRACE not only finds what the goal is but also learns how to guide an agent towards it.

**GRACE Code Reuse:** A key advantage of representing rewards as code is the natural emergence of reusable functions that collectively form a domain-specific reward library. We study this phenomenon in the multi-task *BabyAI* setting (Figure 6). In the early generations of evolutionary search, GRACE actively generates many new modules to explore alternative reward structures. After generation 10, the rate of new module creation drops sharply. At this point, GRACE shifts toward reusing the most effective, high-level modules it has already discovered.

To further illustrate this reuse, Figure 6 (right) shows call counts for a selected set of modules within the evolving reward API. For instance, the *Goal* module, which summarizes a set of goals, is initially used sparingly but becomes heavily invoked following a code refactor at generation 30. Likewise, the *agent\_pos* function is reused at least five times after its introduction. These trends demonstrate that GRACE progressively builds a reward library that supports efficient multi-task generalization.

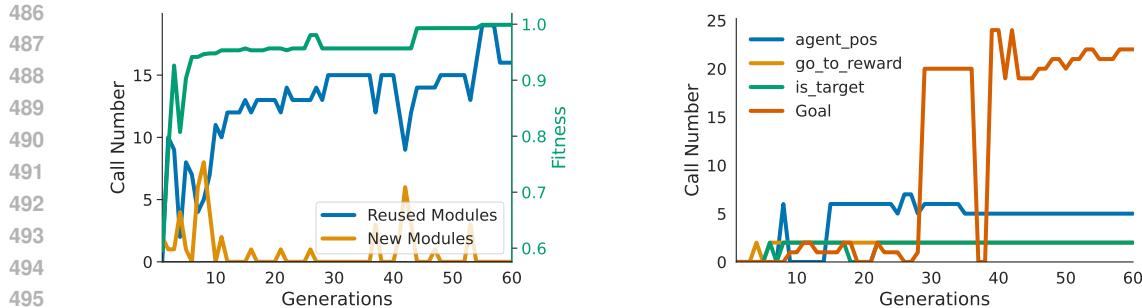


Figure 6: **Module and function reuse across generations** On the left, we show at each generation step the number of newly created modules and the number of existing and thus reused modules from prior rewards, contrasted with the fitness in the reward population. On the right, we show number of times a module are being re-used, for a select set of modules.

## 5 DISCUSSION

**Limitations** A key limitation of GRACE is its limited scalability to high-dimensional state spaces for evolving reward functions. First, generating a reward from high-dimensional observations (such as pixels or waveform audio) directly requires the model to perform symbolic feature extraction. Second, the amount of expert and suboptimal trajectories that can be passed to the LLM is limited by its context length, which makes learning GRACE rewards from large datasets challenging.

**Conclusion** We introduce GRACE, a novel framework that leverages LLMs within an evolutionary search to address the critical challenge of interpretability in IRL. Our empirical results demonstrate that by representing reward functions as executable code, we can move beyond the "black-box" models of traditional IRL and produce rewards that are transparent, verifiable, and effective in RL learning. We show that GRACE successfully recovers accurate and generalizable rewards from few expert trajectories, in stark contrast to deep IRL methods like GAIL. This sample efficiency suggests that the strong priors and reasoning capabilities of LLMs provide a powerful inductive bias. Furthermore, we demonstrate the framework's practical utility by applying it to the complex AndroidWorld environment, showing that GRACE can learn rewards for a variety of tasks directly from unlabeled user interaction data with real-world applications.

## 6 REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our research, we commit to making our code, datasets, and experimental configurations publicly available upon acceptance of this paper. We have already included extensive details within the paper itself. The appendix provides the full prompts used to interact with the LLM for goal identification, initial reward generation, evolutionary mutation, and reward shaping (Appendix A.9). Furthermore, all hyperparameters required to reproduce our results are listed in Appendix A.6.

## REFERENCES

Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, pp. 1–8. ACM, 2004.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,

540 Scott Gray, Nick Ryder, Michael Pavlov, Alethea Power, Lukasz Kaiser, Miljan Bavarian,  
 541 Clemens Winter, Phil Tillet, Felipe Such, Dave Cummings, Matthias Plappert, Fotios Chantzis,  
 542 Elizabeth Barnes, Ariel Herbert-Voss, William Guss, Alex Nichol, Igor Paino, Nikolas Tezak,  
 543 Jie Tang, Igor Babuschkin, Suchir Balaji, Suyog Jain, William Saunders, Christopher Hesse,  
 544 Mark Carr, Aitor Lewkowycz, David Dohan, Howard Mao, Lily Thompson, Erica Frank, Joshua  
 545 Chen, Victor Butoi, David Hernandez, Liane DasSarma, Maxwell Chan, Mateusz Litwin, Scott  
 546 Gray, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario  
 547 Amodei. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*,  
 548 2021.

549 Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia,  
 550 Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of  
 551 grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

552 Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep  
 553 reinforcement learning from human preferences. In *Advances in Neural Information Processing  
 554 Systems (NeurIPS)*, pp. 4299–4307, 2017.

555 Nicola Dainese, Matteo Merler, Minttu Alakuijala, and Pekka Marttinen. Generating code world  
 556 models with large language models guided by monte carlo tree search. *Advances in Neural Infor-  
 557 mation Processing Systems*, 37:60429–60474, 2024.

558 Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Springer, 2003.

559 Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. Omni-epic: Open-endedness via  
 560 models of human notions of interestingness with environments programmed in code. In *Inter-  
 561 national Conference on Learning Representations*, 2025. URL [https://openreview.net/  
 562 forum?id=Y1XkzMJpPd](https://openreview.net/forum?id=Y1XkzMJpPd).

563 Farama Foundation, Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente,  
 564 Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Min-  
 565 grid: Modular customizable reinforcement learning environments. [https://github.com/  
 566 Farama-Foundation/Minigrid](https://github.com/Farama-Foundation/Minigrid), 2025. Accessed: 2025-09-24.

567 C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem.  
 568 Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL [http://  
 569 /github.com/google/brax](https://github.com/google/brax).

570 Daniel Fried, Joshua Ainslie, David Grangier, Tal Linzen, and Dani Yogatama. Incoder: A genera-  
 571 tive model for code infilling and synthesis. In *International Conference on Learning Representa-  
 572 tions (ICLR)*, 2022.

573 David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-  
 574 Wesley, 1989.

575 Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural  
 576 Information Processing Systems*, volume 29, 2016.

577 Martin Klissarov, Pierluca D’Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal  
 578 Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence  
 579 feedback. *arXiv preprint arXiv:2310.00166*, 2023.

580 Martin Klissarov, Devon Hjelm, Alexander Toshev, and Bogdan Mazoure. On the modeling capabili-  
 581 ties of large language models for sequential decision making. *arXiv preprint arXiv:2410.05656*,  
 582 2024.

583 Raymond Li, Loubna Ben Allal, Yacine Jernite Zi, Denis Kocetkov, Chenxi Mou, Aleksandra Piktus,  
 584 Laura Weber, Wenhao Xiao, Jihad Bibi, Stella Biderman, et al. Starcoder: may the source be with  
 585 you! *arXiv preprint arXiv:2305.06161*, 2023.

586 Wei Li, William Bishop, Alice Li, Chris Rawles, Folawiyo Campbell-Ajala, Divya Tyamagundlu,  
 587 and Oriana Riva. On the effects of data scale on computer control agents. *arXiv e-prints*, pp.  
 588 arXiv–2406, 2024.

594 Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman,  
 595 Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via  
 596 coding large language models. *arXiv preprint arXiv: Arxiv-2310.12931*, 2023.

597

598 Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri  
 599 Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement  
 600 with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594, 2023.

601

602 Christoph Molnar. *Interpretable machine learning*. Lulu. com, 2020.

603

604 Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings  
 605 of the Seventeenth International Conference on Machine Learning (ICML)*, pp. 663–670. Morgan  
 606 Kaufmann, 2000.

607

608 Erik Nijkamp, Richard Pang, Hiroaki Hayashi, Tian He, Baptiste Roziere, Canwen Xu, Susan Li,  
 609 Dan Jurafsky, Luke Zettlemoyer, Veselin Stoyanov, and Hyung Won Chung. Codegen: An open  
 610 large language model for code with multi-turn program synthesis. In *International Conference on  
 611 Learning Representations (ICLR)*, 2022.

612

613 Alexander Novikov, NgÂN VÃ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt  
 614 Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian,  
 615 et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint  
 616 arXiv:2506.13131*, 2025a.

617

618 Alexander Novikov, NgÂN VÃ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt  
 619 Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian,  
 620 M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian  
 621 Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and  
 622 algorithmic discovery, 2025b. URL <https://arxiv.org/abs/2506.13131>.

623

624 OpenAI, :, Aaron Hurst, Adam Lerer, Adam P. Goucher, Adam Perelman, Aditya Ramesh, Aidan  
 625 Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, Aleksander MÄEdry, Alex  
 626 Baker-Whitcomb, Alex Beutel, Alex Borzunov, Alex Carney, Alex Chow, Alex Kirillov, Alex  
 627 Nichol, Alex Paino, Alex Renzin, Alex Tachard Passos, Alexander Kirillov, Alexi Christakis,  
 628 Alexis Conneau, Ali Kamali, Allan Jabri, Allison Moyer, Allison Tam, Amadou Crookes, Amin  
 629 Tootoochian, Amin Tootoonchian, Ananya Kumar, Andrea Vallone, Andrej Karpathy, Andrew  
 630 Braunstein, Andrew Cann, Andrew Codispoti, Andrew Galu, Andrew Kondrich, Andrew Tul-  
 631 loch, Andrey Mishchenko, Angela Baek, Angela Jiang, Antoine Pelisse, Antonia Woodford,  
 632 Anuj Gosalia, Arka Dhar, Ashley Pantuliano, Avi Nayak, Avital Oliver, Barret Zoph, Behrooz  
 633 Ghorbani, Ben Leimberger, Ben Rossen, Ben Sokolowsky, Ben Wang, Benjamin Zweig, Beth  
 634 Hoover, Blake Samic, Bob McGrew, Bobby Spero, Bogo Giertler, Bowen Cheng, Brad Lightcap,  
 635 Brandon Walkin, Brendan Quinn, Brian Guarraci, Brian Hsu, Bright Kellogg, Brydon Eastman,  
 636 Camillo Lugaressi, Carroll Wainwright, Cary Bassin, Cary Hudson, Casey Chu, Chad Nelson,  
 637 Chak Li, Chan Jun Shern, Channing Conger, Charlotte Barette, Chelsea Voss, Chen Ding, Cheng  
 638 Lu, Chong Zhang, Chris Beaumont, Chris Hallacy, Chris Koch, Christian Gibson, Christina Kim,  
 639 Christine Choi, Christine McLeavey, Christopher Hesse, Claudia Fischer, Clemens Winter, Coley  
 640 Czarnecki, Colin Jarvis, Colin Wei, Constantin Koumouzelis, Dane Sherburn, Daniel Kappler,  
 641 Daniel Levin, Daniel Levy, David Carr, David Farhi, David Mely, David Robinson, David Sasaki,  
 642 Denny Jin, Dev Valladares, Dimitris Tsipras, Doug Li, Duc Phong Nguyen, Duncan Findlay,  
 643 Edede Oiwoh, Edmund Wong, Ehsan Asdar, Elizabeth Proehl, Elizabeth Yang, Eric Antonow,  
 644 Eric Kramer, Eric Peterson, Eric Sigler, Eric Wallace, Eugene Brevdo, Evan Mays, Farzad Kho-  
 645 rasani, Felipe Petroski Such, Filippo Raso, Francis Zhang, Fred von Lohmann, Freddie Sulit,  
 646 Gabriel Goh, Gene Oden, Geoff Salmon, Giulio Starace, Greg Brockman, Hadi Salman, Haiming  
 647 Bao, Haitang Hu, Hannah Wong, Haoyu Wang, Heather Schmidt, Heather Whitney, Heewoo Jun,  
 Hendrik Kirchner, Henrique Ponde de Oliveira Pinto, Hongyu Ren, Huiwen Chang, Hyung Won  
 Chung, Ian Kivlichan, Ian O’Connell, Ian O’Connell, Ian Osband, Ian Silber, Ian Sohl, Ibrahim  
 Okuyucu, Ikai Lan, Ilya Kostrikov, Ilya Sutskever, Ingmar Kanitscheider, Ishaan Gulrajani, Ja-  
 cob Coxon, Jacob Menick, Jakub Pachocki, James Aung, James Betker, James Crooks, James  
 Lennon, Jamie Kiros, Jan Leike, Jane Park, Jason Kwon, Jason Phang, Jason Teplitz, Jason Wei,  
 Jason Wolfe, Jay Chen, Jeff Harris, Jenia Varavva, Jessica Gan Lee, Jessica Shieh, Ji Lin, Jiahui  
 Yu, Jiayi Weng, Jie Tang, Jieqi Yu, Joanne Jang, Joaquin Quinonero Candela, Joe Beutler, Joe

648 Landers, Joel Parish, Johannes Heidecke, John Schulman, Jonathan Lachman, Jonathan McKay,  
 649 Jonathan Uesato, Jonathan Ward, Jong Wook Kim, Joost Huizinga, Jordan Sitkin, Jos Kraaijeveld,  
 650 Josh Gross, Josh Kaplan, Josh Snyder, Joshua Achiam, Joy Jiao, Joyce Lee, Juntang Zhuang,  
 651 Justyn Harriman, Kai Fricke, Kai Hayashi, Karan Singhal, Katy Shi, Kavin Karthik, Kayla Wood,  
 652 Kendra Rimbach, Kenny Hsu, Kenny Nguyen, Keren Gu-Lemberg, Kevin Button, Kevin Liu, Kiel  
 653 Howe, Krithika Muthukumar, Kyle Luther, Lama Ahmad, Larry Kai, Lauren Itow, Lauren Work-  
 654 man, Leher Pathak, Leo Chen, Li Jing, Lia Guy, Liam Fedus, Liang Zhou, Lien Mamitsuka,  
 655 Lilian Weng, Lindsay McCallum, Lindsey Held, Long Ouyang, Louis Feuvrier, Lu Zhang, Lukas  
 656 Kondraciuk, Lukasz Kaiser, Luke Hewitt, Luke Metz, Lyric Doshi, Mada Afslak, Maddie Simens,  
 657 Madelaine Boyd, Madeleine Thompson, Marat Dukhan, Mark Chen, Mark Gray, Mark Hudnall,  
 658 Marvin Zhang, Marwan Aljubeh, Mateusz Litwin, Matthew Zeng, Max Johnson, Maya Shetty,  
 659 Mayank Gupta, Meghan Shah, Mehmet Yatbaz, Meng Jia Yang, Mengchao Zhong, Mia Glaese,  
 660 Mianna Chen, Michael Janner, Michael Lampe, Michael Petrov, Michael Wu, Michele Wang,  
 661 Michelle Fradin, Michelle Pokrass, Miguel Castro, Miguel Oom Temudo de Castro, Mikhail  
 662 Pavlov, Miles Brundage, Miles Wang, Minal Khan, Mira Murati, Mo Bavarian, Molly Lin, Murat  
 663 Yesildal, Nacho Soto, Natalia Gimelshein, Natalie Cone, Natalie Staudacher, Natalie Summers,  
 664 Natan LaFontaine, Neil Chowdhury, Nick Ryder, Nick Stathas, Nick Turley, Nik Tezak, Niko Fe-  
 665 lix, Nithanth Kudige, Nitish Keskar, Noah Deutsch, Noel Bundick, Nora Puckett, Ofir Nachum,  
 666 Ola Okelola, Oleg Boiko, Oleg Murk, Oliver Jaffe, Olivia Watkins, Olivier Godement, Owen  
 667 Campbell-Moore, Patrick Chao, Paul McMillan, Pavel Belov, Peng Su, Peter Bak, Peter Bakkum,  
 668 Peter Deng, Peter Dolan, Peter Hoeschele, Peter Welinder, Phil Tillet, Philip Pronin, Philippe  
 669 Tillet, Prafulla Dhariwal, Qiming Yuan, Rachel Dias, Rachel Lim, Rahul Arora, Rajan Troll, Ran-  
 670 dall Lin, Rapha Gontijo Lopes, Raul Puri, Reah Miyara, Reimar Leike, Renaud Gaubert, Reza  
 671 Zamani, Ricky Wang, Rob Donnelly, Rob Honsby, Rocky Smith, Rohan Sahai, Rohit Ramchan-  
 672 dani, Romain Huet, Rory Carmichael, Rowan Zellers, Roy Chen, Ruby Chen, Ruslan Nigmat-  
 673 ullin, Ryan Cheu, Saachi Jain, Sam Altman, Sam Schoenholz, Sam Toizer, Samuel Miserendino,  
 674 Sandhini Agarwal, Sara Culver, Scott Ethersmith, Scott Gray, Sean Grove, Sean Metzger, Shamez  
 675 Hermani, Shantanu Jain, Shengjia Zhao, Sherwin Wu, Shino Jomoto, Shirong Wu, Shuaiqi Xia,  
 676 Sonia Phene, Spencer Papay, Srinivas Narayanan, Steve Coffey, Steve Lee, Stewart Hall, Suchir  
 677 Balaji, Tal Broda, Tal Stramer, Tao Xu, Tarun Gogineni, Taya Christianson, Ted Sanders, Tejal  
 678 Patwardhan, Thomas Cunningham, Thomas Degry, Thomas Dimson, Thomas Raoux, Thomas  
 679 Shadwell, Tianhao Zheng, Todd Underwood, Todor Markov, Toki Sherbakov, Tom Rubin, Tom  
 680 Stasi, Tomer Kaftan, Tristan Heywood, Troy Peterson, Tyce Walters, Tyna Eloundou, Valerie Qi,  
 681 Veit Moeller, Vinnie Monaco, Vishal Kuo, Vlad Fomenko, Wayne Chang, Weiyi Zheng, Wenda  
 682 Zhou, Wesam Manassra, Will Sheu, Wojciech Zaremba, Yash Patil, Yilei Qian, Yongjik Kim,  
 683 Youlong Cheng, Yu Zhang, Yuchen He, Yuchen Zhang, Yujia Jin, Yunxing Dai, and Yury Malkov.  
 684 Gpt-4o system card, 2024. URL <https://arxiv.org/abs/2410.21276>.

685 Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong  
 686 Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kel-  
 687 ton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike,  
 688 and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv*  
 689 preprint [arXiv:2203.02155](https://arxiv.org/abs/2203.02155), 2022.

690 Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and trans-  
 691 fer reinforcement learning. In *International Conference on Learning Representations (ICLR)*,  
 692 2016.

693 Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John  
 694 Wiley & Sons, 2014.

695 Nathan Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. In *Proceed-  
 696 ings of the 23rd International Conference on Machine Learning (ICML)*, pp. 729–736. ACM,  
 697 2006.

698 Christopher Rawles, Alice Li, Daniel Rodriguez, Oriana Riva, and Timothy Lillicrap. Android in  
 699 the wild: A large-scale dataset for android device control, 2023. URL <https://arxiv.org/abs/2307.10088>.

700 Christopher Rawles, Sarah Clinckemaillie, Yifan Chang, Jonathan Waltz, Gabrielle Lau, Mary-  
 701 beth Fair, Alice Li, William Bishop, Wei Li, Folawiyo Campbell-Ajala, et al. Androidworld: A

702 dynamic benchmarking environment for autonomous agents. *arXiv preprint arXiv:2405.14573*,  
 703 2024.

704

705 Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog,  
 706 M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang,  
 707 Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from pro-  
 708 gram search with large language models. *Nature*, 625(7995):468–475, 2024a. doi: 10.1038/  
 709 s41586-023-06924-6.

710 Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog,  
 711 M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang,  
 712 Omar Fawzi, et al. Mathematical discoveries from program search with large language models.  
 713 *Nature*, 625(7995):468–475, 2024b.

714 Stéphane Ross, Geoffrey Gordon, and J. Andrew Bagnell. A reduction of imitation learning and  
 715 structured prediction to no-regret online learning. In *Proceedings of the 14th International Con-  
 716 ference on Artificial Intelligence and Statistics (AISTATS)*, pp. 627–635, 2011.

717

718 Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a  
 719 scalable alternative to reinforcement learning. In *Proceedings of the 36th International Con-  
 720 ference on Machine Learning (ICML)*, 2017.

721

722 Silvia Sapora, Gokul Swamy, Chris Lu, Yee Whye Teh, and Jakob Nicolaus Foerster. Evil: Evolution  
 723 strategies for generalisable imitation learning, 2024.

724

725 John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy  
 726 optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

727

728 Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient meth-  
 729 ods for reinforcement learning with function approximation. In *Advances in Neural Information  
 Processing Systems (NeurIPS)*, volume 12, pp. 1057–1063, 1999.

730

731 Gokul Swamy, Sanjiban Choudhury, J Andrew Bagnell, and Steven Wu. Of moments and match-  
 732 ing: A game-theoretic framework for closing the imitation gap. In *International Conference on  
 733 Machine Learning*, pp. 10022–10032. PMLR, 2021.

734

735 Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. *Advances  
 736 in neural information processing systems*, 20, 2007.

737

738 Yee Whye Teh, Victor Bapst, Wojciech M. Czarnecki, John Quan, James Kirkpatrick, Raia Had-  
 739 sell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. In  
 740 *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4499–4509, 2017.

741

742 Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control.  
 In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033,  
 743 2012. doi: 10.1109/IROS.2012.6386109.

744

745 David Venuto, Mohammad Sami Nur Islam, Martin Klissarov, Doina Precup, Sherry Yang, and  
 746 Ankit Anand. Code as reward: Empowering reinforcement learning with VLMs. In Ruslan  
 747 Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and  
 748 Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*,  
 749 volume 235 of *Proceedings of Machine Learning Research*, pp. 49368–49387. PMLR, 21–27 Jul  
 2024a. URL <https://proceedings.mlr.press/v235/venuto24a.html>.

750

751 David Venuto, Sami Nur Islam, Martin Klissarov, Doina Precup, Sherry Yang, and Ankit  
 752 Anand. Code as reward: Empowering reinforcement learning with vlms. *arXiv preprint  
 753 arXiv:2402.04764*, 2024b.

754

755 Yuxiang Wang, Yuchen Lin, Dongfu Jiang, Bill Y. Chen, Xiang Shen, Jidong Zhao, Xiang Yu, Chen  
 Li, Xiao Qin, and Jie Sun. Llm-blender: Ensembling large language models with pairwise ranking  
 and generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational  
 756 Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2023.

756 Aaron Wilson, Alan Fern, and Prasad Tadepalli. Multi-task reinforcement learning: A hierarchical  
 757 bayesian approach. In *Proceedings of the 24th International Conference on Machine Learning*  
 758 (*ICML*), pp. 1015–1022. ACM, 2007.

759  
 760 Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Are-  
 761 nas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to  
 762 rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023.

763 Abhay Zala, Jaemin Cho, Han Lin, Jaehong Yoon, and Mohit Bansal. Envgen: Generating and  
 764 adapting environments via llms for training embodied agents. In *Conference on Language Mod-  
 765 eling (CoLM)*, 2024.

766 William Zankner, Rohan Mehta, Eric Wallace, Jack Fitzsimons, Y. Yang, Alex Mei, Daniel Levy,  
 767 William S. Moses, and Joseph E. Gonzalez. Critique-out-loud reward models. 2024. URL  
 768 <https://arxiv.org/abs/2408.11791>.

770 Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang,  
 771 Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and  
 772 chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

773 Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Lan-  
 774 guage agent tree search unifies reasoning acting and planning in language models. *arXiv preprint  
 775 arXiv:2310.04406*, 2023.

776 Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy  
 777 inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial  
 778 Intelligence (AAAI)*, pp. 1433–1438. AAAI Press, 2008.

## 781 A APPENDIX

### 782 A.1 RELATIONS TO INVERSE REINFORCEMENT LEARNING

783 **Proposition 2.** Suppose  $m(s) = 1$  iff  $s \in \mathcal{S}_g$  else  $m(s) = -1$ , then GRACE optimizes,  
 784  $\min_{\pi} \max_r J(\pi_E, m \circ r) - J(\pi, -m \circ r)$ , which is a variation of Eq. (2)

785 *Proof.* Suppose  $m(s) = 1$  iff  $s \in \mathcal{S}_g$  else  $m(s) = -1$  is a mask over goal states. Then, the fitness  
 786 function from Eq. 3 can be re-written in terms of the policy return akin to Eq. 1:

$$f(r) = \mathbb{E}_{s \sim \mathcal{S}_g}[r(s)] - \mathbb{E}_{s \sim \mathcal{S}_{ng}}[r(s)] \quad (5)$$

$$= \mathbb{E}_{\tau \sim D^+, s \in \tau}[m(s)r(s)] - \mathbb{E}_{\tau \sim D^-, s \in \tau}[-m(s)r(s)] \quad (6)$$

$$= J(\pi_E, m \circ r) - J(\pi, -m \circ r) \quad (7)$$

787 where  $m$  flips the reward value either if the state is non-goal and generated by the expert or it is a  
 788 goal and generated by the learned policy.

789 The operator  $m$  can either be defined in Phase 1 by the LLM, or it can default to  $m(s) = 1$  iff  
 790  $s \in \mathcal{S}_E$  (expert states) or  $m(s) = -1$  iff  $s \in \mathcal{S}_L$  (learner states). Phase 2, the reward refinement  
 791 stage is maximizing  $f$  w.r.t the reward. Phase 3, on the other side, is maximizing the return of  $\pi$ , or  
 792 minimizing its negative. Thus, GRACE attempts to solve:

$$\min_{\pi} \max_r J(\pi_E, m \circ r) - J(\pi, -m \circ r)$$

800  $\square$

### 801 A.2 GOAL IDENTIFICATION

802 Goal identification is the critical first step (Phase 1) of the GRACE framework, where an LLM  
 803 automatically labels states from expert demonstration trajectories ( $\mathcal{D}^+$ ) as either goal states ( $s_g$ ) or  
 804 non-goal states ( $s_{ng}$ ). This process creates the initial dataset that the evolutionary search uses to  
 805 refine the reward functions. We evaluated the effectiveness of this automated approach using gpt-40

(OpenAI et al., 2024), with the results presented in Table 3. The findings show that providing the model with textual representations of states is highly effective, achieving 94% accuracy. In contrast, relying on image-based input alone was significantly less effective, with accuracy dropping to 49%. However, it is likely that models with more comprehensive visual pre-training would be substantially better at identifying goal states from image-only inputs. This is still much better than chance, as the trajectories average around 20 steps. The experiment also tested performance on shuffled trajectories to see if the model relied on temporal order. Accuracy with text input saw a minor drop to 88%, indicating that while the model leverages the sequence of events, it is not entirely dependent on it to identify goal states.

Table 3: Model Accuracy Comparison

Metric	gpt-4o w/		
	Text	Images	Text and Images
Accuracy	$0.94 \pm 0.24$	$0.49 \pm 0.38$	$0.88 \pm 0.34$
Accuracy on Shuffled	$0.88 \pm 0.48$	$0.49 \pm 0.50$	$0.75 \pm 0.43$

In the more complex AndroidControl domain, GRACE showed a remarkable ability not only to identify the goal state within a trajectory but also to refine the task’s textual instruction to accurately reflect the demonstrated behavior. A few examples highlight this robustness:

- **Refining Instructions to Match Behavior:** GRACE resolves ambiguities between an instruction and the corresponding trajectory. For instance, in a trajectory where the user was instructed to "set a timer" but did not start it, GRACE updated the instruction to explicitly include a "don't start the timer" clause. Similarly, when a user was asked to "set an alarm for 9am" but also performed the extra step of naming the alarm, GRACE appended the instruction to include the naming step, ensuring the final instruction precisely matched the expert demonstration.
- **Discarding Irrelevant Trajectories:** The system correctly identifies and filters out trajectories where the user’s actions are inconsistent with the instruction’s domain. When a user was instructed to perform a task in the ‘Clock’ app but completed it in the ‘ClockBuddy’ app, GRACE identified the application mismatch. This allowed the trajectory to be filtered from the dataset for the intended ‘Clock’ app task. A similar process occurred when a user was given a nonsensical instruction like "give me directions for X in the clock app" and then used Google Maps.

864 A.3 ADDITIONAL ONLINE RESULTS  
865

866	Task	PPO	GAIL	GRACE
868	OpenRedDoor	1.00	1.00	1.00
869	GoToObjS4	1.00	1.00	1.00
870	GoToRedBlueBall	0.96	0.40	0.99
871	GoToRedBallGrey	0.97	0.77	0.99
872	Pickup	0.10	0.00	0.09
873	Open	0.30	0.18	0.22
874	OpenRedBlueDoors	1.00	0.96	0.98
875	OpenDoorLoc	0.39	0.40	1.00
876	GoToLocalS8N7	0.64	0.39	0.97
877	GoToDoor	0.74	0.37	0.99
878	SortColors (new)	0.00	0.00	0.00

879 Table 4: **Success rates on additional BabyAI environments.** The performance of our method,  
880 GRACE, is compared against two key baselines: PPO, trained on the ground-truth reward, and  
881 GAIL, trained using 2000 expert trajectories per task. GRACE’s performance is evaluated with 8  
882 expert trajectories per task to demonstrate its high sample efficiency. All values represent the final  
883 success rate at the end of training.884  
885 A.4 EXTENDED DISCUSSION AND FUTURE WORK  
886887 GRACE’s reliance on programmatic reward functions introduces several limitations, particularly  
888 when compared to traditional deep neural network based approaches. These limitations also point  
889 toward promising directions for future research.  
890891 **Input modality** While generating rewards as code offers interpretability and sample efficiency, it  
892 struggles in domains where the reward depends on complex, high-dimensional perceptual inputs.  
893 Code is inherently symbolic and structured, making it less suited for interpreting raw sensory data  
894 like images or audio. For instance, creating a programmatic reward for a task like "navigate to the  
895 object that looks most fragile" is non-trivial, as "fragility" is a nuanced visual concept. NNs, in  
896 contrast, excel at learning features directly from this kind of data. Programmatic rewards can also  
897 be brittle: a small, unforeseen perturbation in the environment that violates a hard-coded assumption  
898 could cause the reward logic to fail completely, whereas NNs often degrade more gracefully.  
899900 **Data Quantity** GRACE demonstrates remarkable performance with very few demonstrations.  
901 This is a strength in data-scarce scenarios. However, it is a limitation when vast amounts of data are  
902 available. Deep IRL methods like GAIL are designed to scale with data and may uncover subtle,  
903 complex patterns from millions of demonstrations that would be difficult to capture in an explicit  
904 program. While GRACE’s evolutionary search benefits from tight feedback on a small dataset, it is  
905 not clear how effectively it could learn from a massive dataset.  
906907 **Failure Cases** Although GRACE is highly sample-efficient, it is not a magic bullet. For example,  
908 in the BabyAI-OpenTwoDoors task, GRACE often proposed a reward that didn’t take into account  
909 the order in which the doors were being opened. Similarly, in the new BabyAI-SortColors task,  
910 it would sometimes return a reward that only accounted for picking up and dropping both objects,  
911 without paying attention to where they were being dropped. While these errors can be easily fixed  
912 by providing a relevant negative trajectory or by treating all learner-generated states as negative  
913 trajectories, they highlight that GRACE can still misinterpret an agent’s true intent based on expert  
914 demonstrations alone.915 **Hybrid Approaches** These limitations can be substantially mitigated by extending the GRACE  
916 framework to incorporate tool use, combining the strengths of both systems. The LLM could be  
917 granted access to a library of pre-trained models (e.g., object detectors, image classifiers, or seg-  
mentation models). The LLM’s task would then shift from writing low-level image processing code

918 to writing high-level logic that calls these tools and reasons over their outputs. A final direction  
919 involves generating hybrid reward functions that are part code and part neural network. The LLM  
920 could define the overall structure, logic, and shaping bonuses in code, but instantiate a small, learn-  
921 able NN module for a specific, difficult-to-program component of the reward. This module could  
922 then be fine-tuned using the available demonstrations, creating a reward function that is both largely  
923 interpretable and capable of handling perceptual nuance. By exploring these hybrid approaches,  
924 future iterations of GRACE could retain the benefits of interpretability and sample efficiency while  
925 overcoming the inherent limitations of purely programmatic solutions in complex, perception-rich  
926 environments.

927

928

929

930

931

932

933

934

935

936

937

938

939

940

941

942

943

944

945

946

947

948

949

950

951

952

953

954

955

956

957

958

959

960

961

962

963

964

965

966

967

968

969

970

971

972 A.5 NEW BABYAI LEVELS  
973974 To evaluate the generalization and reasoning capabilities of GRACE and mitigate concerns of data  
975 contamination from pre-existing benchmarks, we designed three novel BabyAI levels.  
976977 **PlaceBetween** The agent is placed in a single room with three distinct objects (e.g., a red ball, a  
978 green ball, and a blue ball). The instruction requires the agent to pick up a specific target object  
979 and place it on an empty cell that is strictly between the other two anchor objects. Success requires  
980 being on the same row or column as the two anchors, creating a straight line. This task moves  
981 beyond simple navigation, demanding that the agent understand the spatial relationship "between"  
982 and act upon a configuration of three separate entities.  
983984 **OpenMatchingDoor** This level is designed to test indirect object identification and chained inference.  
985 The environment consists of a single room containing one key and multiple doors of different  
986 colors. The instruction is to "open the door matching the key". The agent cannot solve the task by  
987 simply parsing an object and color from the instruction. Instead, it must first locate the key, visually  
988 identify its color, and then find and open the door of the corresponding color. This task assesses the  
989 agent's ability to perform a simple chain of reasoning: find object A, infer a property from it, and  
990 then use that property to identify and interact with target object B.  
991992 **SortColors** The environment consists of two rooms connected by a door, with a red ball in one  
993 room and a blue ball in the other. The instruction is a compound goal: "put the red ball in the  
994 right room and put the blue ball in the left room". To make the task non-trivial, the objects' initial  
995 positions are swapped relative to their goal locations. The agent must therefore execute a sequence  
996 of sub-tasks for each object: pick up the object, navigate to the other room, and drop it. This level  
997 tests the ability to decompose a complex language command and carry out a plan to satisfy multiple,  
998 distinct objectives.  
999  
1000  
1001  
1002  
1003  
1004  
1005  
1006  
1007  
1008  
1009  
1010  
1011  
1012  
1013  
1014  
1015  
1016  
1017  
1018  
1019  
1020  
1021  
1022  
1023  
1024  
1025

1026  
1027

## A.6 HYPERPARAMETERS

1028  
1029

Table 5: Hyperparameters for Training BabyAI with PPO

1030  
1031  
1032  
1033  
1034  
1035  
1036  
1037  
1038  
1039

Parameter	Value
Base Model	llava-onevision-qwen2-0.5b-ov-hf
Gamma	0.999
Learning Rate	3e-5
Entropy Coef	1e-5
Num Envs	10
Num Steps	64
Episode Length	100
PPO Epochs	2
Num Minibatch	6

1040  
1041

Table 6: Hyperparameters for Training AndroidWorld

1042  
1043  
1044  
1045  
1046  
1047  
1048  
1049  
1050  
1051  
1052  
1053  
1054  
1055  
1056

Parameter	Value
Base Model	Qwen2.5-VL-3B-Instruct
LoRA Rank	512
LoRA Alpha	32
LoRA Dropout	0.1
Critic Hidden Size	2048
Critic Depth	4
Gamma	0.999
Learning Rate	3e-5
Entropy Coef	0.0
Num Envs	16
Num Steps	16
Episode Length	20
PPO Epochs	2
Num Minibatch	2

1057  
1058

Table 7: Hyperparameters for GRACE Evolution

1059  
1060  
1061  
1062  
1063  
1064  
1065  
1066  
1067  
1068  
1069  
1070  
1071  
1072  
1073  
1074  
1075  
1076  
1077  
1078  
1079

Parameter	Value
Population Size	20
Elite	4
Num Generations	100
Include expert trajectory chance	0.25
Incorrect state only chance	0.5
Expert state only chance	0.75
Model	gpt-4o

1080  
1081

## A.7 EVOLUTION EXAMPLES

```

1082 1 def _parse_colour_from_text(text: Optional[str]) -> Optional[int]:
1083 2     if text is None:
1084 3         return None
1085 4
1086 5     colour_words: Dict[str, int] = {
1087 6         "red": 0,
1088 7         "green": 1,
1089 8         "blue": 2,
1090 9         "yellow": 3,           "purple": 3,
1091 10        "yellow": 4,
1092 11        "orange": 5,          # keep old mapping
1093 12        "grey": 5,           # alias for the observed colour code in the trajectory
1094 13        "gray": 5,
1095 14    }
1096 15    lower = text.lower()
1097 16    for word, code in colour_words.items():
1098 17        if word in text.lower(): lower:
1099 18            return code
1100 19    return None
1101 20
1102 21 def _parse_goal_type(text: Optional[str]) -> str:
1103 22     if text is None:
1104 23         return "key"
1105 24     txt = text.lower()
1106 25     if "ball" in txt:
1107 26         return "ball"
1108 27     if "box" in txt:
1109 28         return "box"
1110 29     return "key"
1111 30

```

Figure 7: **GRACE iteratively refines the initial BabyAI reward function (iteration 0) to handle unseen entities (iteration 10).** Using execution traces, the agent fixes its color code mistake and adds a new box entity.

```

1110 1 from __future__ import annotations
1111 2
1112 3 import re
1113 4 from typing import Optional, Tuple
1114 5
1115 6 import numpy as np
1116 7
1117 8 COLOR2ID = {
1118 9     "red": 0,
1119 10    "green": 1,
1120 11    "blue": 2,
1121 12    "purple": 3,
1122 13    "yellow": 4,
1123 14    "grey": 5,
1124 15    "gray": 5,           # US spelling
1125 16}
1126 17
1127 18 OBJECT2ID = {
1128 19    "empty": 0,
1129 20    "wall": 1,
1130 21    "floor": 2,
1131 22    "door": 3,
1132 23    "key": 5,
1133 24    "ball": 6,
1134 25    "box": 8,
1135 26    "agent": 10,
1136 27}
1137 28
1138 29 # Map MiniGrid direction codes (stored in the 3-rd channel of the agent cell)
1139 30 # to row/col deltas. Empirically direction 0 points *down/south* in the
1140 31 # provided trajectories.
1141 32 DIR2VEC: dict[int, Tuple[int, int]] = {
1142 33    0: (1, 0),          # south
1143 34    1: (0, 1),          # east

```

```

1134
1135     2: (-1, 0), # north
1136     3: (0, -1), # west
1137 }
1138
1139 def _parse_goal(extra_info: str) -> Tuple[int, Optional[int]]:
1140     """Return *(object_id, colour_id)* parsed from *extra_info*."""
1141     if not extra_info:
1142         raise ValueError("extra_info must specify the target, e.g. 'the red ball'.")
1143
1144     tokens = re.findall(r"[a-zA-Z]+", extra_info.lower())
1145     obj_id: Optional[int] = None
1146     col_id: Optional[int] = None
1147     for tok in tokens:
1148         if obj_id is None and tok in OBJECT2ID:
1149             if tok in COLOR2ID and col_id is None:
1150                 col_id = COLOR2ID[tok]
1151             if tok in OBJECT2ID and obj_id is None:
1152                 obj_id = OBJECT2ID[tok]
1153             if col_id is None and tok in COLOR2ID:
1154                 col_id = COLOR2ID[tok]
1155     if obj_id is None:
1156         raise ValueError(
1157             f"Could not parse target object from extra_info='{extra_info}'."
1158         )
1159     return obj_id, col_id # colour may be None (wild-card)
1160
1161
1162 class Reward:
1163     """Success when definition (single-step, dense reward):
1164     100.0 if the **first** cell in front of the agent is either
1165     on / adjacent (according to the
1166     closest target object (Manhattan distance  $d'$ ), OR
1167     direction stored in the third observation channel) contains a
1168     matching target has disappeared from the observable grid (picked up).
1169
1170     Shaping:  $r = 1 / (1+d)$  with  $d$  the Manhattan distance to the closest
1171     still-visible target, clipped at 0 object.
1172     <1.0 shaping reward  $1/(d+1)$  otherwise.
1173     0.0 if either the agent or a matching target is out of view, not visible.
1174
1175     The implementation is modular so new goal
1176     types can be handled by extending the OBJECT/COLOR lookup tables or by
1177     replacing the *success predicate*.
1178
1179     """
1180
1181     SUCCESS_REWARD = 100.0
1182
1183     def __init__(self, extra_info: Optional[str] = None):
1184         self.tgt_obj_id, self.tgt_col_id, self._target_obj_id, self._target_colour_id = \
1185             _parse_goal(extra_info)
1186
1187     def __call__(self, state: np.ndarray) -> float: # enable direct call
1188         return self.reward_fn(state)
1189
1190     def reward_fn(self, state: np.ndarray) -> float:
1191         """state: (H, W, 3)"""
1192         agent_pos = self._find_agent(state)
1193         if agent_pos is None:
1194             return 0.0
1195
1196         # mask of all target objects still visible
1197         tgt_mask = (state[:, :, 0] == self.tgt_obj_id) & (
1198             state[:, :, 1] == self.tgt_col_id
1199             +
1200             if not tgt_mask.any():
1201                 # object gone -> picked up / carried

```

```

1188
1189100     -return self.SUCCESS_REWARD
1190101
1191102     # distance to the closest visible target
1192103     tgt_positions = np.argwhere(tgt_mask)
1193104     dists = np.abs(tgt_positions - agent_pos).sum(axis=1)
1194105
1195106     target_positions = self._find_targets(state)
1196107     if target_positions.size == 0:
1197108         # No matching target in view -> no shaping.
1198109         return 0.0
1199110
1200111     # -----
1201112     # Success predicate → target must be directly in front of the agent.
1202113     # -----
1203114     if self._is_target_in_front(agent_pos, state):
1204115         return 100.0
1205116
1206117     # -----
1207118     # Shaping: inverse Manhattan distance (< 1.0) to the *nearest* target.
1208119     # -----
1209120     dists = np.abs(target_positions - agent_pos).sum(axis=1)
1210121     min_dist = int(dists.min())
1211122     if min_dist <= 1:
1212123         -return self.SUCCESS_REWARD
1213124
1214125     return 1.0 / (1.0 + min_dist)
1215126
1216127     @staticmethod
1217128     def _find_agent(state: np.ndarray) -> Optional[np.ndarray]:
1218129         """Return (row, col) of """Locate the first agent
1219130         pixel found, in the observation (row, col) or None. *None* if absent."""
1220131         locs = np.argwhere(state[:, :, 0] == OBJECT2ID["agent"])
1221132         if locs.size == 0:
1222133             return None
1223134         return locs[0]
1224135
1225136         def _find_targets(self, state: np.ndarray) -> np.ndarray:
1226137             """Return an (N, 2) array of row/col positions of matching targets."""
1227138             obj_mask = state[:, :, 0] == self._target_obj_id
1228139             if self._target_colour_id is not None:
1229140                 col_mask = state[:, :, 1] == self._target_colour_id
1230141                 mask = obj_mask & col_mask
1231142             else:
1232143                 mask = obj_mask
1233144             return np.argwhere(mask)
1234145
1235146             def _is_target_in_front(self, agent_pos: np.ndarray, state: np.ndarray) -> bool:
1236147                 """Return *True* iff the cell directly in front of the agent matches target."""
1237148                 row, col = agent_pos
1238149                 agent_dir = int(state[row, col, 2])
1239150                 drow, dcol = DIR2VEC.get(agent_dir, (1, 0)) # default to south if unknown
1240151                 f_row, f_col = row + drow, col + dcol
1241152
1242153                 # Out of bounds → cannot be success.
1243154                 if not (0 <= f_row < state.shape[0] and 0 <= f_col < state.shape[1]):
1244155                     return False
1245156
1246157                 # Check object id
1247158                 if state[f_row, f_col, 0] != self._target_obj_id:
1248159                     return False
1249160
1250161                 # Check colour if colour was specified.

```

```
1242 161 |     if (             
1243 162 |         self._target_colour_id is not None             
1244 163 |         and state[f_row, f_col, 1] != self._target_colour_id             
1245 164 |     ):             
1246 165 |         return False             
1247 166 |     return True             
1248 167 |  
1249  
1250           Figure 8: Example of code evolution across many generations.  
1251  
1252  
1253  
1254  
1255  
1256  
1257  
1258  
1259  
1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276  
1277  
1278  
1279  
1280  
1281  
1282  
1283  
1284  
1285  
1286  
1287  
1288  
1289  
1290  
1291  
1292  
1293  
1294  
1295
```

1296 A.8 GENERATED REWARDS  
1297

```

1298 1 # -----
1299 2 #           IMPORTS
1300 3 # -----
1301 4 import json
1302 5 import math
1303 6 import re
1304 7 from typing import Callable, List, Optional, Set, Tuple
1305 8
1306 9 # -----
1307 10 #           GENERIC & NORMALISATION HELPERS
1308 11 # -----
1309 12
1310 13
1311 14 def _contains_any(text: str, keywords) -> bool:
1312 15     text_l = text.lower()
1313 16     return any(k.lower() in text_l for k in keywords)
1314 17
1315 18
1316 19 def _has_stopwatch(text: str) -> bool:
1317 20     t = text.lower()
1318 21     return any(p in t for p in ("stopwatch", "stop watch", "stop-watch"))
1319 22
1320 23
1321 24 # ----- Tab-selection helpers -----
1322 25
1323 26
1324 27 def _tab_selected(state: str, label: str) -> bool:
1325 28     pattern = (
1326 29         rf'^(content_description|text)\s*:\s*"(label)"[^"\n]*?"is_selected"\s*:\s*true'
1327 30     )
1328 31     return bool(re.search(pattern, state, re.I))
1329 32
1330 33
1331 34 def _alarm_tab_selected(state: str) -> bool:
1332 35     return _tab_selected(state, "Alarm") or _tab_selected(state, "Alarms")
1333 36
1334 37
1335 38 def _timer_tab_selected(state: str) -> bool:
1336 39     return _tab_selected(state, "Timer")
1337 40
1338 41
1339 42 def _stopwatch_tab_selected(state: str) -> bool:
1340 43     return _tab_selected(state, "Stopwatch")
1341 44
1342 45
1343 46 def _clock_tab_selected(state: str) -> bool:
1344 47     return _tab_selected(state, "Clock")
1345 48
1346 49
1347 50 # ----- Text normalisation helper -----
1348 51
1349 52
1350 53 def _normalize_time_text(txt: str) -> str:
1351 54     txt2 = txt.replace(";", ":")
1352 55     txt2 = re.sub(r"\b([ap])\s*(?:\.\.|\.m|m)\b", r"\1m", txt2, flags=re.I)
1353 56     return txt2
1354 57
1355 58
1356 59 # -----
1357 60 #           TIMER / DURATION PARSING
1358 61 # -----
1359 62
1360 63
1361 64 def _parse_requested_time(text: str) -> int:
1362 65     text = text.replace("-", " ")
1363 66     hours = minutes = seconds = 0
1364 67     for patt, mult in (
1365 68         (r"(\d+)\s*hour", 3600),
1366 69         (r"(\d+)\s*minute", 60),
1367 70         (r"(\d+)\s*second", 1),
1368 71     ):
1369         m = re.search(patt, text, re.I)
1370         if m:
1371             val = int(m.group(1)) * mult
1372             if mult == 3600:
1373                 hours = val // 3600
1374             elif mult == 60:
1375                 minutes = val // 60
1376             else:
1377                 pass
1378
1379

```

```

1350
1351     seconds = val
1352     if hours == minutes == seconds == 0:
1353         m = re.search(r"(\d+)\s*-\?\s*min", text, re.I)
1354         if m:
1355             minutes = int(m.group(1))
1356         else:
1357             m = re.search(r"(\d+)", text)
1358             if m:
1359                 minutes = int(m.group(1))
1360             total = hours * 3600 + minutes * 60 + seconds
1361             return total if total > 0 else 60
1362
1363     # -----
1364     # ADDITIONAL HELPERS
1365     # -----
1366
1367     def _parse_adjust_timer_amount(instr: str) -> Optional[int]:
1368         instr_l = instr.lower()
1369         verb = r"(?:add|increase|extend|plus|up|extra|more|additional)"
1370         unit = r"(hours|minutes|seconds)"
1371         pat1 = re.compile(rf"^{verb}\s+(\d+)\s*(?:more\s+)?{unit}")
1372         pat2 = re.compile(rf"by\s+(\d+)\s*{unit}")
1373         seconds: List[int] = []
1374         for pat in (pat1, pat2):
1375             for m in pat.finditer(instr_l):
1376                 num = int(m.group(1))
1377                 u = m.group(2)
1378                 if u.startswith("hour"):
1379                     seconds.append(num * 3600)
1380                 elif u.startswith("minute"):
1381                     seconds.append(num * 60)
1382                 else:
1383                     seconds.append(num)
1384         if seconds:
1385             return max(1, min(seconds))
1386         return None
1387
1388     def _parse_alarm_time(instr: str) -> Tuple[int, int, Optional[str]]:
1389         instr_n = _normalize_time_text(instr)
1390         instr_l = instr_n.lower()
1391         m = re.search(r"(\d{1,2})\s*[:.]\s*(\d{2})\s*(am|pm)?", instr_l)
1392         if m:
1393             h, minute, ap = int(m.group(1)), int(m.group(2)), m.group(3)
1394         else:
1395             m = re.search(r"\b(\d{1,2})\s*(am|pm)\b", instr_l)
1396             if m:
1397                 h, minute, ap = int(m.group(1)), 0, m.group(2)
1398             else:
1399                 return 7, 0, "am"
1400         if ap:
1401             ap = ap.lower()
1402             if ap == "pm" and h != 12:
1403                 h += 12
1404             if ap == "am" and h == 12:
1405                 h = 0
1406         return h % 24, minute, ap
1407
1408     def _extract_timer_components(state: str) -> Optional[Tuple[int, int, int]]:
1409         m = re.search(r"(\d+)\s*minutes?\s*(\d+)\s*seconds?", state, re.IGNORECASE)
1410         if m:
1411             minutes = int(m.group(1))
1412             seconds = int(m.group(2))
1413             return (0, minutes, seconds)
1414
1415         m = re.search(r"(\d+)h\s*(\d+)m\s*(\d+)s", state, re.IGNORECASE)
1416         if m:
1417             hours = int(m.group(1))
1418             minutes = int(m.group(2))
1419             seconds = int(m.group(3))
1420             return (hours, minutes, seconds)
1421
1422     # Case 3: "MM:SS" format, ensuring it's not part of a timestamp (like 12:30 PM)
1423     for mm_match in re.finditer(r"(\d{1,2}):(\d{2})(?!s*[AaPp][Mm])", state):
1424         mm, ss = int(mm_match.group(1)), int(mm_match.group(2))
1425         if not (0 <= ss < 60):
1426             continue
1427         context = state[mm_match.end() : mm_match.end() + 80].lower()

```

```

1404
1405     if "minute" in context or "timer" in context or "remaining" in context:
1406         return (0, mm, ss)
1407
1408     if not _timer_tab_selected(state):
1409         return None
1410
1411     tokens = re.findall(r'\"text\"\s*:\s*\"([^\"]+)\"', state)
1412     tokens = [t.strip() for t in tokens]
1413
1414     for i in range(len(tokens) - 4):
1415         if (
1416             re.fullmatch(r"\d{1,2}", tokens[i])
1417             and tokens[i + 1] == ":" and re.fullmatch(r"\d{2}", tokens[i + 2])
1418             and tokens[i + 3] == ":" and re.fullmatch(r"\d{2}", tokens[i + 4])
1419         ):
1420             h = int(tokens[i])
1421             m_val = int(tokens[i + 2])
1422             s = int(tokens[i + 4])
1423             if 0 <= m_val < 60 and 0 <= s < 60:
1424                 return (h, m_val, s)
1425
1426     for i in range(len(tokens) - 2):
1427         if (
1428             re.fullmatch(r"\d{1,2}", tokens[i])
1429             and tokens[i + 1] == ":" and re.fullmatch(r"\d{2}", tokens[i + 2])
1430         ):
1431             m_val = int(tokens[i])
1432             s_val = int(tokens[i + 2])
1433             if 0 <= s_val < 60:
1434                 return (0, m_val, s_val)
1435
1436     return None
1437
1438 def _extract_timer_value(state: str) -> int:
1439     timer_components = _extract_timer_components(state)
1440     if timer_components:
1441         hh, mm, ss = timer_components
1442         return int(hh) * 3600 + int(mm) * 60 + int(ss)
1443     else:
1444         return None
1445
1446 # --- UI helpers -----
1447
1448 def _button_visible(state: str, label: str) -> bool:
1449     return bool(
1450         re.search(rf'\"{content_description|text}\\"\s*:\s*\"{label}\\"", state, re.I)
1451     )
1452
1453 def _timer_screen_visible(state: str) -> bool:
1454     if _timer_tab_selected(state):
1455         return True
1456     s = state.lower()
1457     return "remaining" in s or "minutes timer" in s
1458
1459 def _is_timer_running(state: str) -> bool:
1460     return _button_visible(state, "Pause")
1461
1462 def _timer_keypad_mode(state: str) -> bool:
1463     return bool(re.search(r"\b\d{1,2}h\s*\d{1,2}m\s*\d{1,2}s\b", state))
1464
1465 def _is_timer_paused(state: str) -> bool:
1466     if _timer_keypad_mode(state):
1467         return False
1468     if _button_visible(state, "Start") and not _button_visible(state, "Pause"):
1469         return True
1470     if not _timer_screen_visible(state):
1471         return False
1472     s = state.lower()
1473     return "timer paused" in s or ("paused" in s and "timer" in s)
1474
1475 def _timer_keypad_zero(state: str) -> bool:
1476     if not all(
1477

```

```
1458 242     re.search(rf"\"text\"\s*:\s*{lbl}\"", state, re.I)
1459 243     for lbl in ("hour", "min", "sec")
1460 244   ):
1461 245     return False
1462 246   return len(re.findall(r'"text"\s*:\s*0{2}"', state)) >= 3
1463 248
1464 249 def _timer_deleted(state: str) -> bool:
1465 250     s = state.lower()
1466 251     if "no timers" in s:
1467 252       return True
1468 253     val = _extract_timer_value(state)
1469 254     if val == 0 and not _is_timer_running(state):
1470 255       return True
1471 256   return _timer_keypad_zero(state)
1472 257
1473 258
1474 259 def _stopwatch_running(state: str) -> bool:
1475 260   return (
1476 261     _button_visible(state, "Pause")
1477 262     or _button_visible(state, "Stop")
1478 263     or "stopwatch running" in state.lower()
1479 264 )
1480 265
1481 266
1482 267 def _stopwatch_time_zero(state: str) -> bool:
1483 268   if re.search(r"\b0{1,2}(?:0{2}){1,3}\b(?!:\d{2})", state):
1484 269     return True
1485 270   nums = re.findall(r'"text"\s*:\s*(\d{2})"', state)
1486 271   return bool(nums) and all(n == "00" for n in nums)
1487 272
1488 273
1489 274 def _timer_paused_notification(state: str) -> bool:
1490 275   return bool(
1491 276     re.search(r"the\s+clock\s+notification:\s+timer", state, re.I)
1492 277     or re.search(r"timer\s+paused", state, re.I)
1493 278 )
1494 279
1495 280
1496 281 def _alarm_context_present(state: str) -> bool:
1497 282   return _alarm_tab_selected(state) or bool(re.search(r"\balarm\b", state, re.I))
1498 283
1499 284
1500 285 def _parse_new_timer_label(instr_l: str) -> str:
1501 286   for kw in (" as ", " named ", " called ", " name "):
1502 287     if kw in instr_l:
1503 288       part = instr_l.split(kw, 1)[1]
1504 289       part = re.split(r"\b[.,;]\b|\bfor\b|\btimer\b", part, flags=re.I)[0]
1505 290     return part.strip()
1506 291   return ""
1507 292
1508 293
1509 294 def _timer_label_present(state: str, label: str) -> bool:
1510 295   if not label:
1511 296     return False
1512 297   return bool(
1513 298     re.search(
1514 299       rf'\"(text|content_description)\"\s*:\s*{re.escape(label)}\"', state, re.I)
1515 300   )
1516 301
1517 302
1518 303
1519 304 def _safe_json_dumps(obj) -> str:
1520 305   try:
1521 306     return json.dumps(obj, ensure_ascii=False)
1522 307   except Exception:
1523 308     return json.dumps({"error": "debug-serialization failed"})
1524 309
1525 310
1526 311 def _any_alarm_present(state: str) -> bool:
1527 312   sl = state.lower()
1528 313   if "alarm set" in sl:
1529 314     return True
1530 315   if _alarm_tab_selected(state) and re.search(r"\b\d{1,2}:\d{2}\s*(?:am|pm)\b", sl):
1531 316     return True
1532 317   return False
1533 318
1534 319
1535 320 def _is_alarm_deleted(state: str) -> bool:
1536 321   s = state.lower()
1537 322   return any(
```

```

1512
1513     re.search(p, s)
1514     for p in (
1515         r"alarm (deleted|removed|dismissed)",
1516         r"\bno (active )?alarms?\b",
1517         r"tap here to create an alarm",
1518         r"alarm deleted",
1519     )
1520
1521
1522     def _snooze_completed(state: str) -> bool:
1523         s_low = state.lower()
1524         if "alarm snoozed" in s_low:
1525             return True
1526         if re.search(r"snoozed\s+for\s+\d+", s_low):
1527             return True
1528         if re.search(r"\bsnooz(ing|ed)\b", s_low):
1529             return True
1530         if "select snooze duration" in s_low:
1531             return True
1532         return False
1533
1534     def _rename_dialog_open(state: str) -> bool:
1535         s = state.lower()
1536         if "enter timer name" in s:
1537             return True
1538         has_buttons = re.search(r"\btext\s*:\s*(ok|cancel)\b", state, re.I)
1539         has_edit = re.search(r"\bis_editable\s*:\s*true\b", state, re.I)
1540         return bool(has_buttons and has_edit)
1541
1542
1543     def _detect_alarm_time(state: str) -> bool:
1544         return bool(re.search(r"\b\d{1,2}\s*:\s*\d{2}\s*(?:\s*[ap]m)\b", state, re.I))
1545
1546
1547     def _selected_weekdays(state: str) -> Set[str]:
1548         selected = set()
1549         for key, full, abbrev in (
1550             ("sunday", "Sunday", "S"),
1551             ("monday", "Monday", "M"),
1552             ("tuesday", "Tuesday", "T"),
1553             ("wednesday", "Wednesday", "W"),
1554             ("thursday", "Thursday", "T"),
1555             ("friday", "Friday", "F"),
1556             ("saturday", "Saturday", "S"),
1557         ):
1558             patt = rf'("content_description"|"text")\s*:\s*"(?:{full}|{abbrev})"[^\n]*?(?'
1559             if "is_selected"|"is_checked")\s*:\s*true'
1560             if re.search(patt, state, re.I):
1561                 selected.add(key)
1562
1563     return selected
1564
1565
1566     def _alarm_time_present(state: str, hour24: int, minute: int, ap: Optional[str]):
1567         s = state.lower().replace("\u200a", "")
1568         h12 = hour24 % 12 or 12
1569         patterns = [rf"\b0*\{h12}\:{minute:02d}\s*(?:am|pm)\b"]
1570         if minute == 0:
1571             patterns.append(rf"\b0*\{h12}\s*(?:am|pm)\b")
1572         patterns.append(rf"\b0*\{hour24}\:{minute:02d}\b")
1573         for p in patterns:
1574             if re.search(p, s):
1575                 if ap and not re.search(rf"\b{ap}\b", s):
1576                     continue
1577                 return True
1578         return False
1579
1580
1581     # ----- NEW HELPER -----
1582
1583     def _day_toggle_buttons_visible(state: str) -> bool:
1584         """Detect if the row of weekday toggle buttons is visible."""
1585         matches = re.findall(r"\btext\s*:\s*(S|M|T|W|F)\b", state)
1586         unique = set(matches)
1587         return len(matches) >= 5 and len(unique.intersection({"M", "T", "W", "F"})) >= 3
1588
1589
1590     # -----
1591     # REWARD CLASS

```

```

1566
1567 # -----
1568
1569 class Reward:
1570     """Dense reward function for Google Clock tasks."""
1571
1572     _SHAPING_INC = 0.3
1573     _ADJ_INC_THRESHOLD = 10
1574
1575     # -----
1576     # INIT
1577     # -----
1578     def __init__(self, extra_info: Optional[str] = None):
1579         self.raw_instr: str = extra_info or ""
1580         self.instruction: str = self.raw_instr.lower()
1581         self.instruction_norm_full = _normalize_time_text(self.raw_instr)
1582         self.instruction_norm = self.instruction_norm_full.lower()
1583
1584         # Task detection
1585         self.task_type = self._infer_task()
1586
1587         # Stopwatch flags
1588         self.restart_mode = False
1589         self._reset_seen = False
1590
1591         # Goal parsing / bookkeeping
1592         self.goal_seconds = 0
1593         self.goal_label = ""
1594         self.goal_hour24 = 0
1595         self.goal_minute = 0
1596         self.goal_hms = (0, 0, 0)
1597         self.goal_ap: Optional[str] = None
1598         self.city_keyword = ""
1599         self.city_keywords: List[str] = []
1600         self.recurrence_days: Set[str] = set()
1601         self.alarm_any_time = False
1602
1603         # Timer-adjust bookkeeping
1604         self.initial_timer_val: Optional[int] = None
1605         self.prev_timer_val: Optional[int] = None
1606         self.max_timer_val: Optional[int] = None
1607         self.increments = 0
1608         self.needed_increments = 0
1609         self._countdown_seen = False
1610
1611         # Alarm creation flag
1612         self._alarm_creation_seen = False
1613
1614         # delete-alarm bookkeeping
1615         self._alarm_present_ever = False
1616
1617         # adjust-alarm bookkeeping
1618         self.orig_hour24 = 0
1619         self.orig_minute = 0
1620         self._orig_seen = False
1621
1622         # pause-timer stability tracking
1623         self._prev_timer_val_for_pause: Optional[int] = None
1624         self._same_val_steps: int = 0
1625
1626         # snooze-specific
1627         self._snooze_dialog_seen = False
1628
1629         # Generic bookkeeping
1630         self.goal_achieved = False
1631         self._best_level = 0
1632         self._t = 0
1633         self._confirm_goal_seen = False
1634
1635         # Map tasks to progress-functions
1636         self._progress_fns: dict[str, Callable[[str], int]] = {
1637             "reset_stopwatch": self._pl_reset_stopwatch,
1638             "restart_stopwatch": self._pl_restart_stopwatch,
1639             "start_stopwatch": self._pl_start_stopwatch,
1640             "pause_stopwatch": self._pl_pause_stopwatch,
1641             "pause_timer": self._pl_pause_timer,
1642             "delete_timer": self._pl_delete_timer,
1643             "delete_alarm": self._pl_delete_alarm,
1644             "add_city": self._pl_add_city,
1645             "set_alarm": self._pl_set_alarm,
1646             "adjust_alarm": self._pl_adjust_alarm,
1647         }

```

```

1620
1621     "rename_timer": self._pl_rename_timer,
1622 }
1623
1624     # Goal-specific parsing / bookkeeping
1625     if self.task_type == "set_timer" or self.task_type == "run_timer":
1626         self.goal_seconds = _parse_requested_time(self.instruction)
1627         h = self.goal_seconds // 3600
1628         rem = self.goal_seconds % 3600
1629         m = rem // 60
1630         s = rem % 60
1631         self.goal_hms = (h, m, s)
1632     if self.task_type == "adjust_timer":
1633         inc_secs = _parse_adjust_timer_amount(
1634             self.instruction_norm_full
1635         ) or _parse_requested_time(self.instruction)
1636         self.goal_seconds = max(1, inc_secs)
1637         self.needed_increments = max(1, math.ceil(self.goal_seconds / 60))
1638     if self.task_type == "rename_timer":
1639         self.goal_seconds = _parse_requested_time(self.instruction)
1640         self.goal_label = _parse_new_timer_label(self.instruction)
1641     if self.task_type == "set_alarm":
1642         explicit = re.search(
1643             r"\d{1,2}(:\d{2})?\s*(am|pm)", self.instruction_norm_full, re.I
1644         )
1645         if explicit:
1646             self.alarm_any_time = False
1647             self._parse_alarm_goal_time()
1648         else:
1649             self.alarm_any_time = True
1650             self.recurrence_days = self._parse_recurrence_days(self.instruction_norm)
1651     if self.task_type == "adjust_alarm":
1652         self.goal_hour24, self.goal_minute = self._parse_adjusted_alarm()
1653         self.goal_ap = None
1654         self.orig_hour24, self.orig_minute, _ = _parse_alarm_time(
1655             self.instruction_norm_full
1656         )
1657     if self.task_type == "add_city":
1658         self.city_keyword = self._parse_city_name(self.instruction) or "italy"
1659         self.city_keywords = [self.city_keyword]
1660         first = self.city_keyword.split()[0] if self.city_keyword else ""
1661         if first and first not in self.city_keywords:
1662             self.city_keywords.append(first)
1663     if self.task_type == "reset_stopwatch":
1664         if re.search(r"\brestart\b", self.instruction) or re.search(
1665             r"start\s+(?:over|again)", self.instruction
1666         ):
1667             self.restart_mode = True
1668
1669     # -----
1670     # PUBLIC API
1671     # -----
1672     def reward_fn(self, state: str) -> float:
1673         self._t += 1
1674         if self.task_type == "set_alarm":
1675             self._update_alarm_creation_seen(state)
1676         if self.goal_achieved:
1677             return 100.0
1678         if self.task_type in self._progress_fns:
1679             return self._reward_from_progress(self._progress_fns[self.task_type], state)
1680         if self.task_type == "set_timer" or self.task_type == "run_timer":
1681             return self._reward_timer(state, self.task_type == "set_timer")
1682         if self.task_type == "adjust_timer":
1683             return self._reward_adjust_timer(state)
1684         if self.task_type == "snooze_alarm":
1685             return self._reward_snooze(state)
1686         return 0.0
1687
1688     def debug_fn(self, state: str) -> str:
1689         dbg = {
1690             "step": self._t,
1691             "task_type": self.task_type,
1692             "goal_achieved": self.goal_achieved,
1693             "best_level": self._best_level,
1694         }
1695         if self.task_type in {"set_timer", "run_timer", "adjust_timer"}:
1696             dbg.update(
1697                 {
1698                     "goal_seconds": self.goal_seconds,
1699                     "increments": self.increments,
1700                     "countdown_seen": self._countdown_seen,
1701                 }
1702             )
1703
1704

```

```

1674
1675     )
1676     if self.task_type == "rename_timer":
1677         dbg["goal_label"] = self.goal_label
1678     if self.task_type == "snooze_alarm":
1679         dbg["dialog_seen"] = self._snooze_dialog_seen
1680     return _safe_json_dumps(dbg)
1681
1682     # -----
1683     # TASK INFERENCE
1684     # -----
1685     def _infer_task(self) -> str:
1686         instr = self.instruction
1687         has_sw = _has_stopwatch(instr)
1688
1689         if has_sw and _contains_any(instr, ["pause", "stop"]):
1690             return "pause_stopwatch"
1691         elif has_sw and _contains_any(
1692             instr, ["restart", "start over", "start again", "begin again"]
1693         ):
1694             return "restart_stopwatch"
1695         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
1696             return "reset_stopwatch"
1697         if has_sw:
1698             return "start_stopwatch"
1699
1700         if (
1701             (re.search(r"\btime\b", instr) or "clock" in instr)
1702             and re.search(r"\bin\s+\w+", instr)
1703             and not _contains_any(instr, ["timer", "alarm"])
1704         ):
1705             return "add_city"
1706
1707         if "timer" in instr:
1708             if _contains_any(instr, ["delete", "remove", "clear"]):
1709                 return "delete_timer"
1710             if _contains_any(instr, ["pause", "stop", "cancel"]):
1711                 return "pause_timer"
1712             if _contains_any(instr, ["rename", "name", "called", "label"]):
1713                 return "rename_timer"
1714             if re.search(
1715                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
1716             ):
1717                 dont_start_req = bool(
1718                     re.search(
1719                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
1720                         r"|(?:\bw+out|s+starting\b) "
1721                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
1722                         r"|(?:\bleave|s+it\s+paused\b) "
1723                         r"|(?:\bkeep|s+it\s+paused\b) ",
1724                         instr,
1725                     )
1726                 )
1727                 if dont_start_req:
1728                     return "set_timer"
1729                 else:
1730                     return "run_timer"
1731             if _contains_any(instr, ["increase", "extend", "more", "up"]):
1732                 return "adjust_timer"
1733             if re.search(
1734                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
1735                 instr,
1736             ):
1737                 return "adjust_timer"
1738             return "run_timer"
1739
1740         if "snooze" in instr:
1741             return "snooze_alarm"
1742         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
1743             return "delete_alarm"
1744         if "alarm" in instr and _contains_any(
1745             instr,
1746             [
1747                 "delay",
1748                 "resched",
1749                 "push",
1750                 "move",
1751                 "change",
1752                 "shift",
1753                 "defer",
1754                 "later",
1755                 "increase",
1756             ],
1757         ):
1758             return "snooze_alarm"
1759
1760     return _infer_task
1761
1762     # -----
1763     # DIALOGUE
1764     # -----
1765     def _infer_dialogue(self) -> str:
1766         instr = self.instruction
1767         has_sw = _has_stopwatch(instr)
1768
1769         if has_sw and _contains_any(instr, ["stop", "cancel"]):
1770             return "stop_dialogue"
1771         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
1772             return "reset_dialogue"
1773
1774         if (
1775             (re.search(r"\btime\b", instr) or "clock" in instr)
1776             and re.search(r"\bin\s+\w+", instr)
1777             and not _contains_any(instr, ["timer", "alarm"])
1778         ):
1779             return "add_dialogue"
1780
1781         if "timer" in instr:
1782             if _contains_any(instr, ["delete", "remove", "clear"]):
1783                 return "delete_dialogue"
1784             if _contains_any(instr, ["pause", "stop", "cancel"]):
1785                 return "pause_dialogue"
1786             if _contains_any(instr, ["rename", "name", "called", "label"]):
1787                 return "rename_dialogue"
1788             if re.search(
1789                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
1790             ):
1791                 dont_start_req = bool(
1792                     re.search(
1793                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
1794                         r"|(?:\bw+out|s+starting\b) "
1795                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
1796                         r"|(?:\bleave|s+it\s+paused\b) "
1797                         r"|(?:\bkeep|s+it\s+paused\b) ",
1798                         instr,
1799                     )
1800                 )
1801                 if dont_start_req:
1802                     return "set_dialogue"
1803                 else:
1804                     return "run_dialogue"
1805             if _contains_any(instr, ["increase", "extend", "more", "up"]):
1806                 return "adjust_dialogue"
1807             if re.search(
1808                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
1809                 instr,
1810             ):
1811                 return "adjust_dialogue"
1812             return "run_dialogue"
1813
1814         if "snooze" in instr:
1815             return "snooze_dialogue"
1816
1817         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
1818             return "delete_dialogue"
1819
1820         if "alarm" in instr and _contains_any(
1821             instr,
1822             [
1823                 "delay",
1824                 "resched",
1825                 "push",
1826                 "move",
1827                 "change",
1828                 "shift",
1829                 "defer",
1830                 "later",
1831                 "increase",
1832             ],
1833         ):
1834             return "snooze_dialogue"
1835
1836     return _infer_dialogue
1837
1838     # -----
1839     # DIALOGUE
1840     # -----
1841     def _infer_dialogue(self) -> str:
1842         instr = self.instruction
1843         has_sw = _has_stopwatch(instr)
1844
1845         if has_sw and _contains_any(instr, ["stop", "cancel"]):
1846             return "stop_dialogue"
1847         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
1848             return "reset_dialogue"
1849
1850         if (
1851             (re.search(r"\btime\b", instr) or "clock" in instr)
1852             and re.search(r"\bin\s+\w+", instr)
1853             and not _contains_any(instr, ["timer", "alarm"])
1854         ):
1855             return "add_dialogue"
1856
1857         if "timer" in instr:
1858             if _contains_any(instr, ["delete", "remove", "clear"]):
1859                 return "delete_dialogue"
1860             if _contains_any(instr, ["pause", "stop", "cancel"]):
1861                 return "pause_dialogue"
1862             if _contains_any(instr, ["rename", "name", "called", "label"]):
1863                 return "rename_dialogue"
1864             if re.search(
1865                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
1866             ):
1867                 dont_start_req = bool(
1868                     re.search(
1869                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
1870                         r"|(?:\bw+out|s+starting\b) "
1871                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
1872                         r"|(?:\bleave|s+it\s+paused\b) "
1873                         r"|(?:\bkeep|s+it\s+paused\b) ",
1874                         instr,
1875                     )
1876                 )
1877                 if dont_start_req:
1878                     return "set_dialogue"
1879                 else:
1880                     return "run_dialogue"
1881             if _contains_any(instr, ["increase", "extend", "more", "up"]):
1882                 return "adjust_dialogue"
1883             if re.search(
1884                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
1885                 instr,
1886             ):
1887                 return "adjust_dialogue"
1888             return "run_dialogue"
1889
1890         if "snooze" in instr:
1891             return "snooze_dialogue"
1892
1893         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
1894             return "delete_dialogue"
1895
1896         if "alarm" in instr and _contains_any(
1897             instr,
1898             [
1899                 "delay",
1900                 "resched",
1901                 "push",
1902                 "move",
1903                 "change",
1904                 "shift",
1905                 "defer",
1906                 "later",
1907                 "increase",
1908             ],
1909         ):
1910             return "snooze_dialogue"
1911
1912     return _infer_dialogue
1913
1914     # -----
1915     # DIALOGUE
1916     # -----
1917     def _infer_dialogue(self) -> str:
1918         instr = self.instruction
1919         has_sw = _has_stopwatch(instr)
1920
1921         if has_sw and _contains_any(instr, ["stop", "cancel"]):
1922             return "stop_dialogue"
1923         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
1924             return "reset_dialogue"
1925
1926         if (
1927             (re.search(r"\btime\b", instr) or "clock" in instr)
1928             and re.search(r"\bin\s+\w+", instr)
1929             and not _contains_any(instr, ["timer", "alarm"])
1930         ):
1931             return "add_dialogue"
1932
1933         if "timer" in instr:
1934             if _contains_any(instr, ["delete", "remove", "clear"]):
1935                 return "delete_dialogue"
1936             if _contains_any(instr, ["pause", "stop", "cancel"]):
1937                 return "pause_dialogue"
1938             if _contains_any(instr, ["rename", "name", "called", "label"]):
1939                 return "rename_dialogue"
1940             if re.search(
1941                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
1942             ):
1943                 dont_start_req = bool(
1944                     re.search(
1945                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
1946                         r"|(?:\bw+out|s+starting\b) "
1947                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
1948                         r"|(?:\bleave|s+it\s+paused\b) "
1949                         r"|(?:\bkeep|s+it\s+paused\b) ",
1950                         instr,
1951                     )
1952                 )
1953                 if dont_start_req:
1954                     return "set_dialogue"
1955                 else:
1956                     return "run_dialogue"
1957             if _contains_any(instr, ["increase", "extend", "more", "up"]):
1958                 return "adjust_dialogue"
1959             if re.search(
1960                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
1961                 instr,
1962             ):
1963                 return "adjust_dialogue"
1964             return "run_dialogue"
1965
1966         if "snooze" in instr:
1967             return "snooze_dialogue"
1968
1969         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
1970             return "delete_dialogue"
1971
1972         if "alarm" in instr and _contains_any(
1973             instr,
1974             [
1975                 "delay",
1976                 "resched",
1977                 "push",
1978                 "move",
1979                 "change",
1980                 "shift",
1981                 "defer",
1982                 "later",
1983                 "increase",
1984             ],
1985         ):
1986             return "snooze_dialogue"
1987
1988     return _infer_dialogue
1989
1990     # -----
1991     # DIALOGUE
1992     # -----
1993     def _infer_dialogue(self) -> str:
1994         instr = self.instruction
1995         has_sw = _has_stopwatch(instr)
1996
1997         if has_sw and _contains_any(instr, ["stop", "cancel"]):
1998             return "stop_dialogue"
1999         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2000             return "reset_dialogue"
2001
2002         if (
2003             (re.search(r"\btime\b", instr) or "clock" in instr)
2004             and re.search(r"\bin\s+\w+", instr)
2005             and not _contains_any(instr, ["timer", "alarm"])
2006         ):
2007             return "add_dialogue"
2008
2009         if "timer" in instr:
2010             if _contains_any(instr, ["delete", "remove", "clear"]):
2011                 return "delete_dialogue"
2012             if _contains_any(instr, ["pause", "stop", "cancel"]):
2013                 return "pause_dialogue"
2014             if _contains_any(instr, ["rename", "name", "called", "label"]):
2015                 return "rename_dialogue"
2016             if re.search(
2017                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2018             ):
2019                 dont_start_req = bool(
2020                     re.search(
2021                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2022                         r"|(?:\bw+out|s+starting\b) "
2023                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2024                         r"|(?:\bleave|s+it\s+paused\b) "
2025                         r"|(?:\bkeep|s+it\s+paused\b) ",
2026                         instr,
2027                     )
2028                 )
2029                 if dont_start_req:
2030                     return "set_dialogue"
2031                 else:
2032                     return "run_dialogue"
2033             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2034                 return "adjust_dialogue"
2035             if re.search(
2036                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2037                 instr,
2038             ):
2039                 return "adjust_dialogue"
2040             return "run_dialogue"
2041
2042         if "snooze" in instr:
2043             return "snooze_dialogue"
2044
2045         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2046             return "delete_dialogue"
2047
2048         if "alarm" in instr and _contains_any(
2049             instr,
2050             [
2051                 "delay",
2052                 "resched",
2053                 "push",
2054                 "move",
2055                 "change",
2056                 "shift",
2057                 "defer",
2058                 "later",
2059                 "increase",
2060             ],
2061         ):
2062             return "snooze_dialogue"
2063
2064     return _infer_dialogue
2065
2066     # -----
2067     # DIALOGUE
2068     # -----
2069     def _infer_dialogue(self) -> str:
2070         instr = self.instruction
2071         has_sw = _has_stopwatch(instr)
2072
2073         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2074             return "stop_dialogue"
2075         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2076             return "reset_dialogue"
2077
2078         if (
2079             (re.search(r"\btime\b", instr) or "clock" in instr)
2080             and re.search(r"\bin\s+\w+", instr)
2081             and not _contains_any(instr, ["timer", "alarm"])
2082         ):
2083             return "add_dialogue"
2084
2085         if "timer" in instr:
2086             if _contains_any(instr, ["delete", "remove", "clear"]):
2087                 return "delete_dialogue"
2088             if _contains_any(instr, ["pause", "stop", "cancel"]):
2089                 return "pause_dialogue"
2090             if _contains_any(instr, ["rename", "name", "called", "label"]):
2091                 return "rename_dialogue"
2092             if re.search(
2093                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2094             ):
2095                 dont_start_req = bool(
2096                     re.search(
2097                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2098                         r"|(?:\bw+out|s+starting\b) "
2099                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2100                         r"|(?:\bleave|s+it\s+paused\b) "
2101                         r"|(?:\bkeep|s+it\s+paused\b) ",
2102                         instr,
2103                     )
2104                 )
2105                 if dont_start_req:
2106                     return "set_dialogue"
2107                 else:
2108                     return "run_dialogue"
2109             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2110                 return "adjust_dialogue"
2111             if re.search(
2112                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2113                 instr,
2114             ):
2115                 return "adjust_dialogue"
2116             return "run_dialogue"
2117
2118         if "snooze" in instr:
2119             return "snooze_dialogue"
2120
2121         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2122             return "delete_dialogue"
2123
2124         if "alarm" in instr and _contains_any(
2125             instr,
2126             [
2127                 "delay",
2128                 "resched",
2129                 "push",
2130                 "move",
2131                 "change",
2132                 "shift",
2133                 "defer",
2134                 "later",
2135                 "increase",
2136             ],
2137         ):
2138             return "snooze_dialogue"
2139
2140     return _infer_dialogue
2141
2142     # -----
2143     # DIALOGUE
2144     # -----
2145     def _infer_dialogue(self) -> str:
2146         instr = self.instruction
2147         has_sw = _has_stopwatch(instr)
2148
2149         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2150             return "stop_dialogue"
2151         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2152             return "reset_dialogue"
2153
2154         if (
2155             (re.search(r"\btime\b", instr) or "clock" in instr)
2156             and re.search(r"\bin\s+\w+", instr)
2157             and not _contains_any(instr, ["timer", "alarm"])
2158         ):
2159             return "add_dialogue"
2160
2161         if "timer" in instr:
2162             if _contains_any(instr, ["delete", "remove", "clear"]):
2163                 return "delete_dialogue"
2164             if _contains_any(instr, ["pause", "stop", "cancel"]):
2165                 return "pause_dialogue"
2166             if _contains_any(instr, ["rename", "name", "called", "label"]):
2167                 return "rename_dialogue"
2168             if re.search(
2169                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2170             ):
2171                 dont_start_req = bool(
2172                     re.search(
2173                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2174                         r"|(?:\bw+out|s+starting\b) "
2175                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2176                         r"|(?:\bleave|s+it\s+paused\b) "
2177                         r"|(?:\bkeep|s+it\s+paused\b) ",
2178                         instr,
2179                     )
2180                 )
2181                 if dont_start_req:
2182                     return "set_dialogue"
2183                 else:
2184                     return "run_dialogue"
2185             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2186                 return "adjust_dialogue"
2187             if re.search(
2188                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2189                 instr,
2190             ):
2191                 return "adjust_dialogue"
2192             return "run_dialogue"
2193
2194         if "snooze" in instr:
2195             return "snooze_dialogue"
2196
2197         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2198             return "delete_dialogue"
2199
2200         if "alarm" in instr and _contains_any(
2201             instr,
2202             [
2203                 "delay",
2204                 "resched",
2205                 "push",
2206                 "move",
2207                 "change",
2208                 "shift",
2209                 "defer",
2210                 "later",
2211                 "increase",
2212             ],
2213         ):
2214             return "snooze_dialogue"
2215
2216     return _infer_dialogue
2217
2218     # -----
2219     # DIALOGUE
2220     # -----
2221     def _infer_dialogue(self) -> str:
2222         instr = self.instruction
2223         has_sw = _has_stopwatch(instr)
2224
2225         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2226             return "stop_dialogue"
2227         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2228             return "reset_dialogue"
2229
2230         if (
2231             (re.search(r"\btime\b", instr) or "clock" in instr)
2232             and re.search(r"\bin\s+\w+", instr)
2233             and not _contains_any(instr, ["timer", "alarm"])
2234         ):
2235             return "add_dialogue"
2236
2237         if "timer" in instr:
2238             if _contains_any(instr, ["delete", "remove", "clear"]):
2239                 return "delete_dialogue"
2240             if _contains_any(instr, ["pause", "stop", "cancel"]):
2241                 return "pause_dialogue"
2242             if _contains_any(instr, ["rename", "name", "called", "label"]):
2243                 return "rename_dialogue"
2244             if re.search(
2245                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2246             ):
2247                 dont_start_req = bool(
2248                     re.search(
2249                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2250                         r"|(?:\bw+out|s+starting\b) "
2251                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2252                         r"|(?:\bleave|s+it\s+paused\b) "
2253                         r"|(?:\bkeep|s+it\s+paused\b) ",
2254                         instr,
2255                     )
2256                 )
2257                 if dont_start_req:
2258                     return "set_dialogue"
2259                 else:
2260                     return "run_dialogue"
2261             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2262                 return "adjust_dialogue"
2263             if re.search(
2264                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2265                 instr,
2266             ):
2267                 return "adjust_dialogue"
2268             return "run_dialogue"
2269
2270         if "snooze" in instr:
2271             return "snooze_dialogue"
2272
2273         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2274             return "delete_dialogue"
2275
2276         if "alarm" in instr and _contains_any(
2277             instr,
2278             [
2279                 "delay",
2280                 "resched",
2281                 "push",
2282                 "move",
2283                 "change",
2284                 "shift",
2285                 "defer",
2286                 "later",
2287                 "increase",
2288             ],
2289         ):
2289             return "snooze_dialogue"
2290
2291     return _infer_dialogue
2292
2293     # -----
2294     # DIALOGUE
2295     # -----
2296     def _infer_dialogue(self) -> str:
2297         instr = self.instruction
2298         has_sw = _has_stopwatch(instr)
2299
2300         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2301             return "stop_dialogue"
2302         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2303             return "reset_dialogue"
2304
2305         if (
2306             (re.search(r"\btime\b", instr) or "clock" in instr)
2307             and re.search(r"\bin\s+\w+", instr)
2308             and not _contains_any(instr, ["timer", "alarm"])
2309         ):
2310             return "add_dialogue"
2311
2312         if "timer" in instr:
2313             if _contains_any(instr, ["delete", "remove", "clear"]):
2314                 return "delete_dialogue"
2315             if _contains_any(instr, ["pause", "stop", "cancel"]):
2316                 return "pause_dialogue"
2317             if _contains_any(instr, ["rename", "name", "called", "label"]):
2318                 return "rename_dialogue"
2319             if re.search(
2320                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2321             ):
2322                 dont_start_req = bool(
2323                     re.search(
2324                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2325                         r"|(?:\bw+out|s+starting\b) "
2326                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2327                         r"|(?:\bleave|s+it\s+paused\b) "
2328                         r"|(?:\bkeep|s+it\s+paused\b) ",
2329                         instr,
2330                     )
2331                 )
2332                 if dont_start_req:
2333                     return "set_dialogue"
2334                 else:
2335                     return "run_dialogue"
2336             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2337                 return "adjust_dialogue"
2338             if re.search(
2339                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2340                 instr,
2341             ):
2342                 return "adjust_dialogue"
2343             return "run_dialogue"
2344
2345         if "snooze" in instr:
2346             return "snooze_dialogue"
2347
2348         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2349             return "delete_dialogue"
2350
2351         if "alarm" in instr and _contains_any(
2352             instr,
2353             [
2354                 "delay",
2355                 "resched",
2356                 "push",
2357                 "move",
2358                 "change",
2359                 "shift",
2360                 "defer",
2361                 "later",
2362                 "increase",
2363             ],
2364         ):
2365             return "snooze_dialogue"
2366
2367     return _infer_dialogue
2368
2369     # -----
2370     # DIALOGUE
2371     # -----
2372     def _infer_dialogue(self) -> str:
2373         instr = self.instruction
2374         has_sw = _has_stopwatch(instr)
2375
2376         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2377             return "stop_dialogue"
2378         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2379             return "reset_dialogue"
2380
2381         if (
2382             (re.search(r"\btime\b", instr) or "clock" in instr)
2383             and re.search(r"\bin\s+\w+", instr)
2384             and not _contains_any(instr, ["timer", "alarm"])
2385         ):
2386             return "add_dialogue"
2387
2388         if "timer" in instr:
2389             if _contains_any(instr, ["delete", "remove", "clear"]):
2390                 return "delete_dialogue"
2391             if _contains_any(instr, ["pause", "stop", "cancel"]):
2392                 return "pause_dialogue"
2393             if _contains_any(instr, ["rename", "name", "called", "label"]):
2394                 return "rename_dialogue"
2395             if re.search(
2396                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2397             ):
2398                 dont_start_req = bool(
2399                     re.search(
2400                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2401                         r"|(?:\bw+out|s+starting\b) "
2402                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2403                         r"|(?:\bleave|s+it\s+paused\b) "
2404                         r"|(?:\bkeep|s+it\s+paused\b) ",
2405                         instr,
2406                     )
2407                 )
2408                 if dont_start_req:
2409                     return "set_dialogue"
2410                 else:
2411                     return "run_dialogue"
2412             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2413                 return "adjust_dialogue"
2414             if re.search(
2415                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2416                 instr,
2417             ):
2418                 return "adjust_dialogue"
2419             return "run_dialogue"
2420
2421         if "snooze" in instr:
2422             return "snooze_dialogue"
2423
2424         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2425             return "delete_dialogue"
2426
2427         if "alarm" in instr and _contains_any(
2428             instr,
2429             [
2430                 "delay",
2431                 "resched",
2432                 "push",
2433                 "move",
2434                 "change",
2435                 "shift",
2436                 "defer",
2437                 "later",
2438                 "increase",
2439             ],
2440         ):
2440             return "snooze_dialogue"
2441
2442     return _infer_dialogue
2443
2444     # -----
2445     # DIALOGUE
2446     # -----
2447     def _infer_dialogue(self) -> str:
2448         instr = self.instruction
2449         has_sw = _has_stopwatch(instr)
2450
2451         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2452             return "stop_dialogue"
2453         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2454             return "reset_dialogue"
2455
2456         if (
2457             (re.search(r"\btime\b", instr) or "clock" in instr)
2458             and re.search(r"\bin\s+\w+", instr)
2459             and not _contains_any(instr, ["timer", "alarm"])
2460         ):
2461             return "add_dialogue"
2462
2463         if "timer" in instr:
2464             if _contains_any(instr, ["delete", "remove", "clear"]):
2465                 return "delete_dialogue"
2466             if _contains_any(instr, ["pause", "stop", "cancel"]):
2467                 return "pause_dialogue"
2468             if _contains_any(instr, ["rename", "name", "called", "label"]):
2469                 return "rename_dialogue"
2470             if re.search(
2471                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2472             ):
2473                 dont_start_req = bool(
2474                     re.search(
2475                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2476                         r"|(?:\bw+out|s+starting\b) "
2477                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2478                         r"|(?:\bleave|s+it\s+paused\b) "
2479                         r"|(?:\bkeep|s+it\s+paused\b) ",
2480                         instr,
2481                     )
2482                 )
2483                 if dont_start_req:
2484                     return "set_dialogue"
2485                 else:
2486                     return "run_dialogue"
2487             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2488                 return "adjust_dialogue"
2489             if re.search(
2490                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2491                 instr,
2492             ):
2493                 return "adjust_dialogue"
2494             return "run_dialogue"
2495
2496         if "snooze" in instr:
2497             return "snooze_dialogue"
2498
2499         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2500             return "delete_dialogue"
2501
2502         if "alarm" in instr and _contains_any(
2503             instr,
2504             [
2505                 "delay",
2506                 "resched",
2507                 "push",
2508                 "move",
2509                 "change",
2510                 "shift",
2511                 "defer",
2512                 "later",
2513                 "increase",
2514             ],
2515         ):
2515             return "snooze_dialogue"
2516
2517     return _infer_dialogue
2518
2519     # -----
2520     # DIALOGUE
2521     # -----
2522     def _infer_dialogue(self) -> str:
2523         instr = self.instruction
2524         has_sw = _has_stopwatch(instr)
2525
2526         if has_sw and _contains_any(instr, ["stop", "cancel"]):
2527             return "stop_dialogue"
2528         if has_sw and _contains_any(instr, ["reset", "zero", "set to zero", "clear"]):
2529             return "reset_dialogue"
2530
2531         if (
2532             (re.search(r"\btime\b", instr) or "clock" in instr)
2533             and re.search(r"\bin\s+\w+", instr)
2534             and not _contains_any(instr, ["timer", "alarm"])
2535         ):
2536             return "add_dialogue"
2537
2538         if "timer" in instr:
2539             if _contains_any(instr, ["delete", "remove", "clear"]):
2540                 return "delete_dialogue"
2541             if _contains_any(instr, ["pause", "stop", "cancel"]):
2542                 return "pause_dialogue"
2543             if _contains_any(instr, ["rename", "name", "called", "label"]):
2544                 return "rename_dialogue"
2545             if re.search(
2546                 r"\badd\b[^n]*?\b\d+s*(?:hour|minute|second)s?\s+timer", instr
2547             ):
2548                 dont_start_req = bool(
2549                     re.search(
2550                         r"^(?:\b(?:don't|do|s+not)\s+(?:start|run)\b) "
2551                         r"|(?:\bw+out|s+starting\b) "
2552                         r"|(?:\b(?:but|and)\s+don't)s+start\b) "
2553                         r"|(?:\bleave|s+it\s+paused\b) "
2554                         r"|(?:\bkeep|s+it\s+paused\b) ",
2555                         instr,
2556                     )
2557                 )
2558                 if dont_start_req:
2559                     return "set_dialogue"
2560                 else:
2561                     return "run_dialogue"
2562             if _contains_any(instr, ["increase", "extend", "more", "up"]):
2563                 return "adjust_dialogue"
2564             if re.search(
2565                 r"\badd\b[^n]*?\b(minutes?|hours?|seconds?)\b[^n]*?\bto\b[^n]*?\btimer\b",
2566                 instr,
2567             ):
2568                 return "adjust_dialogue"
2569             return "run_dialogue"
2570
2571         if "snooze" in instr:
2572             return "snooze_dialogue"
2573
2574         if _contains_any(instr, ["delete", "remove"]) and "alarm" in instr:
2575             return "delete_dialogue"
2576
2577         if "alarm" in instr and _contains_any(
2578             instr,
2579             [
2580                 "delay",
2581                 "resched",
2582                 "push",
2583                 "move",
2584                 "change",
2585                 "shift",
2586                 "defer",
2587                 "later",
2
```

```

1728
1729     ],
1730     ):
1731         if "alarm" in instr:
1732             return "set_alarm"
1733
1734         if _contains_any(
1735             instr, ["add", "timezone", "time zone", "city", "world clock"]
1736         ):
1737             return "add_city"
1738
1739     return "none"
1740
1741     def _update_alarm_creation_seen(self, state: str):
1742         s = state.lower()
1743         if any(kw in s for kw in ("add alarm", "alarm time", "select time")):
1744             self._alarm_creation_seen = True
1745
1746         # -----
1747         #           GENERIC reward helpers
1748         # -----
1749     def _reward_from_progress(self, fn: Callable[[str], int], state: str) -> float:
1750         lvl = fn(state)
1751         if self.task_type == "set_alarm":
1752             if lvl >= 3:
1753                 if self._alarm_creation_seen:
1754                     self.goal_achieved = True
1755                     return 100.0
1756                 if self._confirm_goal_seen or self._best_level >= 2:
1757                     self.goal_achieved = True
1758                     return 100.0
1759                 self._confirm_goal_seen = True
1760                 self._best_level = max(self._best_level, 2)
1761                 return 0.99
1762             self._confirm_goal_seen = False
1763             if lvl >= 3:
1764                 self.goal_achieved = True
1765                 return 100.0
1766             if lvl > self._best_level:
1767                 inc = (lvl - self._best_level) * self._SHAPING_INC
1768                 self._best_level = lvl
1769                 return min(inc, 0.99)
1770
1771     return 0.0
1772
1773     # -----
1774     #           TIMER-specific dense reward
1775     # -----
1776     def _reward_timer(self, state: str, start_req: bool) -> float:
1777         reward = 0.0
1778         if _timer_tab_selected(state):
1779             reward += 0.2
1780
1781         current_val = _extract_timer_components(state)
1782         if current_val is None:
1783             return min(reward, 0.99)
1784         cur_hh, cur_mm, cur_ss = current_val
1785         current_digit_string = f"{cur_hh:02d}{cur_mm:02d}{cur_ss:02d}".lstrip("0")
1786         if current_digit_string == "":
1787             current_digit_string = "0"
1788         goal_digit_string = f"{self.goal_hms[0]:02d}{self.goal_hms[1]:02d}{self.goal_hms[2]:02d}".lstrip("0")
1789         if goal_digit_string == "":
1790             goal_digit_string = "0"
1791         running = _is_timer_running(state)
1792         if current_digit_string == goal_digit_string and running:
1793             if start_req and running:
1794                 self.goal_achieved = True
1795                 return 100.0
1796             if not start_req and not running:
1797                 self.goal_achieved = True
1798                 return 100.0
1799
1800         matching_digits = 0
1801         for i in range(0, min(len(current_digit_string), len(goal_digit_string))):
1802             if goal_digit_string[i] == current_digit_string[i]:
1803                 matching_digits += 1
1804             else:
1805                 # Stop counting as soon as a mismatch occurs
1806                 break
1807         reward += (matching_digits / len(goal_digit_string)) * 0.7
1808
1809     return min(reward, 0.99)
1810
1811     # -----
1812     #           Other dense rewards (adjust_timer, snooze)

```

```

1782
1783     # -----
1784     def _reward_adjust_timer(self, state: str) -> float:
1785         reward = 0.0
1786         if _timer_screen_visible(state):
1787             reward += 0.2
1788         current_val = _extract_timer_value(state)
1789         if current_val is None:
1790             return min(reward, 0.99)
1791         if self.initial_timer_val is None:
1792             self.initial_timer_val = self.prev_timer_val = self.max_timer_val = (
1793                 current_val
1794             )
1795             return min(reward, 0.99)
1796         if current_val > (self.max_timer_val or 0):
1797             self.max_timer_val = current_val
1798         diff_step = current_val - (self.prev_timer_val or current_val)
1799         if diff_step > self._ADJ_INC_THRESHOLD:
1800             self.increments += max(1, int(round(diff_step / 60.0)))
1801         elif diff_step < -1:
1802             self._countdown_seen = True
1803             self.prev_timer_val = current_val
1804             net_increase_max = (self.max_timer_val or current_val) - self.initial_timer_val
1805             fraction_by_inc = self.increments / max(1, self.needed_increments)
1806             fraction_by_delta = net_increase_max / max(1, self.goal_seconds)
1807             progress_fraction = min(1.0, max(fraction_by_inc, fraction_by_delta))
1808             reward += 0.8 * progress_fraction
1809             tol = max(2, int(self.goal_seconds * 0.05))
1810             goal_reached_primary = (
1811                 self.increments >= self.needed_increments
1812                 or net_increase_max >= self.goal_seconds - tol
1813             )
1814             committed = (
1815                 _is_timer_running(state) or _is_timer_paused(state) or self._countdown_seen
1816             )
1817             keypad = _timer_keypad_mode(state)
1818             secondary_success = (
1819                 not goal_reached_primary
1820                 and net_increase_max >= 0.4 * self.goal_seconds
1821                 and self.increments >= 1
1822                 and self._countdown_seen
1823                 and committed
1824                 and not keypad
1825             )
1826             if (goal_reached_primary or secondary_success) and committed and not keypad:
1827                 self.goal_achieved = True
1828                 return 100.0
1829             return min(reward, 0.99)
1830
1831     def _reward_snooze(self, state: str) -> float:
1832         s_low = state.lower()
1833         if "select snooze duration" in s_low:
1834             self._snooze_dialog_seen = True
1835             classic_done = (
1836                 "alarm snoozed" in s_low
1837                 or bool(re.search(r"snoozed\s+for\s+\d+", s_low))
1838                 or bool(re.search(r"\bsnooz(ing|ed)\b", s_low))
1839             )
1840             row_done = (
1841                 self._snooze_dialog_seen
1842                 and "select snooze duration" not in s_low
1843                 and "snooze" in s_low
1844                 and bool(re.search(r"\b\d+\s+minutes?\b", s_low))
1845             )
1846             if classic_done or row_done:
1847                 self.goal_achieved = True
1848                 return 100.0
1849             reward = 0.0
1850             if _alarm_tab_selected(state):
1851                 reward += 0.2
1852             if re.search(r"(content_description|text)\s*\s*:\s*\"snooze\"", state, re.I):
1853                 reward += 0.3
1854             if "select snooze duration" in s_low:
1855                 reward += 0.2
1856             return min(reward, 0.99)
1857
1858     # -----
1859     #     Progress-level helpers (stopwatch/timer/alarm)
1860     # -----
1861     def _pl_reset_stopwatch(self, state: str) -> int:
1862         if self.restart_mode:
1863             if _stopwatch_running(state) and self._reset_seen:

```

```

1836
1837     return 3
1838     if _stopwatch_time_zero(state):
1839         self._reset_seen = True
1840         return 2
1841     if _button_visible(state, "Reset") and (
1842         _stopwatch_tab_selected(state) or "stopwatch" in state.lower()
1843     ):
1844         return 1
1845     return 0
1846     if _stopwatch_time_zero(state):
1847         return 3
1848     if _button_visible(state, "Reset") and (
1849         _stopwatch_tab_selected(state) or "stopwatch" in state.lower()
1850     ):
1851         return 2
1852     if _stopwatch_tab_selected(state):
1853         return 1
1854     return 0
1855
1856     def _pl_pause_stopwatch(self, state: str) -> int:
1857         if not _stopwatch_running(state):
1858             return 3
1859         if _stopwatch_tab_selected(state):
1860             return 1
1861         return 0
1862
1863     def _pl_restart_stopwatch(self, state: str) -> int:
1864         running = _stopwatch_running(state)
1865         at_zero = _stopwatch_time_zero(state)
1866         if running and self._reset_seen:
1867             return 3
1868         if at_zero:
1869             self._reset_seen = True
1870             return 2
1871         if _stopwatch_tab_selected(state):
1872             return 1
1873         return 0
1874
1875     def _pl_start_stopwatch(self, state: str) -> int:
1876         if _stopwatch_running(state):
1877             return 3
1878         if "stopwatch" in state.lower() or _stopwatch_tab_selected(state):
1879             return 2
1880         if _contains_any(state.lower(), ["the clock", '"clock"', "alarms", "timer"]):
1881             return 1
1882         return 0
1883
1884     def _pl_pause_timer(self, state: str) -> int:
1885         if _is_timer_paused(state):
1886             return 3
1887         current_val = _extract_timer_value(state)
1888         if current_val is not None:
1889             if self._prev_timer_val_for_pause == current_val:
1890                 self._same_val_steps += 1
1891             else:
1892                 self._same_val_steps = 0
1893             self._prev_timer_val_for_pause = current_val
1894         else:
1895             self._same_val_steps = 0
1896         stable_and_visible = (
1897             _timer_tab_selected(state)
1898             and current_val is not None
1899             and self._same_val_steps >= 1
1900             and not _is_timer_running(state)
1901         )
1902         if stable_and_visible:
1903             return 3
1904         if _timer_paused_notification(state) and _timer_tab_selected(state):
1905             return 3
1906         if _timer_paused_notification(state):
1907             return 2
1908         if _is_timer_running(state):
1909             return 2
1910         if _timer_tab_selected(state):
1911             return 1
1912         return 0
1913
1914     def _pl_delete_timer(self, state: str) -> int:
1915         if _timer_deleted(state):
1916             return 3
1917         if _contains_any(

```

```

1890
1888     state.lower(), ["delete", "remove", "clear", "âšn", "backspace", "cancel"]
1889
1890     ):
1891         return 2
1892     if _timer_tab_selected(state):
1893         return 1
1894     return 0
1895
1896     def _pl_delete_alarm(self, state: str) -> int:
1897         s_low = state.lower()
1898         had_alarm_before = self._alarm_present_ever
1899         alarm_now = _any_alarm_present(state) or _detect_alarm_time(state)
1900         if alarm_now:
1901             self._alarm_present_ever = True
1902         if _is_alarm_deleted(state) and had_alarm_before:
1903             return 3
1904         if " delete" in s_low or "ôšúš" in s_low or re.search(r"trash|remove", s_low):
1905             return 2
1906         if alarm_now:
1907             return 1
1908         return 0
1909
1910     def _pl_add_city(self, state: str) -> int:
1911         city_seen = self.city_keywords and any(
1912             re.search(rf"\b{re.escape(kw)}\b", state, re.I) for kw in self.city_keywords
1913         )
1914         in_search = (
1915             re.search(r"search for a city", state, re.I)
1916             or "select time zone" in state.lower()
1917         )
1918         if city_seen and _clock_tab_selected(state) and not in_search:
1919             return 3
1920         if city_seen:
1921             return 2
1922         if _clock_tab_selected(state):
1923             return 1
1924         return 0
1925
1926     def _pl_set_alarm(self, state: str) -> int:
1927         if self._alarm_goal_met(state):
1928             return 3
1929         if "select time" in state.lower() or "alarm set for" in state.lower():
1930             return 2
1931         if _alarm_tab_selected(state):
1932             return 1
1933         return 0
1934
1935     def _pl_adjust_alarm(self, state: str) -> int:
1936         if not self._orig_seen and _alarm_time_present(
1937             state, self.orig_hour24, self.orig_minute, None
1938         ):
1939             self._orig_seen = True
1940         if (
1941             _alarm_time_present(state, self.goal_hour24, self.goal_minute, None)
1942             and self._orig_seen
1943         ):
1944             return 3
1945         if "select time" in state.lower() or "alarm set for" in state.lower():
1946             return 2
1947         if _alarm_tab_selected(state) or self._orig_seen:
1948             return 1
1949         return 0
1950
1951     def _pl_rename_timer(self, state: str) -> int:
1952         dialog_open = _rename_dialog_open(state)
1953         label_seen = _timer_label_present(state, self.goal_label)
1954         if label_seen and not dialog_open:
1955             return 3
1956         if dialog_open:
1957             return 2
1958         if _timer_tab_selected(state):
1959             return 1
1960         return 0
1961
1962     # -----
1963     # Additional parsing / goal-checking helpers
1964     # -----
1965     def _parse_recurrence_days(self, instr_l: str) -> Set[str]:
1966         days = {
1967             "sunday",
1968             "monday",
1969         }

```

```

1944
1945 968     "tuesday",
1946 969     "wednesday",
1947 970     "thursday",
1948 971     "friday",
1949 972     "saturday",
1950 973     "weekdays",
1951 974     "weekday",
1952 975     "week day",
1953 976     "week days",
1954 977     "weekends",
1955 978     "every day",
1956 979     "everyday",
1957 980 }
1958 981 found: Set[str] = set()
1959 982 for d in days:
1960 983     if d in instr_1:
1961 984         if d in {
1962 985             "weekdays",
1963 986             "weekday",
1964 987             "week day",
1965 988             "week days",
1966 989             "every day",
1967 990             "everyday",
1968 991         }:
1969 992         found.update(
1970 993             {"monday", "tuesday", "wednesday", "thursday", "friday"}
1971 994         )
1972 995     elif d == "weekends":
1973 996         found.update({"saturday", "sunday"})
1974 997     else:
1975 998         found.add(d)
1976 999 return found
1977 1000
1978 1001 def _alarm_goal_met(self, state: str) -> bool:
1979 1002     # time & presence
1980 1003     if self.alarm_any_time:
1981 1004         time_ok = _any_alarm_present(state)
1982 1005     else:
1983 1006         time_ok = _alarm_time_present(
1984 1007             state, self.goal_hour24, self.goal_minute, self.goal_ap
1985 1008         )
1986 1009     if not time_ok or not _alarm_context_present(state):
1987 1010         return False
1988 1011
1989 1012     # recurrence handling
1990 1013     if not self.recurrence_days:
1991 1014         return True
1992 1015
1993 1016     # exact match
1994 1017     if self.recurrence_days.issubset(_selected_weekdays(state)):
1995 1018         return True
1996 1019
1997 1020     # lenient weekday rule
1998 1021     weekdays_set = {"monday", "tuesday", "wednesday", "thursday", "friday"}
1999 1022     if self.recurrence_days == weekdays_set and _day_toggle_buttons_visible(state):
2000 1023         if "not scheduled" not in state.lower(): # ensure days have been picked
2001 1024             return True
2002 1025
2003 1026 return False
2004 1027
2005 1028 def _parse_alarm_goal_time(self):
2006 1029     times = self._extract_times(self.instruction_norm_full)
2007 1030     if not times:
2008 1031         self.goal_hour24, self.goal_minute, self.goal_ap = _parse_alarm_time(
2009 1032             self.instruction_norm_full
2010 1033         )
2011 1034     return
2012 1035     alarm_pos = self.instruction_norm.rfind("alarm")
2013 1036     chosen = next((t[:3] for t in times if t[3] > alarm_pos), times[0][:3])
2014 1037     self.goal_hour24, self.goal_minute, self.goal_ap = chosen
2015 1038
2016 1039 def _parse_adjusted_alarm(self) -> Tuple[int, int]:
2017 1040     base_h, base_m, _ = _parse_alarm_time(self.instruction_norm_full)
2018 1041     m = re.search(
2019 1042         r"\bby\s+(\d+)\s*(hour|hours|minute|minutes)\b", self.instruction_norm
2020 1043     )
2021 1044     if m:
2022 1045         num = int(m.group(1))
2023 1046         unit = m.group(2)
2024 1047         delta = num * (60 if "hour" in unit else 1)
2025 1048         total = (base_h * 60 + base_m + delta) % (24 * 60)
2026 1049         return total // 60, total % 60

```

Listing 1: Android Control Generated Reward.

2052 A.9 PROMPTS

2053

2054

2055

2056

2057

**Goal Identification Prompt**

Given this reward code: {reward\_code}

**Trajectory:**

{trajectory}

Please analyze the state sequence and the agent's instruction.

Identify the index of the goal state. The state indices are 1-based.

**OUTPUT FORMAT:**

Answer in a json format as follows:

'reasoning': Explain your reasoning for choosing the goal state(s).

'goal\_state\_indexes': A list of integers representing the 1-based index of the goal state(s), or -1 if no goal state is present.

2068

2069

2070 Prompt 1: The prompt for identifying the goal state(s) within a trajectory using a given reward  
2071 function.

2072

2073

2074

2075

2076

2077

2078

2079

2080

2081

2082

2083

2084

2085

2086

2087

2088

2089

2090

2091

2092

2093

2094

2095

2096

2097

2098

2099

2100

2101

2102

2103

2104

2105

```

2106
2107
2108 LLM Initial Reward Generation
2109
2110 You are an ML engineer writing reward functions for RL training.
2111 Given a trajectory with marked goal states, create a Python reward
2112 function that can reproduce this behavior.
2113
2114 Requirements:
2115     • Write self-contained Python 3.9 code
2116     • Always return rewards  $\geq 0$ 
2117     • Make the function generic enough to handle variations
2118         (different positions, orientations, etc.)
2119     • Design for modularity - you might extend this reward later to
2120         handle multiple goal types
2121     • Give 100.0 for the goal state and less than 1.0 (modulated for
2122         shaping) for all other states
2123 Environment Details:
2124 {env_code}, {import_instructions}, {state_description}
2125
2126 Trajectories
2127 {expert_trajectories}
2128
2129 Key Instructions:
2130     1. Analyze the trajectory to understand what constitutes success
2131     2. Identify intermediate progress that should be rewarded
2132     3. Create utility functions for reusable reward components
2133 The code will be written to a file and then imported.
2134 OUTPUT FORMAT:
2135 Answer in a json format as follows:
2136 'reasoning': Given the reason for your answer
2137 'reward_class_code': Code for the Reward function class in the
2138 format:
2139 # imports
2140 <imports_here>
2141 # utils functions
2142 <utils functions here>
2143 # reward function
2144 class Reward:
2145     def __init__(self, extra_info=None):
2146         <code_here>
2147
2148     def reward_fn(self, state):
2149         <code_here>
2150
2151     def debug_fn(self, state):
2152         <code_here>
2153
2154 The Reward class will be initialized with the extra_info argument.
2155 Describe in the comments of the class the behaviour you are trying to
2156 reproduce.
2157 reward_fn and debug_fn receive only state as argument. The debug_fn
2158 should return a string that will be printed and shown to you after
2159 calling reward_fn on each state. You can print internal class
properties to help you debug the function. Extract any needed
information from the state or store it in the class. The Reward
class will be re-initialised at the beginning of each episode.
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2789
2790
2791
2792
2793
2794
2795
2796
2797
2798
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2889
2890
2891
2892
2893
2894
2895
2896
2897
2898
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3089
3090
3091
3092
3093
3094
3095
3096
3097
3098
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3190
3191
3192
3193
3194
3195
3196
3197
3198
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3289
3290
3291
3292
3293
3294
3295
3296
3297
3298
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3389
3390
3391
3392
3393
3394
3395
3396
3397
3398
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3489
3490
3491
3492
3493
3494
3495
3496
3497
3498
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
3550
3551
3552
3553
3554
3555
3556
3557
3558
3559
3560
3561
3562
3563
3564
3565
3566
3567
3568
3569
3570
3571
3572
3573
3574
3575
3576
3577
3578
3579
3580
3581
3582
3583
3584
3585
3586
3587
3588
3589
3590
3591
3592
3593
3594
3595
3596
3597
3598
3599
3600
3601
3602
3603
3604
3605
3606
3607
3608
3609
3610
3611
3612
3613
3614
3615
3616
3617
3618
3619
3620
3621
3622
3623
3624
3625
3626
3627
3628
3629
3630
3631
3632
3633
3634
3635
3636
3637
3638
3639
3640
3641
3642
3643
3644
3645
3646
3647
3648
3649
3650
3651
3652
3653
3654
3655
3656
3657
3658
3659
3660
3661
3662
3663
3664
3665
3666
3667
3668
3669
3670
3671
3672
3673
3674
3675
3676
3677
3678
3679
3680
3681
3682
3683
3684
3685
3686
3687
3688
3689
3690
3691
3692
3693
3694
3695
3696
3697
3698
3699
3700
3701
3702
3703
3704
3705
3706
3707
3708
3709
3710
3711
3712
3713
3714
3715
3716
3717
3718
3719
3720
3721
3722
3723
3724
3725
3726
3727
3728
3729
3730
3731
3732
3733
3734
3735
3736
3737
3738
3739
3740
3741
3742
3743
3744
3745
3746
3747
3748
3749
3750
3751
3752
3753
3754
3755
3756
3757
3758
3759
3760
3761
3762
3763
3764
3765
3766
3767
3768
3769
3770
3771
3772
3773
3774
3775
3776
3777
3778
3779
3780
3781
3782
3783
3784
3785
3786
3787
3788
3789
3790
3791
3792
3793
3794
3795
3796
3797
3798
3799
3800
3801
3802
3803
3804
3805
3806
3807
3808
3809
3810
3811
3812
3813
3814
3815
3816
3817
3818
3819
3820
3821
3822
3823
3824
3825
3826
3827
3828
3829
3830
3831
3832
3833
3834
3835
3836
3837
3838
3839
3840
3841
3842
3843
3844
3845
3846
3847
3848
3849
3850
3851
3852
3853
3854
3855
3856
3857
3858
3859
3860
3861
3862
3863
3864
3865
3866
3867
3868
3869
3870
3871
3872
3873
3874
3875
3876
3877
3878
3879
3880
3881
3882
3883
3884
3885
3886
3887
3888
3889
3890
3891
3892
3893
3894
3895
3896
3897
3898
3899
3900
3901
3902
3903
3904
3905
3906
3907
3908
3909
3910
3911
3912
3913
3914
3915
3916
3917
3918
3919
3920
3921
3922
3923
3924
3925
3926
3927
3928
3929
3930
3931
3932
3933
3934
3935
3936
3937
3938
3939
3940
3941
3942
3943
3944
3945
3946
3947
3948
3949
3950
3951
3952
3953
3954
3955
3956
3957
3958
3959
3960
3961
3962
3963
3964
3965
3966
3967
3968
3969
3970
3971
3972
3973
3974
3975
3976
3977
3978
3979
3980
3981
3982
3983
3984
3985
3986
3987
3988
3989
3990
3991
3992
3993
3994
3995
3996
3997
3998
3999
4000
4001
4002
4003
4004
4005
4006
4007
4008
4009
4010
4011
4012
4013
4014
4015
4016
4017
4018
4019
4020
4021
4022
4023
4024
4025
4026
4027
4028
4029
4030
4031
4032
4033
4034
4035
4036
4037
4038
4039
4040
4041
4042
4043
4044
4045
4046
4047
4048
4049
4050
4051
4052
4053
4054
4055
4056
4057
4058
4059
4060
4061
4062
4063
4064
4065
4066
4067
4068
4069
4070
4071
4072
4073
4074
4075
4076
4077
4078
4079
4080
4081
4082
4083
4084
4085
4086
4087
4088
4089
4090
4091
4092
4093
4094
4095
4096
4097
4098
4099
4100
4101
4102
4103
4104
4105
4106
4107
4108
4109
4110
4111
4112
4113
4114
4115
4116
4117
4118
4119
4120
4121
4122
4123
4124
4125
4126
4127
4128
4129
4130
4131
4132
4133
4134
4135
4136
4137
4138
4139
4140
4141
4142
4143
4144
4145
4146
4147
4148
4149
4150
4151
4152
4153
4154
4155
4156
4157
4158
4159
4160
4161
4162
4163
4164
4165
4166
4167
4168
4169
4170
4171
4172
4173
4174
4175
4176
4177
4178
4179
4180
4181
4182
4183
4184
4185
4186
4187
4188
4189
4190
4191
4192
4193
4194
4195
4196
4197
4198
4199
4200
4201
4202
4203
4204
4205
4206
4207
4208
4209
4210
4211
4212
4213
4214
4215
4216
4217
4218
4219
4220
4221
4222
4223
4224
4225
4226
4227
4228
4229
4230
4231
4232
4233
4234
4235
4236
4237
4238
4239
4240
4241
4242
4243
4244
4245
4246
4247
4248
4249
4250
4251
4252
4253
4254
4255
4256
4257
4258
4259
4260
4261
4262
4263
4264
4265
4266
4267
4268
4269
4270
4271
4272
4273
4274
4275
4276
4277
4278
4279
4280
4281
4282
4283
42
```

```

2160
2161
2162
2163 You are an ML engineer writing reward functions for RL training.
2164 Given a trajectory with marked goal states, create a Python reward
2165 function that can reproduce this behavior.
2166
Requirements:
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
Evolution Mutation Prompt
You are an ML engineer writing reward functions for RL training.
Given a trajectory with marked goal states, create a Python reward
function that can reproduce this behavior.

Requirements:
• Write self-contained Python 3.9 code
• Always return rewards  $\geq 0$ 
• Make the function generic enough to handle variations
  (different positions, orientations, etc.)
• Design for modularity - you might extend this reward later to
  handle multiple goal types
• Give 100.0 for the goal state and less than 1.0 (modulated for
  shaping) for all other states

Original Reward Code:
{{code}}


{{import_message}}
{{state_description}}


--


CRITICAL: Incorrect Trajectories
The reward function above FAILED on the following trajectories. It
either assigned a high reward to a failed trajectory or failed to
assign the highest reward to the correct goal state. The predicted
rewards for each step are shown.
Change the reward function to fix these errors. The goal is to
make the reward function correctly identify the goal state (or lack
thereof) in these examples.

Key Instructions:
1. Analyze the trajectory to understand what constitutes success
2. Identify intermediate progress that should be rewarded
3. Create utility functions for reusable reward components
4. Implement goal switching logic using extra_info to determine
  which reward function to use
5. Reuse existing utilities where possible
6. Make sure the logic you write generalises to variations in
  'extra_info'

{{incorrect_trajectories}}


{{expert_traj_str}}


--


Now, provide the mutated version of the reward function that
addresses these errors.

OUTPUT FORMAT:
Answer in a json format as follows:
'reasoning': Briefly explain the corrective change you made.
{{REWARD_OUTPUT_FORMAT}}
{{REWARD_EXTRA_INFO}}

```

Prompt 3: The prompt used for evolutionary mutation, providing feedback on incorrect trajectories.

2214 **Evolution Shaping Prompt**

2215

2216 You are an ML engineer writing reward functions for RL training.

2217 Given a trajectory with marked goal states, create a Python reward

2218 function that can reproduce this behavior. **Requirements:**

2219

- Write self-contained Python 3.9 code
- Always return rewards  $\geq 0$
- Make the function generic enough to handle variations (different positions, orientations, etc.)
- Design for modularity - you might extend this reward later to handle multiple goal types
- Give 100.0 for the goal state and less than 1.0 (modulated for shaping) for all other states

2220

2221 **Original Reward Code:**

2222 {env\_code}

2223

2224 {import\_message}

2225 {state\_description}

2226

2227 --

2228 **CRITICAL: Incorrectly Shaped Trajectories**

2229 The reward function above is not shaped optimally on the following

2230 trajectories. This is an expert trajectory, so the reward should

2231 monotonically increase from one state to the next. The predicted

2232 rewards for each step are shown.

2233 Change the reward function to fix these errors.

2234

2235 {incorrect\_expert\_trajectories}

2236 --

2237

2238 Now, provide the mutated version of the reward function that

2239 addresses these errors.

2240

2241 **OUTPUT FORMAT:**

2242 Answer in a json format as follows:

2243 'reasoning': Briefly explain the corrective change you made.

2244 {REWARD\_OUTPUT\_FORMAT}

2245 {REWARD\_EXTRA\_INFO}

**Prompt 4: The prompt used for refining reward shaping based on expert trajectories.**

## A.10 LLM USAGE STATEMENT

2255 We wish to disclose the role of LLMs in the preparation of this work to ensure transparency.

2257 **Manuscript Writing** We employed LLMs to assist in the writing process. This included rephrasing  
2258 sentences and paragraphs to enhance clarity and flow, and checking for grammatical errors and  
2259 stylistic consistency. While LLMs helped refine the presentation of our ideas, all core arguments,  
2260 scientific claims, and the overall structure of the paper were developed by the human authors.

2262 **Code Development and Debugging** In the software development process, LLMs were used as  
2263 a coding assistant. This involved generating specific utility functions based on detailed prompts,  
2264 providing explanations for complex error messages, and suggesting alternative implementations for  
2265 performance or readability improvements. The overall software architecture and core algorithms  
2266 were designed and implemented by the human authors, who verified and tested all LLM-assisted  
2267 code.