Building Open-Ended Embodied Agent via Language-Policy Bidirectional Adaptation

Shaopeng Zhai [*1] **Jie Wang** [*1] **Tianyi Zhang** [*1] **Fuxian Huang** [*1]
Qi Zhang [*1] **Ming Zhou** [*1] **Jing Hou** [*2 1] **Yu Qiao** [1] **Yu Liu** [1]

Abstract

Building open-ended learning agents involves challenges in pre-trained language model (LLM) and reinforcement learning (RL) approaches. LLMs struggle with context-specific real-time interactions, while RL methods face efficiency issues for exploration. To this end, we propose OpenContra, a co-training framework that cooperates LLMs and GRL to construct an open-ended agent capable of comprehending arbitrary human instructions. The implementation comprises two stages: (1) fine-tuning an LLM to translate human instructions into structured goals, and curriculum training a goal-conditioned RL policy to execute arbitrary goals; (2) collaborative training to make the LLM and RL policy learn to adapt each, achieving open-endedness on instruction space. We conduct experiments on Contra, a battle royale FPS game with a complex and vast goal space. The results show that an agent trained with OpenContra comprehends arbitrary human instructions and completes goals with a high completion ratio, which proves that OpenContra may be the first practical solution for constructing open-ended embodied agents.

1. Introduction

The challenge of building general-capable agents in the field of AI remains a significant endeavor (Stanley et al., 2017; Parker-Holder, 2022). This challenge necessitates the development of agents with the ability to continuously learn new skills and even create novel ones, a domain commonly referred to as open-ended learning. The research in open-ended learning is broadly categorized into two main factions: (1) pre-trained LLM agents for open-ended planning (Wang et al., 2023; Ouyang et al., 2022), and (2) RL agents for open-ended control (Team et al., 2021; Balduzzi et al., 2019).

LLM-based methods, particularly those with closed source architectures, focus on planning based on general knowledge acquired during the pre-training stage (Wang et al., 2023). However, they may struggle to comprehend agent interactions in specific contexts and be impossible to learn in the case of real-time interaction. In contrast, RL methods conduct open-ended learning in an end-to-end manner, developing in diverse methodologies such as population-based training (Jaderberg et al., 2019; Team et al., 2021) and curriculum reinforcement learning (Wang et al., 2019; Samvelyan et al., 2022). While RL methods naturally learn contextual understanding by enabling agents to interact directly with environments, these approaches can be inefficient when aiming for open-endedness over the entire goal space, especially under tight computation budgets. Additionally, achieving meaningful open-endedness is essential under practical conditions, grounded by human instructions.

To address these challenges, we propose a collaborative training framework that combines LLM and RL. In this framework, the LLM is responsible for contextual understanding and translating human instructions into structured goals, while the RL agent makes decisions to execute these goals. While some attempts have been made in this direction, existing studies often focus on single-sided learning, where one module is frozen while the other is trainable. This overlooks the potential cooperation between the LLM and RL agent. More specifically, existing efforts typically aim to improve training efficiency or reduce module interaction costs (Hu et al., 2023; Du et al., 2023).

To substantiate our proposal, we present OpenContra, a two-stage implementation. In the first stage, OpenContra conducts independent training, enabling the LLM to generate goals corresponding to natural language instructions and environment contexts and RL agents to be capable of completing as many goals as possible. The LLM is initially trained with human-annotated datasets to generate formatted goal descriptions, followed by multi-step fine-tuning to enhance the precision of goal generation. A warm-up from non-goal RL is performed for RL training to master ba-

[*]Equal contribution [1]Shanghai AI Laboratory [2]Tongji University. Correspondence to: Shaopeng Zhai <zhaishaopeng@pjlab.org.cn>, Yu Liu <liuyu@pjlab.org.cn>.

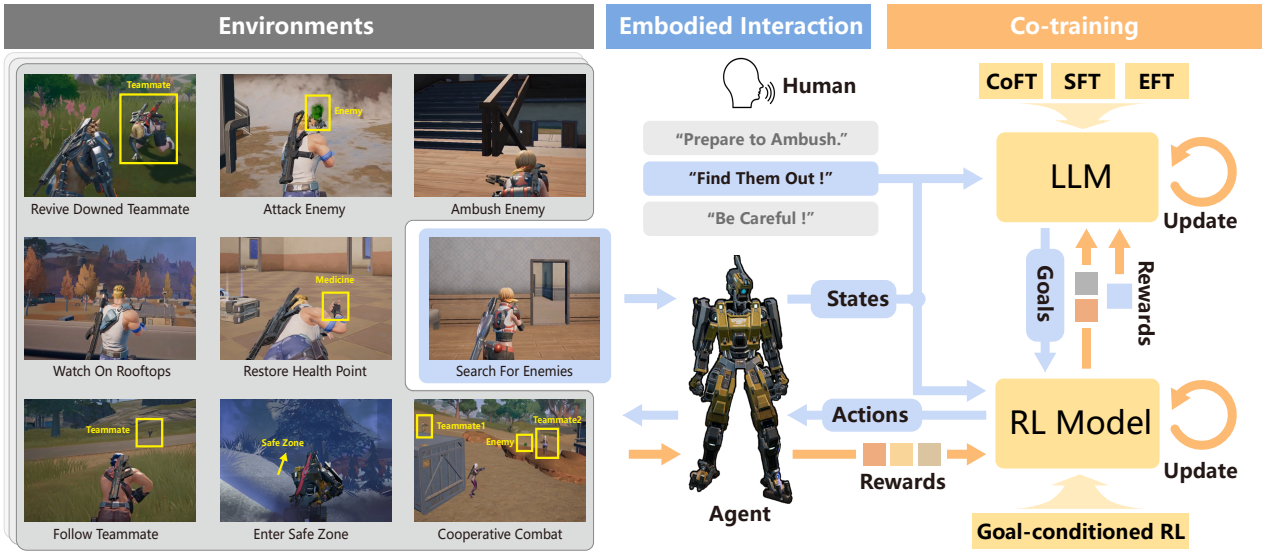


Figure 1: Overview of OpenContra for Contra, a Battle Royale FPS game.

sic skills. Subsequently, a curriculum GRL procedure (Liu et al., 2022; Narvekar et al., 2020) is introduced to train a goal-conditioned policy for goal execution, starting from the well-trained non-goal RL agent. In the second stage, we collaboratively train the LLM and the RL agent to adapt to each other and complete goals corresponding to human instructions. Specifically, we create a dataset of instructions from diverse sources and then random sample instructions to combine with environment states to generate goals with the LLM. These goals are further used to train the goal-conditioned RL agent as in the previous stage. Simultaneously, for the training of the LLM, we introduce Reinforcement Learning with Agent Feedback (RLAF), which considers both environmental feedback and instruction completion. Thus, the LLM will be optimized towards comprehending agent interactions.

We employ Contra as a testbed, a Battle Royale FPS game requiring high-frequency skill operation, with a high-dimensional goal space. Our evaluation includes human-instructed tests to validate open-endedness, wherein arbitrary human instructions are given to the LLM module, and the generated goals' quality and completion ratio is examined (Figure 1).

2. Background

Goal-conditioned Reinforcement Learning. Formally, the goal-conditioned reinforcement learning (GRL) could be formulated as a goal-augmented Markov Decision Process $\mathcal{M}$ (Liu et al., 2022) as follows. Denoting $\mathcal{M}$ a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{G}, \mathcal{P}, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$, $\mathcal{A}$, $\mathcal{G}$ the state space, action space and goal space, respectively. In general, $\mathcal{G}$ is a projection of $\mathcal{S}$, i.e., $\mathcal{G} = Proj(\mathcal{S})$. $\mathcal{P}$ defines the state transition function that given a state and action tuple (s_t, a_t) at timestep t, i.e., $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, where $\Delta(\cdot)$ denotes a distribution. $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$ defines the reward function $r(s, a, g)$ with a given state, action, and goal. At the beginning of an episode τ, a goal g will be sampled from a distribution P_g, which generally defines a target for the agent to solve $\mathcal{M}$. We could further formulate a policy function for the agent that denotes the decision-making at each time step as $\pi : \mathcal{S} \times \mathcal{G} \to \Delta(\mathcal{A})$, which is a distribution over the action space. To solving a $\mathcal{M}$, the agent with policy π should maximize its accumulative reward over the goal space as

$$\mathbb{E}_{a_t \sim \pi(\cdot|s_t,g), s_{t+1} \sim \mathcal{P}(\cdot|s_t,a_t), g \sim P_g, s_0 \sim P_s} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t, g) \right], \tag{1}$$

where P_s the state distribution determines the sampling of s_0, $\gamma \in [0, 1)$ discounted the reward at each time step to guarantee the convergence of policy learning. Normally, $r(s_t, a_t, g)$ is binary to denote whether g has been completed, i.e., $r(s_t, a_t, g)$ values 0 or 1 according to whether $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$ satisfies g, i.e.,

$$r(s_t, a_t, g) = \begin{cases} 1 & s_{t+1} \text{ satisfies } g \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

To approximate the accumulative reward related to the policy $\pi(\cdot|s_t, g)$, GRL suggests using the Universal Value Function Approximator (UVFA) $V(s_t, g)$ to replace the state value function in traditional RL (Schaul et al., 2015a). The

goal representation is still an open question, generally determined by the task to be resolved. We summarize the most popular categories as vector-based (Florensa et al., 2018; Ren et al., 2019; Nair et al., 2018; Fang et al., 2019; Pitis et al., 2020), vision-based (Campero et al., 2020; Nair & Finn, 2019; Warde-Farley et al., 2018; Mendonca et al., 2021; Bousmalis et al., 2023; Groth et al., 2021) and text-based (Brohan et al., 2023; Lynch & Sermanet, 2020). In this work, we model the goal space as a structured vector corresponding to multiple sub-goal spaces. One research direction of goal-conditioned RL is to learn a policy (or agent) that completes as many goals as possible, which could be regarded as achieving open-endedness in the goal space $\mathcal{G}$. However, it is almost impossible to traverse all goals from $\mathcal{G}$ as (1) P_g may be agnostic to an agent, and (2) $\mathcal{G}$ may be uncountable and continuous. To satisfy the open-endedness requirements, a popular solution is leveraging curriculum reinforcement learning (Weng, 2020), i.e., automatically discovering novel goals from past training. Despite numerous curriculum approaches for achieving open-endedness, a critical challenge hinders the progress, i.e., sample inefficiency, due to the setting of binary reward signals (Equation (2)). As a solution to this problem, existing research is central on reward shaping (Ng et al., 1999; Ecoffet et al., 2021; Ding et al., 2023; Trott et al., 2019) and hindsight goal relabelling (Andrychowicz et al., 2017; Fang et al., 2019; Zheng et al., 2022). Reward shaping is a straightforward and efficient idea to address the sparse rewarding problem, while it relies on domain knowledge and complex feature engineering. Hindsight relabelling is inspired by (Schaul et al., 2015b), which introduces a prioritized replay mechanism to improve the sample efficiency of RL methods.

LLMs and Human-AI Interaction. LLMs a class of networks that execute in auto-regressive for text generation tasks. Assuming a given sequence of text tokens is $x_{1:t} = (x_1, \ldots, x_t)$, and the generation of a next token x_{t+1} is formulated as a probabilistic model: $x_{t+1} = \arg\max_{x \in \mathcal{D}} P(x|x_{1:t})$, which satisfies the derivation of Markov chain. As for the training of LLMs, the target is equivalently to find a parameter set θ_{LLM} which satisfies the optimal generation, i.e., $\theta_{\text{LLM}} = \arg\max_\theta P(x_{t+1}|x_{1:t}; \theta)$. With the booming of LLM research (Brown et al., 2020; Touvron et al., 2023; OpenAI, 2023), it is feasible to leverage LLMs as an interface or human behavior model to interact with autonomous agents. This research lies in human-AI interaction (or human-AI coordination/collaboration), and lots of work focuses on achieving collaboration by planning and learning with human behavior models (Nikolaidis & Shah, 2013; Sadigh et al., 2016; Swamy et al., 2020; Carroll et al., 2020). Our work is also related to this direction, as our evaluation is conducted with human-given instructions, so the LLM module in our work acts as an interface for the human-AI interaction. The generalization capability could be a key to human-AI interaction, as we want an agent to collaborate with arbitrary humans. Thus, there is a part of existing work focuses on training a best-response policy to a diverse set of human-like policies so that the agent may generalize to new coming humans (Cui et al., 2023; Strouse et al., 2022; Charakorn et al., 2022). In our work, an agent acquires the generalization capability by achieving open-endedness in the goal space.

3. The Contra: A Battle Royale FPS Game

Contra seamlessly merges the last-man-standing gameplay dynamics with the survival, exploration, and scavenging elements inherent in first-person shooting games (Choi & Kim; Gautam et al., 2021). The game unfolds with multiple hostile teams, necessitating players to collaborate with teammates, withstand adversaries, and strive to outlast others in the ever-changing arena. The agent's objectives encompass individual survival and the elimination of encountered enemies. An agent in Contra mandates a sequential acquisition of skills, starting from fundamental abilities like walking, jumping, running, and item collection. As the learning proceeds, an agent must master more intricate skills such as evading enemy projectiles and coordinating tactics with teammates. This characteristic defines an open-ended learning process where the agent continually explores the game environment to refine mastered skills and acquire new ones.

4. A Co-training Framework: OpenContra

Before delving into the detailed introduction of OpenContra, it is necessary to highlight two critical engineering designs to enhance training efficiency. The training inefficiency is often exacerbated in the context of open-ended learning and large-scale tasks where extensive rollouts are required for experience data collection and evaluations. To address this challenge, OpenContra incorporates a distributed RL framework inspired by AlphaStar (Vinyals et al., 2019) with modifications, resulting in the formation of the *Actor-League-Learner* architecture. In this architecture, the *League* is responsible for distributing rollout tasks to a cluster of *Actors* (CPU nodes) for data collection and evaluation, while optimization tasks are delegated to the *Learner* (GPU node) for policy updates. This distributed approach significantly enhances rollout throughput, thereby improving overall training efficiency. Another efficiency challenge stems from the iterative development of Contra. Unlike most of RL research that often relies on a fixed environment for algorithm development, Contra's environmental attributes continuously change over an eight-month development period. To mitigate the burden of retraining caused by these dynamic changes, we employ surgery (Berner et al., 2019) to retain learned skills at the lowest training cost, enabling adaptation to a changing observation/goal space while ensuring com-

patibility with network inputs. Detailed information on the distributed RL framework can be found in Appendix F, and version changes are documented in Table 8. This section focuses on the algorithmic design aspects of OpenContra.

4.1. Mastering Novel Skills via Non-goal RL

In the realm of GRL, the prevalent approach involves curriculum learning a goal-conditioned policy from scratch. However, achieving goal completion requires an agent to first master basic skills in interacting with the environment. This necessity imposes an additional computational burden on policy learning due to the extensive exploration required. Given the challenges posed by high-dimensional goal exploration, this work diverges from the common practice of leveraging goal decomposition to alleviate the issue. Instead, we opt for an explicit separation of basic skill learning and goal-conditioned learning. A key element in achieving this separation is ensuring that the agent acquires basic skills through non-goal RL before engaging in goal-conditioned learning. To fulfill this objective, we employ Proximal Policy Optimization (PPO) (Schulman et al., 2017), a highly efficient Actor-Critic algorithm (Sutton & Barto, 2018). Furthermore, we introduce a multi-head state value function designed to enhance the efficiency of policy learning, as depicted in Figure 6. This design encompasses two distinct heads: (1) a head dedicated to skill learning and (2) a head focused on learning to avoid obstacles when navigating.

4.2. Pre-training via Curriculum GRL

In the context of encouraging the agent to learn and achieve arbitrary goals in Contra, we define a goal space $\mathcal{G}$ as a Cartesian product of 68 sub-goal spaces: $\mathcal{G} = \mathcal{G}^1 \times \cdots \times \mathcal{G}^{68}$. Each $\mathcal{G}^i$ ($i \in [1, 68]$) is a set $\{g^i|\emptyset, g^i_1, \ldots, g^i_{|\mathcal{G}^i|-1}\}$, where $\emptyset$ indicates that a sub-goal is not selected as a valid value. For a given goal $g \in \mathcal{G}$, it is represented as $g = (g^1, g^2, \ldots, g^{68})$, with $g^i \in \mathcal{G}^i$. The composition of each goal considers two types of features: unit features corresponding to the agent and other players, and environment features that can be determined or affected by agent interactions. Refer to Table 7 for detailed specifications. To encode a goal instance, we use a Residual-Network (He et al., 2016), denoted as $\textsc{ResNet}(\cdot)$, resulting in $\textsc{ResNet}(g)$. This encoded goal is then concatenated with the corresponding state embedding of g and used as input for the policy and value networks.

In the realm of curriculum learning, we explore both automatic and manual paradigms. Automatic curricula, such as Unsupervised Environment Design (UED) (Dennis et al., 2020), were tested with slight modifications, but the agent struggled to explore efficiently. This may be attributed to our goal space being significantly vaster than those tested in previous works, resulting in a sparse regret distribution unsuitable for scheduling goal learning. Consequently, we opt for a manual curriculum, staging goal-conditioned learning through curriculum random and hindsight goal-generation.

Curriculum Random Goal-generation. This approach aims to warm up the agent, facilitating exploration and better comprehension of goal input. Given the expensive cost of exploring the high-dimensional goal space, we gradually activate five dimensions until the full set is activated. Activation for new dimensions is determined by whether the agent has converged on the completion ratio.

Hindsight Goal-generation. While random goal-generation aids exploration, it introduces challenges, including generating many unreasonable goals comprising contradictory sub-goals (Table 3) and encouraging the agent to explore more complex goals. The former arises because random goal-generation is independent of a specified environment context, and the latter is due to the goal-generation independent of the agent policy. To address these issues and promote open-endedness, we propose training a goal generator with Hindsight Replay Buffer (HER) (Andrychowicz et al., 2017), as summarized in Algorithm 2 (refer to Appendix H).

As introduced in the aforementioned, sparse rewarding is a critical issue in Goal-RL, exacerbating inefficient exploration. To mitigate this, we extend the multi-head value function with a goal-completion value head to encourage the agent to complete goals. The corresponding reward is calculated as the Euclidean norm difference between two consecutive states and a goal:

$$r(s_t, a_t, g) = \|g - Proj(s_{t-1})\|_p - \|g - Proj(s_t)\|_p, \quad (3)$$

where $\|\cdot\|_p$ indicates the p-norm. This reward provides a denser signal to the agent about its proximity to the goal, offering more nuanced information than a binary signal indicating whether it has reached the goal or not. In our current implementation, we set $p = 1$.

To further address degeneration, a KL-divergence distance between non-goal and goal-conditioned policies is introduced as a regularizer. The policy learning objective is reformulated as:

$$\max_\theta J(\pi_{g,\theta}) - \mathbb{1}_{g=\emptyset} \cdot D_{KL}(\pi^\star \parallel \pi_{g,\theta}), \quad (4)$$

where $\pi^\star$ represents the optimized non-goal policy, $J(\pi_\theta)$ is the return, and $\mathbb{1}_{g=\emptyset}$ indicates that the KL-divergence term is only activated for an empty goal.

4.3. Multi-step Fine-tuning for Goal Generation

We fine-tune a pre-trained LLM to enable the LLM to generate goals with natural language instructions from set $\mathcal{I}$

and abstracted environmental states from set $\mathcal{O}$ as input. To construct the training dataset $\mathcal{D} := \mathcal{I} \times \mathcal{O} \times \mathcal{G}$, we first build $\mathcal{O}$ a dataset of abstracted state from trajectories collected by the RL agent, where each abstraction $o \in \mathcal{O}$ contains essential features of all players and the present environment state (Appendix B). By accompanying $\mathcal{O}$ with a instruction set $\mathcal{I}$, we form a temporary dataset $\mathcal{D}_x = \{(o, i) | (o, i) \in \mathcal{O} \times \mathcal{I}\}$. Then, we leverage GPT-4 (OpenAI, 2023) to generate appropriate goals for $\mathcal{D}_x$ to construct $\mathcal{D}$. To ensure the goals generated by GPT-4 conform to the format we want, a comprehensive prompt engineering endeavor was conducted to establish a set of predetermined rules for GPT-4. The rule-based prompts that guide GPT-4's responses are meticulously documented in Table 10, and the examples of prompts for generation are listed in Table 17. As for the source of instruction set $\mathcal{I}$, we leverage multiple types of goal generations to ensure its diversity and scale. Specifically, we consider four types, including (1) **HI (Human Instructions)**: constructed with human-annotated commands; (2) **SI (State Instructions)**: GPT-4-generated instructions by giving a pair of states where the first comes from agent trajectories and the second is generated by modifying features of the first item; (3) **AI (Agent Instructions)**: GPT-4-generated instructions by giving agent trajectory and the corresponding initial state; and (4) **RI (Random Instructions)**: a mixture of previous three datasets to form a supplementary dataset. For more details, refer to Appendix E.

With these datasets, we conduct fine-tuning for a ChatGLM-6B (Zeng et al., 2023; Du et al., 2022) with LoRA (Hu et al., 2021) in three steps, as illustrated in Figure 9, including (1) **CoT-assisted fine-tuning (CoFT)**: each question as a task to fine-tune the LLM, aiming to expand the training data volume, enhancing the reasoning and understanding to $\mathcal{D}_x$; (2) **Supervised Fine-tuning (SFT)**: to format the LLM-generated goals strictly, and further enhancing the accuracy; and (3) **Ensemble Fine-tuning (EFT)**: multiple model checkpoints are used to generate goal candidates for each $(o, i) \in \mathcal{D}_x$, then sub-goals with top counts will be used to reconstruct as ground truth to fine-tune the model to improve the precision of goal generation.

4.4. Collaborative Training

After the training in previous steps, the RL agent obtained the capability to complete assigned goals, while the LLM learned to interpret human instructions and translate them into goals in the correct format. In the next stage, we aim to enhance the goal completion ratio corresponding to arbitrary human instructions, thereby achieving open-endedness of human instructions. Since the RL agent and LLM are independently trained, the LLM lacks execution feedback to tune the goal generation, which may result in unreasonability (Table 3), and the RL agent may perform a low completion ratio to the goals generated by the LLM if they were not or less explored in previous training. Thus, introducing co-training aims to resolve the above limitations and make the goals generated by the LLM linguistically sound and not be divorced from the agent's ability to execute them.

The implementation of RLAF relies on PPO with a multi-factor reward that considers multiple aspects to ensure the LLM-based goal generation toward a high completion ratio and consistency with given human instructions. Specifically, they are (1) the evaluation of goal completion: the higher the completion ratio, the higher the reward value; (2) the evaluation of crucial sub-goal completion: for each batch of training data, we set the intersection of instructions in the batch and SI as the examination cases, and pairing each examination instruction with a preset target sub-goal required for successful execution, if the LLM outputs without such a sub-goal, then reward -2, and $+2$ in vice, and further positive rewards are bestowed if a target sub-goal has been achieved; and (3) the evaluation of outputting the proper goal format: we punish the LLM according to edit distance. For more details, refer to Appendix K.

We observed continual training making the LLM and the RL agent tend to compromise to a local optimal, i.e., the LLM outputs goals tend to comfort a high completion ratio for the agent but neglect consistency with human instructions, and the RL agent simultaneously rewards the LLM with a high completion ratio. To fix this issue, we propose a periodic reset of the LLM training, i.e., the parameters of the LLM will be reset for every set number of steps so that the two components can avoid being trapped in a local convergence, achieving enhanced goal completion, and keeping goals consistent with human instructions. Considering the training efficiency of LLMs, we conduct LoRA (Hu et al., 2021) to update the model weights. Figure 9 illustrates the whole training process, and Algorithm 1 summarizes the corresponding pseudo-code.

5. Experiment

We conduct empirical experiments to evaluate the efficacy of both stages of our proposed OpenContra. To make the Contra satisfy the learning requirements, we give well-designed spaces and reward functions as follows.

Observation Space. The observation space encompasses many factors, including unit features detailing the agent's status, those of other players, and monsters. It also encompasses environmental features capturing interaction events and changes in the safe zone. Additionally, an agent-centric RGB bird's-eye-view of the local environment is considered. Due to the heterogeneity in the shape and data type of these features, we process them independently and concatenate them as inputs to both the policy and value network. Detailed information is available in Table 4.

Algorithm 1 COLLABORATIVE TRAINING

Input: θ the parameters of agent policy; β_{llm} the pre-trained parameters of the LLM; β_{sft} the fine-tuned LoRA parameters of the LLM; $\mathcal{I}$ the full instruction set; $\mathcal{I}_S$ the state instruction set
Reload θ, β_{llm}
Merge β_{sft} into β_{llm}
Build new LoRA of LLM, β_{LoRA}
for loop=1, 2, ... **do**
  Initialize parameters of LoRA, β_{LoRA}
  for iteration=1, 2, ..., n **do**
    Agents from a batch of workers send states $\{s_j|j=1,\ldots,m\}$ to LLM
    Random sample a batch of Instructions: $\mathcal{I}_{train}=\{i_j|j=1,\ldots,m\}\subset\mathcal{I}$
    Generate goals in string with LLM: $g_s=\{\text{LLM}(i_j,s_j,\beta_{llm},\beta_{LoRA})|j=1,\ldots,m\}$
    Parse g_s to formatted goals: g
    Distribute goals g to agents with policy π_θ, then collect trajectories with returns

$$(\tau,R)=\{(\tau_j,R_j)|j=1,\ldots,m\},\text{where }\tau_j=\{g_j,s_1,a_1,r_1,\ldots,s_{T_j},a_{T_j},r_{T_j}\},R_j=\sum_{t=1}^{T_j}r_t$$

    Run PPO with (τ,R) to update θ for agent policy
    Filter completed goals g_c from g, extract R the set from (τ,R) as agent feedback rewards R^f
    Compute examination reward by evaluating crucial sub-goal completion: $R^e=\text{REWARD}_e(g_c,\mathcal{I}_S\cap\mathcal{I}_{train},g)$, refer to Appendix K
    Compute format reward: $R^m=\text{REWARD}_m(g_s,g)$, refer to Appendix K
    Update the LLM with PPO: $\beta_{LoRA}=\text{PPO}((\beta_{LoRA},\beta_{llm}),(R^f,R^e,R^m),g_s)$
  end for
end for

Action Space. The action space is implemented on top of Contra's micro-operation API, comprising a collection of multi-grained actions. These actions range from fine-grained movements, such as six-degrees-of-freedom movement and weapon usage, to compound actions in coarse-grained categories, such as firing at a target. The total size of the action space is 54. Further details in Table 6.

Reward Functions. A comprehensive representation is employed for the reward function, considering various factors contributing to policy learning and goal completion. These factors are organized as a linear combination to formulate the reward function. The determination of weights for this combination follows a two-fold principle: (1) assigning weights to reward items based on their scales and emphasizing important factors; (2) dynamically adjusting weights in response to learning feedback, such as decreasing or increasing the weights of corresponding factors. Additional information is available in Appendix A.

5.1. Evaluating Curriculum GRL

We evaluate the curriculum GRL within the OpenContra from three distinct perspectives: (1) the completion ratio, (2) generalization capability concerning unseen goals, and (3) robustness when integrating goal-conditioned learning atop non-goal learning. Given that the curriculum GRL in OpenContra comprises random and hindsight stages, our evaluation involves a comparative analysis with two baselines, namely, (1) RANDOM: training the RL agent with randomly generated goals, and (2) HINDSIGHT: training the RL agent with hindsight goal generation. For clarity, we refer to our method as HINDSIGHT FROM RANDOM.

Figure 2(a) presents a comparison of the goal completion ratio across different pretraining methods on a validation dataset where goals are generated using random and hindsight goal generators. As depicted in Figure 2(a), HINDSIGHT FROM RANDOM surpasses all baselines by 4% and 1%, respectively. Figure 2(b) evaluates the generalization on unseen goals, addressing the second aspect mentioned earlier. It is noteworthy that the unseen goals are recombinations of goals obtained with HINDSIGHT and LLM. As indicated in Figure 2(b), HINDSIGHT FROM RANDOM excels over the other two baselines in terms of completion ratio. Figure 2(c) answers the third point by comparing the use of KL-divergence regularizer for policy learning, considering changes in overall performance and the ability to eliminate enemies. Three metrics are designed for evaluation: (1) Mean basic reward per step, which indicates whether the current policy degenerates in performing basic skills per step against a well-trained non-goal policy, and

intentional to emphasize the agent's immediate responsiveness over final results; (2) #Enemies killed, representing the average number of enemies killed by the agent per episode; and (3) #Enemies knocked down, representing the average number of enemies knocked down by the agent per episode.

5.2. Evaluating LLM-based Goal Generation

We assessed the performance of the LLM through two comparative experiments on a GPT-4 generated instruction dataset, aiming to investigate the impact of different dataset construction and fine-tuning paradigms. Evaluation metrics employed include precision, recall, and F1 score. It's worth noting that determining the precision of LLM outputs for certain sub-goals, such as associating the sub-goal "moving speed" with "very fast" versus "fast" poses challenges, as it might be perceived as a negative instance under precision measurement. Consequently, we assert that generating such sub-goals should weigh more heavily in performance evaluation than specific values, necessitating the introduction of corresponding metrics. As a solution, we propose three choice-based metrics: precision (choice), recall (choice), and F1 (choice), to address this requirement. Table 1 provides a comparison of five types of training

Dataset	Precision	Precision (Choice)	Recall	Recall (Choice)	F1	F1 (Choice)
HI	0.435	0.611	0.361	0.517	0.395	0.560
AI	0.474	0.611	0.419	0.532	0.445	0.569
SI	0.444	0.601	0.413	0.539	0.428	0.568
RI	0.499	0.633	0.414	0.526	0.453	0.574
ALL	**0.555**	**0.685**	**0.505**	**0.621**	**0.529**	**0.652**

Table 1: Evaluation on different datasets.

datasets used in the three-stage fine-tuning process for the LLM, with "ALL" representing the proportional mixture of the four base datasets. The comparison reveals that utilizing a mixture significantly outperforms individual base datasets, which indicates a mixture aids the LLM in capturing human command habits, understanding the implications of each abstracted state, and refining the initial execution capability of the policy from various perspectives, thereby enhancing goal generation.

Tuning	Precision	Precision (Choice)	Recall	Recall (Choice)	F1	F1 (Choice)
SFT	0.547	0.663	0.490	0.602	0.517	0.632
CoTF	0.533	0.652	0.487	0.599	0.509	0.624
CoTF → SFT	0.555	0.685	**0.505**	**0.621**	**0.529**	**0.652**
CoTF → SFT→EFT	**0.590**	**0.694**	0.501	0.593	0.516	0.629

Table 2: Evaluation on different tuning methods.

Table 2 compares four kinds of fine-tuning approaches with the three-staged fine-tuning introduced in Section 4.3, including (1) SFT: only use the target prompt without CoT data to supervised fine-tuning, which can be regarded as a baseline for a general SFT approach; (2) CoTF: only CoT-assisted fine-tuning; (3) CoTF → SFT: further SFT target prompt after CoTF; (4) CoTF → SFT→EFT: further ensemble fine-tuning target prompt after CoTF. With the comparison results, we conclude that CoTF and SFT can improve each other and achieve better performance. Furthermore, ensemble fine-tuning significantly enhances precision while marginally decreasing recall, making it more suitable for generating accurate concise goals.

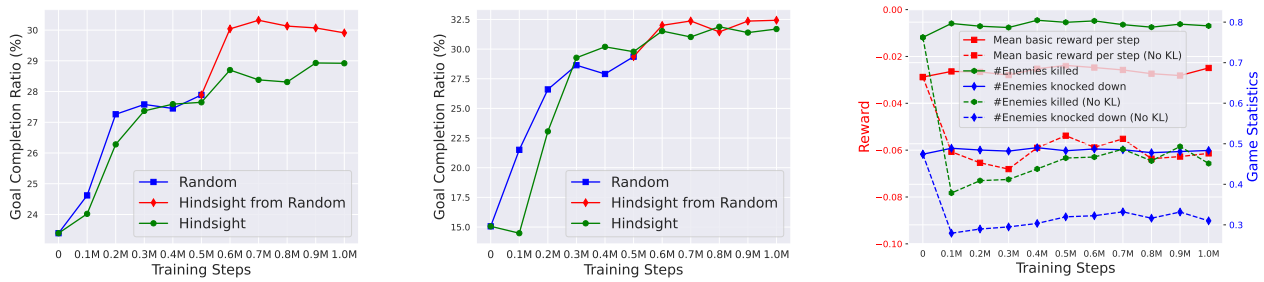
5.3. Evaluating Co-training

We conducted an analysis of the completion ratio corresponding to different goal dimensions during the co-training process, as illustrated in Figure 3(a). It is evident that the completion ratio for human instructions is relatively low when the agent and the LLM have not been co-trained initially. However, as the co-training proceeds, the completion ratio gradually improves, particularly for goals with dimensions ranging from 2 to 4. This indicates that co-training effectively enhances the LLM's understanding of the agent's capabilities and preferences. Conversely, as depicted in Figure 3(b), the initial completion ratio for each training loop increases with each reset, suggesting that co-training enhances the policy's ability to execute the language model's preferred outputs. Furthermore, the combined improvements in the LLM and the agent effectively enhance the ability to execute human instructions.

Instruction	Goal(Before co-training)	Goal(After co-training)
Stop!	Whether prone position: True Average velocity: Static	Average velocity: Static Whether prone position: True Length of distance moved: No Movement
Get down, stay hidden.	Whether prone position: True Average velocity: Static	Whether prone position: True Length of distance moved: No Movement Average velocity: Static Whether seen by enemy: False
Enemy! Rush and fire.	Whether hold a gun: True Whether have bullets: True Horizontal direction of view: Southwest Whether seen enemy: True Average velocity: Fast	Length of distance moved: long Average velocity: Fast Whether hold a gun: True Horizontal direction of movement: Southwest Whether seen enemy: True Damage to enemy: High Horizontal direction of view: Southwest
Enemies nearby, move to defend and avoid damage.	Whether prone position: True Average velocity: Fast Whether hold a gun: True Health level: Full Whether to restore health: True	Length of distance moved: long Average velocity: Fast Whether prone position: False Horizontal direction of movement: North (Position of enemy: South) Whether hold a gun: True

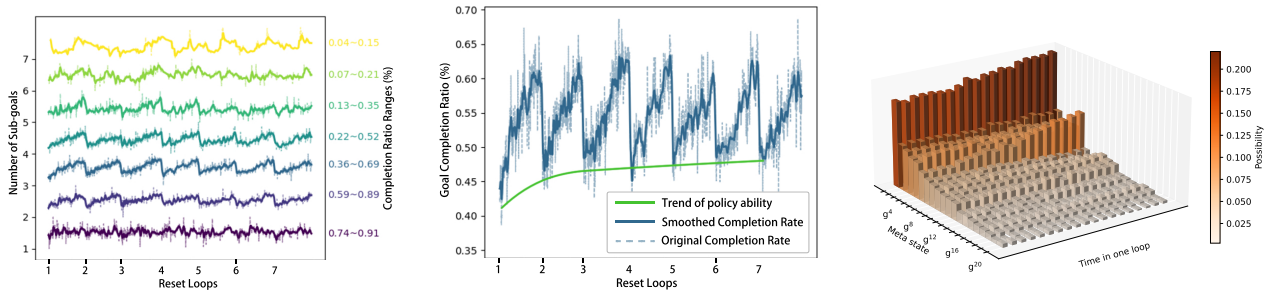
Table 3: Comparison of LLM goal generation. Cyan the helpful, pink the conflicting, and orange the critical sub-goals. The comparison shows that co-training enables goal-generation to avoid conflicts of sub-goals and improves reasonability by including helpful and critical sub-goals.

Additionally, we investigated the change in sub-goal distribution during co-training, as depicted in Figure 3(c) and Figure 11(a). The former illustrates changes within a loop, while the latter indicates changes across loops. The associated explanations for each G_i are provided in Table 14. As



(a) (b) (c)

Figure 2: (a) The goal completion rate on training set; (b) The goal completion rate on unseen goals; (c) The evaluation of policy learning in cases of w/ and w/o KL-divergence regularizer.



(a) Goal completion rate ($1 \leq |g| \leq 7$) (b) Goal completion rate ($|g| = 3$) (c) Sub-goals distribution

Figure 3: (a) The completion ratio of goals with dimension size ranges from 1 to 7; (b) The goal completion ratio of goals with dimension size smaller than 3; (c) The sub-goals distribution changes along the training in one loop of co-training, where the description of each G_i is included in Table 14.

training progresses, the probabilities associated with each sub-goal undergo gradual modifications. For instance, sub-goals with augmented probabilities include variables such as movement speed and the prone state due to their relatively attainable nature and influence in directing the agent toward accomplishing other objectives. Conversely, sub-goals with diminished probabilities encompass inter-agent distances and enemy visibility, which are linked to interactions with other agents and are not directly related to fulfilling linguistic instructions. The language model tends to generate outputs for these sub-goals only when absolutely necessary.

As a case study, we have identified various instances in which objectives have been modified following the co-training process, as shown in Table 3. Evidently, after collaborative training, the LLM demonstrates its capacity to eliminate contradictory and irrational elements within the initial objectives. Furthermore, it exhibits the ability to introduce new objective components, thereby rendering the overall goal more attainable, all while retaining its exceptional semantic comprehension capabilities. These findings underscore the deepened comprehension of the linguistic model regarding the environment achieved through collaborative training.

6. Conclusion

In this paper, we implement a co-training framework named OpenContra experts on learning an open-ended embodied agent, which leverages large language models, goal-conditioned reinforcement learning, and distributed learning to fill the blank of cooperating LLM and RL for efficient open-ended learning. We take a battle royale FPS game, Contra, as its testbed to support our claim, and the empirical results represent that OpenContra shows the potential as a practical solution. Despite the positive results, we admit there are still some limitations to our work that would be expected to be researched in the future—for instance, a truly open-ended goal description instead of the handcrafted goal space in the current version; supporting multi-modality input/output to free from expensive feature engineering.

Author Contribution Statement

The authors confirm their contribution as follows:

Shaopeng Zhai: team leadership, open-ended learning, LLM/RLAF training, agent analysis, architecture design
Jie Wang: infrastructure/framework engineering, non-goal agent training, open-ended learning, ablation studies, feature engineering
Tianyi Zhang: non-goal agent training, open-ended learning, feature engineering
Fuxian Huang: non-goal agent training, paper writing, open-ended learning
Qi Zhang: LLM training, RLAF training, paper writing, ablation studies
Ming Zhou: co-training framework, curriculum research, paper writing
Jing Hou: LLM training, paper writing

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. *Advances in neural information processing systems*, 30, 2017.

Balduzzi, D., Garnelo, M., Bachrach, Y., Czarnecki, W., Perolat, J., Jaderberg, M., and Graepel, T. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pp. 434–443. PMLR, 2019.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. Dota 2 with large scale deep reinforcement learning, 2019.

Bousmalis, K., Vezzani, G., Rao, D., Devin, C., Lee, A. X., Bauza, M., Davchev, T., Zhou, Y., Gupta, A., Raju, A., et al. Robocat: A self-improving foundation agent for robotic manipulation. *arXiv preprint arXiv:2306.11706*, 2023.

Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on Robot Learning*, pp. 287–318. PMLR, 2023.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

Campero, A., Raileanu, R., Kuttler, H., Tenenbaum, J. B., Rocktäschel, T., and Grefenstette, E. Learning with amigo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2020.

Carroll, M., Shah, R., Ho, M. K., Griffiths, T. L., Seshia, S. A., Abbeel, P., and Dragan, A. On the utility of learning about humans for human-ai coordination, 2020.

Charakorn, R., Manoonpong, P., and Dilokthanakul, N. Generating diverse cooperative agents by learning incompatible policies. In *ICML 2022 Workshop AI for Agent-Based Modelling*, 2022. URL https://openreview.net/forum?id=a7vLnGKGIjY.

Choi, G. and Kim, M. Battle royale game: In search of a new game genre. *IJCT*, pp. 5.

Cui, B., Lupu, A., Sokota, S., Hu, H., Wu, D. J., and Foerster, J. N. Adversarial diversity in hanabi. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=uLE3WF3-H_5.

Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., and Levine, S. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33: 13049–13061, 2020.

Ding, H., Tang, Y., Wu, Q., Wang, B., Chen, C., and Wang, Z. Magnetic field-based reward shaping for goal-conditioned reinforcement learning. *IEEE/CAA Journal of Automatica Sinica*, 10(12):1–15, 2023.

Du, Y., Watkins, O., Wang, Z., Colas, C., Darrell, T., Abbeel, P., Gupta, A., and Andreas, J. Guiding pretraining in reinforcement learning with large language models. *arXiv preprint arXiv:2302.06692*, 2023.

Du, Z., Qian, Y., Liu, X., Ding, M., Qiu, J., Yang, Z., and Tang, J. Glm: General language model pretraining with autoregressive blank infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 320–335, 2022.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847): 580–586, 2021.

Fang, M., Zhou, T., Du, Y., Han, L., and Zhang, Z. Curriculum-guided hindsight experience replay. *Advances in neural information processing systems*, 32, 2019.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

Gautam, A., Jain, H., Senger, A., and Dhand, G. Battle royale: First-person shooter game. In *Proceedings of the International Conference on Innovative Computing & Communication (ICICC)*, 2021.

Groth, O., Hung, C.-M., Vedaldi, A., and Posner, I. Goal-conditioned end-to-end visuomotor control for versatile skill primitives. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1319–1325. IEEE, 2021.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hu, B., Zhao, C., Zhang, P., Zhou, Z., Yang, Y., Xu, Z., and Liu, B. Enabling intelligent interactions between an agent and an llm: A reinforcement learning approach, 2023.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. Lora: Low-rank adaptation of large language models, 2021.

Jaderberg, M., Czarnecki, W. M., Dunning, I., Marris, L., Lever, G., Castaneda, A. G., Beattie, C., Rabinowitz, N. C., Morcos, A. S., Ruderman, A., et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Liu, M., Zhu, M., and Zhang, W. Goal-conditioned reinforcement learning: Problems and solutions. *arXiv preprint arXiv:2201.08299*, 2022.

Lynch, C. and Sermanet, P. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., and Pathak, D. Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34:24379–24391, 2021.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6292–6299. IEEE, 2018.

Nair, S. and Finn, C. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation. In *International Conference on Learning Representations*, 2019.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. Curriculum learning for reinforcement learning domains: A framework and survey. *The Journal of Machine Learning Research*, 21(1):7382–7431, 2020.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pp. 278–287. Citeseer, 1999.

Nikolaidis, S. and Shah, J. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pp. 33–40, 2013. doi: 10.1109/HRI.2013.6483499.

OpenAI. Gpt-4 technical report, 2023.

Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744, 2022.

Parker-Holder, J. *Towards truly open-ended reinforcement learning*. PhD thesis, University of Oxford, 2022.

Pitis, S., Chan, H., Zhao, S., Stadie, B., and Ba, J. Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, pp. 7750–7761. PMLR, 2020.

Ren, Z., Dong, K., Zhou, Y., Liu, Q., and Peng, J. Exploration via hindsight goal generation. *Advances in Neural Information Processing Systems*, 32, 2019.

Sadigh, D., Sastry, S., Seshia, S. A., and Dragan, A. D. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and systems*, volume 2, pp. 1–9. Ann Arbor, MI, USA, 2016.

Samvelyan, M., Khan, A., Dennis, M. D., Jiang, M., Parker-Holder, J., Foerster, J. N., Raileanu, R., and Rocktäschel, T. Maestro: Open-ended environment design for multi-agent reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2022.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In *International conference on machine learning*, pp. 1312–1320. PMLR, 2015a.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms, 2017.

Stanley, K. O., Lehman, J., and Soros, L. Open-endedness: The last grand challenge you've never heard of. *While open-endedness could be a force for discovering intelligence, it could also be a component of AI itself*, 2017.

Strouse, D., McKee, K. R., Botvinick, M., Hughes, E., and Everett, R. Collaborating with humans without human data, 2022.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Swamy, G., Schulz, J., Choudhury, R., Hadfield-Menell, D., and Dragan, A. On the utility of model learning in hri, 2020.

Team, O. E. L., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023.

Trott, A., Zheng, S., Xiong, C., and Socher, R. Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards, 2019.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Wagner, R. A. and Fischer, M. J. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1): 168–173, 1974.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Warde-Farley, D., Van de Wiele, T., Kulkarni, T., Ionescu, C., Hansen, S., and Mnih, V. Unsupervised control through non-parametric discriminative rewards. In *International Conference on Learning Representations*, 2018.

Weng, L. Curriculum for reinforcement learning. *lilianweng.github.io*, Jan 2020. URL https://lilianweng.github.io/posts/2020-01-29-curriculum-rl/.

Zeng, A., Liu, X., Du, Z., Wang, Z., Lai, H., Ding, M., Yang, Z., Xu, Y., Zheng, W., Xia, X., Tam, W. L., Ma, Z., Xue, Y., Zhai, J., Chen, W., Liu, Z., Zhang, P., Dong, Y., and Tang, J. GLM-130b: An open bilingual pre-trained model. In *The Eleventh International Conference on Learning Representations (ICLR)*, 2023. URL https://openreview.net/forum?id=-Aw0rrrPUF.

Zheng, Q., Zhang, A., and Grover, A. Online decision transformer. In *international conference on machine learning*, pp. 27042–27059. PMLR, 2022.

A. Contra: The Environment

Contra seamlessly merges the last-man-standing gameplay dynamics with the survival, exploration, and scavenging elements inherent in first-person shooting games (Choi & Kim; Gautam et al., 2021). The game unfolds with multiple hostile teams, necessitating players to collaborate with teammates, withstand adversaries, and strive to outlast others in the ever-changing arena. The agent's objectives encompass individual survival and the elimination of encountered enemies. An agent in Contra mandates a sequential acquisition of skills, starting from fundamental abilities like walking, jumping, running, and item collection. As the learning proceeds, an agent must master more intricate skills such as evading enemy projectiles and coordinating tactics with teammates. This characteristic defines an open-ended learning process where the agent continually explores the game environment to refine mastered skills and acquire new ones. Figure 4 illustrates a map of Contra, which has diverse terrains such as plains, deserts, and snow-capped mountains.

Figure 4: A map in FPS game.

Observation Space. The observation space encompasses various factors, comprising unit features delineating the agent's status, as well as that of other players. Additionally, it includes environmental features characterizing interaction events and alterations in safe-zone configurations. Furthermore, an agent-centric RGB bird's-eye-view of the local environment is incorporated. Given the heterogeneity in the shapes and data types of these features, we adopt an independent processing approach, subsequently concatenating them to serve as input for the policy and value networks. Figure 5 illustrates the

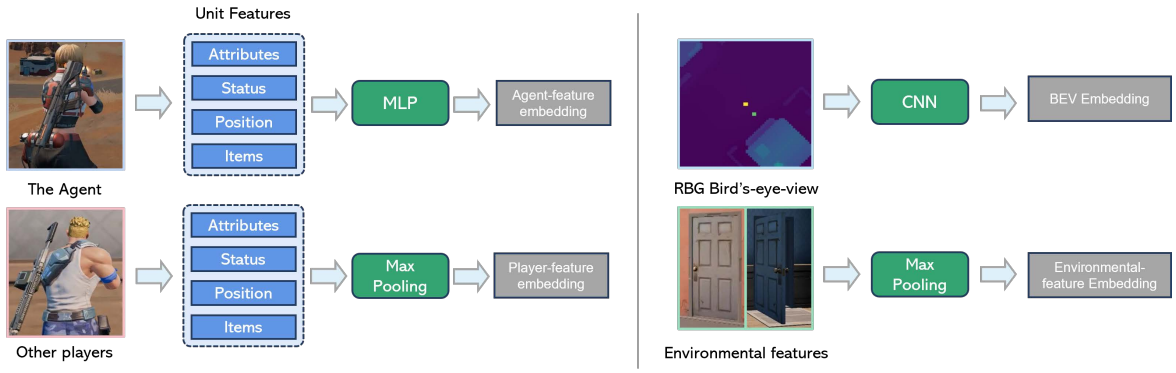


Figure 5: Preprocessing for an observation with four types of features.

network for preprocessing an observation instance in our cases, where an observation includes four types of features as listed in Table 4, and we leverage independent encoding for each of them. Furthermore, considering the iterative development of the environment, it is convenient to leverage surgery with this network architecture to handle these changes.

Action Space. As introduced in Table 6, the instantiation of the action space is achieved through the utilization of the micro-operation API within the Contra framework. This process gives rise to a compilation of actions characterized by diverse levels of granularity, including fine-grained actions, alongside a suite of compound actions encompassing coarse-grained activities. In a detailed breakdown, the action space comprises several distinct types, namely movement direction, yaw direction, pitch direction, body action, basic action, and switch weapon. The movement direction action provides 16 discrete choices, each evenly distributed across a 360-degree spectrum. Likewise, the yaw direction action offers 16 choices with an equitable division of the 360-degree range. The pitch direction action encompasses three distinct values: -45, 0, 45. The body action incorporates nine diverse values: slide, stop, crouch, run, jump, ground, open or close door, rescue, and none. The basic action presents seven different values: fire, reloading, treat, pick up supply, drop supply, stop and fire, stop adjust and fire. Finally, the switch weapon action manifests three values: switch slot 0, switch slot 1, and none. The aggregated dimensionality of the action space is quantified at 54 in total.

Reward Weights. The primary objective of our training regimen is to equip the agent with the ability to play with other players in Contra while concurrently optimizing its success in overcoming opponents. To achieve this objective, we have formulated a diverse array of rewards designed to guide the agent's learning trajectory. However, the complexity involved in designing and fine-tuning these rewards is evident. Thus, our approach to reward weight design is characterized by a two-fold set of principles. Firstly, we allocate weights based on the expected value of each reward, ensuring a proportionate influence on the learning process. Secondly, we integrate a mechanism for smooth weight adjustments, facilitating dynamic and gradual modifications to the reward weightings. These principles collectively contribute to the construction of the reward function, learning an agent policy conducive to the desired optimal performance against opponents. In accordance with the first principle, we assume a referenced maximum return of 20, with different rewards assigned proportions based on their relative importance. For critical actions such as knocking down or killing an enemy, their values are set to approximately 4 (20% out of 20). Conversely, for less critical actions like scouting or incurring time penalties, their values are set to less than 1 (5% out of 20). It is crucial to highlight that the value estimation is derived through the discounted reward, with a discounting coefficient (γ) set to 0.995. Concerning the second principle, throughout the training process, emphasis may be placed on learning specific skills during certain periods. For instance, the coefficient associated with a skill can be gradually amplified before mastery and subsequently reduced after proficiency is achieved. Detailed specifications are outlined in Table 5. In accordance with the aforementioned principles, the reward function has been constructed to systematically amalgamate multiple factors in a linear combination, facilitating their collaborative influence on guiding policy learning. As delineated in Algorithm 2, these factors are broadly classified into three categories: fundamental rewards r^{basic}, obstacle avoidance rewards r^{oa}, and goal achievement reward r^{g}. The fundamental rewards are primarily directed at steering the agent towards enhanced game-playing performance, encompassing collaborative engagement with teammates and eliminating adversaries, etc. In the case of r^{oa}, the objective is to promote natural navigation and forestall the agent from encountering obstacles, such as stones and trees. Regarding the implementation, penalties are imposed on the agent for deviations from the optimal path. This optimal trajectory is determined by assessing the cosine similarity between the agent's current movement direction, a 3D unit vector, provided as an environmental signal, and the ideal obstacle-free trajectory derived from the action sequence in the trajectory:

Feature Class	Field	Description	Dimension
1.Unit feature	Scalar	Includes heroes, teammates, enemies, monster	527
Heroes	Attribute	Character ID,team ID,size,skills	28
	Status	HP, oxygen,speed, peek type, alive state, body state,etc.	44
	Pose	Position, rotation, camera position, camera rotation, etc.	76
	Item	Backpack,weapon	144
Teammates	Attribute	Character ID,team ID,size,skills	28
	Status	HP, oxygen,speed, peek type, alive state, body state,etc.	30
	Pose	Position, rotation, camera position, camera rotation, etc.	43
Enemies	Attribute	Character ID,team ID,size,skills	28
	Status	HP, oxygen,speed, peek type, alive state, body state,etc.	33
	Pose	Position, rotation, camera position, camera rotation, etc.	43
Monsters	Attribute	Monster type,size	8
	Status	HP, max HP, HP percent, target type	6
	Pose	Position, rotation, relative position, distance	16
2.Global feature	Scalar	Includes circle, event, door and supply	65
Circle	Status	State,pain,radius	4
	Position	Blue circle, white circle	6
	Time	existence time, rest time, total time, delay time, appear time	5
Event	Attribute	Type,damage,elapsed time	8
	Position	Occurred position	3
Door	Attribute	Type,state	5
	Pose	Position,relative position,rotation	8
Supply	Status	Type,size,quantity,etc.	19
	Position	Position,relative position,distance	7
3.Invisible enemy feature	Scalar	Invisible (nearby) enemy feature only for value estimation	104
Invisible enemies	Attribute	Character ID,team ID,size,skills	28
	Status	HP,oxygen,speed, peek type, alive state, body state,etc.	33
	Pose	Position, rotation, camera position, camera rotation, etc.	43
4.Spatial feature	Scalar	Heatmap	12288
Heatmap	Region	Altitude map and aerial view map	3x64x64

Table 4: The details of features in the observation space.

$$r_t^{oa} = -\frac{\mathbf{d}_t^{env} \cdot \mathbf{d}_t^{ideal}}{\|\mathbf{d}_t^{env}\|_2 * \|\mathbf{d}_t^{ideal}\|_2}, \tag{5}$$

where $\mathbf{d}_t^{env}$ the actual movement direction of the agent, $\mathbf{d}_t^{ideal}$ the ideal movement direction, which is derived by combining

the ideal movement direction from the previous moment with the movement action taken at the current moment:

$$\mathbf{d}_t^{ideal} = \Phi(\mathbf{d}_{t-1}^{ideal}, a_t) \tag{6}$$

To address the issue of the agent getting stuck on obstacles due to short-term action sequences, we employ a smaller γ for the corresponding value head. Specifically, we set the γ value to 0.92. This adjustment helps mitigate the impact of the obstacle avoidance reward on long-term credit assignment, allowing for a more balanced consideration of immediate and future rewards in the agent's decision-making process. As for the goal-achieving reward, we've introduced in the main text,

name	**reward weights**	**description**
discover enemy	0.02	reward for see an enemy
be discovered by enemy	-0.002	punishment for being seen by an enemy
scout	0.0001	reward for search for an enemy
no-op	-0.0002	punishment for stopping and doing nothing
bullet	0.015	reward for using and refilling bullets
health point	0.03	reward for health point changes
be knocked down	-2.5	punishment for being knocked down
dead	-3.5	punishment for being killed
damage enemy	0.1	reward for damaging an enemy
knock down enemy	4.5	reward for knocking down an enemy
kill enemy	3.5	reward for killing an enemy
approach a downed teammate	0.001	reward for approaching a downed teammate
help up a downed teammate	0.8	reward for helping up a downed teammate
not save a downed teammate	-0.5	punishment for not saving a downed teammate
go to blue circle	0.00015	reward for going to blue circle
be in white circle	-0.00005	small punishment for being outside of white circle
outside blue circle	-0.012	punishment for being outside of blue circle
teammate damage enemy	0.03	reward from teammate damaging enemies
teammate get up	0.6	reward from teammate getting up
I help teammate up	4	reward for helping teammate up
interrupt helping teammate up	-0.05	punishment for the interruption to help teammate up
obstacle avoidance	0.012	punishment for being stuck
goal	1	reward of completing a goal

Table 5: The introduction of different rewards.

please refer to Section 4.2.

B. Environment Abstraction and Goal Space

action space	**54**
movement direction	16
yaw direction	16
pitch direction	3
body action	9
basic action	7
switch weapon action	3

Table 6: Action space.

For a comprehensive understanding of the game environment, a language model undergoes a fine-tuning process due to the scarcity of textual information within the simulation environment. The need arises to articulate non-linguistic elements, and the interaction between an agent and the simulation environment is centered on continuously observing the environment's state and generating corresponding actions. Therefore, the key aspects requiring verbalization primarily involve the state and actions of the agent. However, given the abundance of possible observation states in the simulation environment, it is impractical to use all of these states directly as prompts for the language model, especially considering token limitations. Consequently, there is a crucial need to extract and linguistically transform the most significant meta-states to facilitate successful model interaction. It is noteworthy that smaller language models have limitations in comprehending and manipulating numerical values effectively. To address this challenge, a deliberate effort is made to minimize the use of numerical values during the environmental linguistics process. For example, instead of specifying an agent's speed with specific numeric metrics like "speed: 1m/s $\rightarrow$ 3m/s" a qualitative representation such as "speed: slower $\rightarrow$ faster" is adopted. This technique transforms the original

Goal name	Values
Damage to enemy	[Zero,Low,Little low,Medium,Little high,High]
Whether knock down enemy	[True,False]
Whether kill enemy	[True,False]
Whether seen enemy	[True,False]
Whether seen by enemy	[True,False]
Number of enemies have ever seen	[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
Length of distance moved	[No movement,Short,Medium,Long,Very long]
Average velocity	[Static,Slow,Medium,Fast,Falling]
Horizontal direction of movement	[West,Northwest,North,NorthEast,East,Southeast,South,Southwest]
Horizontal direction of view	[West,Northwest,North,NorthEast,East,Southeast,South,Southwest]
Pitch direction of view	[Low,Little low,Medium,Little high,High]
Health level	[Empty,Low,Medium,High,Full]
Whether to restore health	[True,False]
Whether the health is damaged	[True,False]
Whether rescued teammate	[True,False]
Whether be knocked down	[True,False]
Whether prone position	[True,False]
Whether have a gun	[True,False]
Whether have bullets	[True,False]
Whether have medical kits	[True,False]
Distance with nearest enemy	[Touch,Nearby,Moderate,Far,Out of reach,Extreme Far]
Whether closer with nearest enemy	[True,False]
Whether crouch position	[True,False]
Whether hold a gun	[True,False]
Length of distance from agent to teammate	[Touch,Nearby,Moderate,Far,Out of reach,Extreme Far]
Whether seen by teammate	[True,False]
Teammate's position relative to agent	[West,Northwest,North,NorthEast,East,Southeast,South,Southwest]
Whether follow with the views of teammate	[True,False]
Whether target the same enemy as teammate	[True,False]
Whether follow with the movement direction of teammate	[True,False]
Horizontal direction of movement of enemy	[West,Northwest,North,NorthEast,East,Southeast,South,Southwest,None]
Velocity of enemy	[Static,Slow,Medium,Fast,Falling,None]
Enemy's position relative to agent	[West,Northwest,North,NorthEast,East,Southeast,South,Southwest,None]

Table 7: The items in goal space.

continuous state into a limited, discrete meta-state, thereby enhancing the language model's understanding. Similarly, for expediting language model understanding, a discrete action space is adopted, with each action accompanied by a succinct artificial language description. This discreet articulation of actions contributes to the overall interpretability of the language model within the simulation environment. We list the details in Table 7.

C. RL Network Architecture

Figure 6 shows the network architecture used for reinforcement learning. On top of observation preprocessing in Appendix A, we introduce a backbone implemented with a fully-connected layer followed by three Residual Blocks. As for the policy head and three value heads, we implemented each of them as two connected Residual Blocks. It is noteworthy that the invisible enemy information, such as the nearest enemy's location, has also been introduced as an input to the value estimation, for the consideration of stabilizing the policy learning (Vinyals et al., 2019).

Date	Iteration	#params	Change
4/14/2023	1	1802702	Experiment started
4/27/2023	1808552	1802702	Env-init: Random weapons
5/8/2023	2829170	1803087	Action: Add a fire action for long distance
5/10/2023	3034011	1803087	Env-init:Random safe area in the whole map
5/11/2023	3130353	1803855	Observation: Add number of remaining players in the game
5/12/2023	3198564	2412975	Observation: Add heatmap feature
5/16/2023	3673506	2418111	Observation: Add history rotation feature
5/22/2023	4519567	2418368	Observation: Add rotation change feature
5/29/2023	5442025	2418368	Reward: Add rewards for teamwork
6/2/2023	5899503	2418368	Update new game version
6/13/2023	7306607	3013409	Network: Add obstacle avoidance reward and corresponding value head
6/14/2023	7404118	3015457	Observation: Add distance feature to nearby obstacles
6/16/2023	7628098	3015457	Env-init: Player numbers per team increased to 4
6/19/2023	7974450	3109267	Action: Use attention to select target to attack

Table 8: The essential surgeries for major changes in the training procedure.

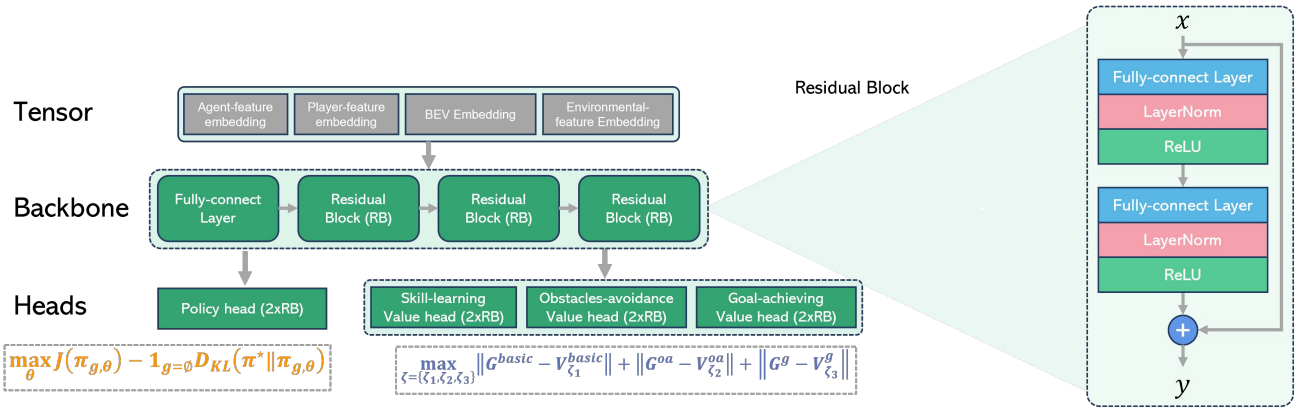


Figure 6: Network structure of our proposed policy.

D. Surgery

As the project proceeded, Contra was continuously improved to satisfy richer featured environment dynamics. However, such an iterative development poses some challenges to the research of open-ended learning in an embodied situation, as the changes in API and environment attributes will make the training be non-stationary. A popular solution to resolve this issue is the surgery introduced by (Berner et al., 2019), which significantly reduces training time by maximizing retention of previously learned abilities. Similarly, we leverage surgery in four aspects to ensure the training adapts to the new changes, including model architecture, observation space, action space, and reward functions. Table 8 illustrates the main changes we conducted and the corresponding parameters. For the surgery of observation space and model architecture, we have introduced a decoupled encoding in Appendix A; for the surgery of action space, we directly extend the policy head in width to satisfy the new action space; for the reward functions, the essentials are to include the newly introduced features which can contribute to the learning, as we introduced in Appendix A, a linear combination has been considered to satisfy this requirement. In our experiment, we propose three novel surgery methods, where two for model architecture and one for observation space. The game environment has changed several times since the training started. The changes are mainly about adding player characters, adding player skills, adding weapon, modifying the map, etc. For all these changes, the proportion of new environments in which we train our policy grows slowly from 0% to 100%. In this case, the variance is relatively small and the performance would quickly recover in the new environment. Figure 7 evaluates the utility of surgery,

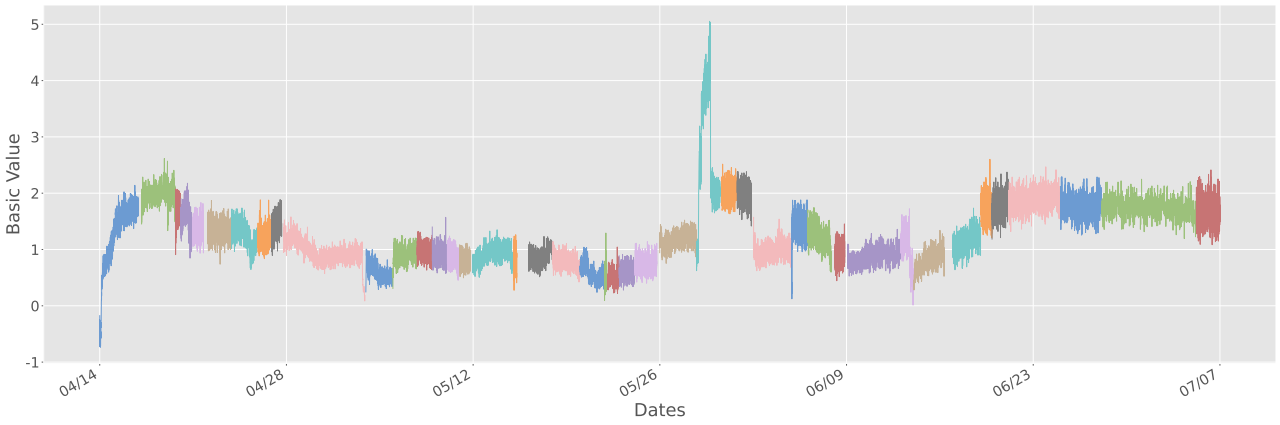


Figure 7: The value changes during the training process.

illustrating the changes in basic value during the training. It can be seen that the values change smoothly for most surgeries. Meanwhile, the values remain stable after the surgeries. These results prove the effectiveness of our surgery.

E. Datasets Construction

The process of fine-tuning the language model is operationalized through a question and answer paradigm. In this framework, we provide the language model with a comprehensive depiction of the present conditions pertaining to the agent, its companions, and adversaries. Additionally, we furnish the model with the linguistic instructions furnished by its teammates. Subsequently, the language model is tasked with generating the agent's subsequent target meta state in accordance with the specifics outlined in the question and answer dataset, as elucidated in Table 17. The response is generated by GPT-4 and subsequently subjected to parsing and formatting processes facilitated by the rule-based coding mechanism. To ensure alignment between the responses generated by GPT-4 and the targeted meta-state format, a comprehensive prompt engineering endeavor was conducted to establish a set of predetermined rules for GPT-4. The rule-based prompts, which were employed to guide GPT-4's responses, are meticulously documented in Table 10.

1. Analyze the verbal orders of teammates and players, what do you want to do? According to the command, also analysis the relevant states of teammates and enemies that need attention. The verbal command of the teammate player is 'You should lie in wait', which means teammate player wants the agent to set up an ambush or take a hiding position.
2. Analyze which states of the agents are most relevant to the verbal commands of teammate player. The agents in the unselected states will adjust themselves to complete your plan. According to the teammate'S command: 2.1. Need to hide: 'Whether prone position', 'Whether crouch position' 2.2. Reduce movement to stay unnoticed: 'Average velocity', 'Length of distance moved' 2.3. Ensure readiness for potential attack: 'Whether hold a gun'
3. Plan how these key states need to be adjusted. According to the teammate'S command: 3.1. 'Whether prone position': Need to hide: 'False' -> 'True' 3.2. 'Whether crouch position': Alternative hiding posture if not prone: 'False' -> 'True' 3.3. 'Average velocity': Reduce movement: 'Fast' -> 'Static' 3.4. 'Length of distance moved': Reduce movement: 'Short' -> 'No movement' 3.5. 'Whether hold a gun': Ensure readiness for potential attack: 'True' -> 'True'
4. Modify the adjustment that may be wrong, and refer to the Rules to analyze which state adjustments may conflict, repeat or be unnecessary, and output the modified adjustment plan: According to the states adjustments in 3: 4.1. 'Whether prone position': 'False' -> 'True' 4.2. 'Average velocity': 'Fast' -> 'Static' 4.3. 'Length of distance moved': 'Short' -> 'No movement' 4.4. 'Whether hold a gun': 'True' -> 'True' (Removed 'Whether crouch position' as it conflicts with 'Whether prone position' and prone position gives better cover for an ambush.)
5. According to the analyze and the planing of the verbal command, further analyze the behavior tendency required in the adjustment process. Mobile: Need to stop all movement: 0% Offense: Must be ready to attack if necessary: 20% Waiting: The main goal is to set up an ambush and wait: 70% Supplies: No need for supplies during this phase: 0% Scouting: Need to observe the surroundings, but not actively scout: 10%
6. Analyze how long the current command needs to be kept. According to the command of the teammate, taking a hiding position or setting up an ambush is usually done with the intention of waiting for an opportune moment to strike or observe the enemy'S movement: The current command needs to be kept by 'long term'.

Table 9: Chain of thought response from GPT4.

In an effort to mitigate the risk of overfitting our model to concise formatted outputs, thereby preserving its capacity for environmental reasoning, we augmented our dataset with a substantial volume of Chain of Thought data. This augmentation approach entails a systematic procedure whereby the large language model is guided through a step-by-step ideation

1.Only select the most relevant and necessary states for planning, and the unplanned states will be adjusted by the agent itself
2.[Choose 1, Choose 2, ...] indicates the values that can be selected for the state. When you plan, you can only choose the value of the state from it, and do not invent new value not listed in [Choice1, Choice2, ...].
3.The selected state can change the current value or maintain the current value. The agent will try to achieve and maintain the value of the state you choose after you give the plan.
4.Agents don't voluntarily discard items (for example guns, bullets, medical kits) unless items are reduced or set as False in your plan, so there is no need to keep them, only to choose when making changes.
5.Do not plan and adjust the states of teammates and enemies, they can move freely and cannot be controlled.
6.Avoid conflicts of states planing. For example, agent unable to move quickly when lying down, and unable to see enemies when length of distance from agent to enemy is far away.
7.Avoid the repetition of states planing. For example, if the Average velocity has been adjusted to be Fast, there is no need to adjust the Whether prone position to False, because the agent can automatically adjust state to fit overlapping meanings.
8.When it is necessary to refer to enemy or teammate information for planing, describe the specific state value during analysis.

Table 10: Rule prompt for GPT4.

process, ultimately culminating in the attainment of the intended target state. Concretely, our methodology commences with an initial inquiry into the semantic interpretation of the given instruction, followed by the identification of pertinent states, contemplation of state adjustments, analysis of action propensities, and an estimation of the requisite temporal considerations. Comprehensive documentation of the detailed prompts and ensuing responses derived from the Chain of Thought procedure can be found in Tables 16 and 9. It is noteworthy that traditional Chain of Thought processes in existing large language models often generate sequential thoughts, a method characterized by a relatively protracted temporal trajectory. This sequential reasoning approach may not be well-suited to the high real-time demands typically encountered in first-person shooter (FPS) games. Furthermore, the singular-step reasoning capabilities inherent in smaller language models are intrinsically modest and prone to errors. Consequently, the amplification of error probabilities within the Chain of Thought reasoning process may not yield superior outcomes. In light of these considerations, we have undertaken a strategy that amalgamates Chain of Thought data with the final target state data, thereby enhancing the fine-tuning of our language model. In the course of test reasoning exercises, the language model promptly generates the ultimate target state, with the Chain of Thought information being implicitly encoded within the neural network parameters.

Instruction Datasets. To cover a comprehensive range of instruction types and state distributions, we generated four types of instruction sets, which, when combined with states sampled from the environment, result in four different datasets. These are the HI (Human Instruction) dataset, constructed based on human-annotated commands; the SI (State Instruction) dataset, built by reverse-generating commands based on state transitions specified by the intelligent agent; the AI (Agent Instruction) dataset, constructed by main kinds of instruction which can be complete by pre-trained Agent; and the RI (Random Instruction) dataset, generated through random sampling of agent state transitions and random commands.

- *Human Instruction Dataset.* In this dataset, we generate open-ended instructions manually, while the corresponding states are sampled from the intelligent agent's interaction logs. These are combined and annotated using GPT-4 based on the prompting method previously described. We found that due to varying frequencies of state changes during the agent's interactions, some states are difficult to capture comprehensively only using random sampling. To ensure a more comprehensive distribution of states in the data and to facilitate better understanding by the language model, we employ a multi-round rejection sampling approach to construct state set. Let S be the set of states waiting to be sampled. We perform multiple rounds of sampling on S, with S_i^{get} representing the set of states sampled in the i-th round, initially empty. Next, we sample a state s from S without replacement and check whether s has any state values not present in S_i^{get}. If it does, we accept it and add it to S_i^{get}, otherwise we reject it. Once all states in S have been sampled, one round is completed. S_i^{get} is the result of i-th round's sampling, and S will be reset for the next round. This sampling method is employed to enhance the comprehensiveness of state coverage in all datasets except the Random Instruction dataset.
- *State Instruction Dataset.* In this dataset, we aim to cover a broader range of state changes in the instructions to enhance

the language model's understanding of various state transitions. To achieve this, we design corresponding goals and instructions for all states. Specifically, for each value of each state, we generate a series of instructions that require the corresponding state and value. These are then annotated using GPT-4 based on the prompting methods previously described. The annotated results are checked; if they do not have corresponding states and values, manual annotation and modification are performed to include the relevant states.

- *Agent Instruction.* In this dataset, we aim to initially align the planning capabilities of the language model with the pre-trained abilities of an intelligent agent based on reinforcement learning policies. To do so, we generate potential corresponding instructions based on actual state changes in agent interactions. Specifically, we first sample a series of agent state pairs at 5-second intervals. For a subset of these, we manually annotate possible corresponding instructions. We then use these manual annotations as a knowledge base and employ the "langchain" method to use these examples to guide the annotation of the remaining data using ChatGPT-3.5. Finally, we represent all the instructions as vectors using OpenAI's embedding API and perform clustering. We select the 14 most representative types of instructions and pair them cyclically with two rounds of sampled states, ultimately constructing a dataset that better reflects the fundamental execution capabilities of the intelligent agent.

- *Random Instruction.* This dataset is primarily designed to enrich the data distribution. It is constructed by randomly generating instructions and fully randomly sampling states, and then annotated using GPT-4 based on the prompting methods previously described.

The quantaty of the aforementioned four types of datasets is 507 for HI, 1098 for SI, 1441 for AI and 1382 for RI. Moreover, the test dataset construct instructions that differ from those used in the training data, then utilize GPT-4 to generate draft labels of goals and modified with manually filtered and annotated. This test dataset used for evaluating the model's ability to plan reasonably in response to instructions. And the size of dataset for each tuning step is 26,568 for CoT-assited fine-tuning, 4,428 for supervised fine-tuning, and 4,994 for ensembling fine-tuning.

F. Distributed Training Framework

To improve the training efficiency, we adopt a distributed training system, shown in Figure 8. In this system, the *Actors* run over CPU nodes to collect training data, then send the collected data to the *Learner* which is deployed on a GPU node.

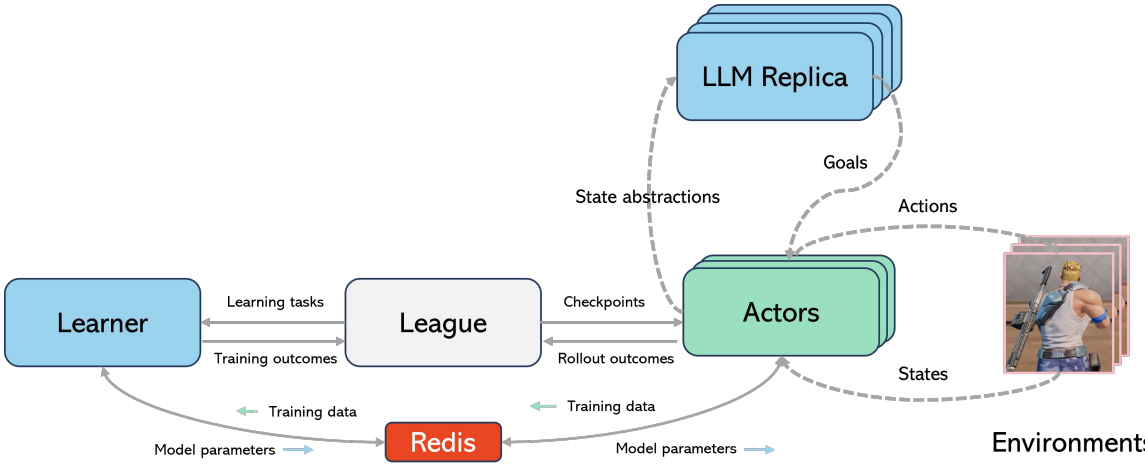


Figure 8: This training system has three main parts: Actor, Learner and League. Actor plays the role of collecting data, Learner trains the policy model with these data and League coordinates the entire training process and displays training results.

We further take a LLM server to enable multiple replicas of LLM for goal generation, which improve the throughput of rollout when the RL training is switch to goal-conditioned cases.

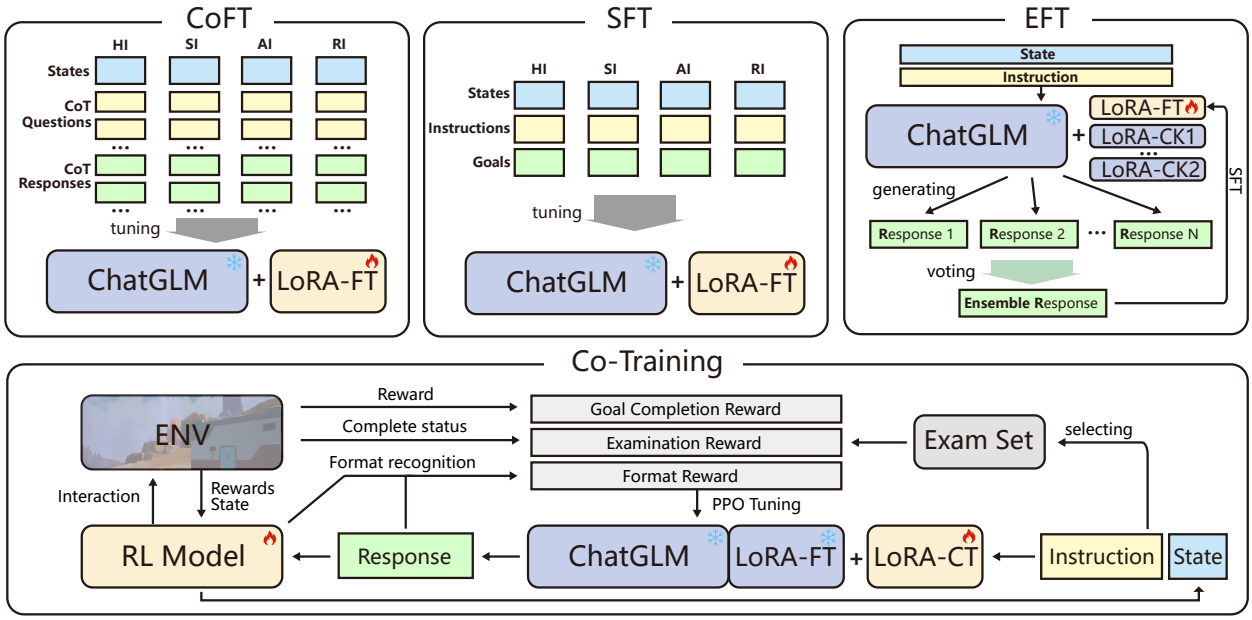
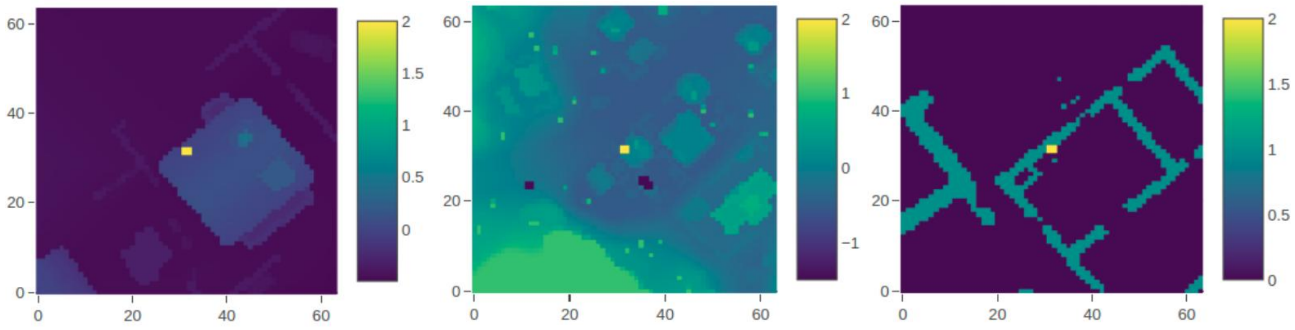


Figure 9: Overview of the training framework with LLM. This training framework has three kinds of LLM tuning approaches: CoFT (Chain of Thoughts assisted Fine-Tuning), SFT (Supervised Fine-Tuning), EFT (Ensemble Fine-Tuning); and one LLM-RL co-training approach.

G. Parameter Settings

The hyperparameters used in our experiment are illustrated in Table 11. Some of them are set by following the official implementation of PPO (Schulman et al., 2017). Due to the limitation of compute resource, we did not tune these hyperparameters. Other dynamic hyperparameters are introduced their corresponding parts.



(a) Altitude map (0.8m) (b) Altitude map (4m) (c) Aerial view map (0.4m)

Figure 10: Illustration of heatmap features in observation space. (a) and (b) are the altitude maps where bright areas are higher than dark areas. (c) is the aerial view map where the disconnected areas are windows or doors. One pixel in (a), (b) and (c) denotes 0.8 meter, 4 meters and 0.4 meter respectively. The small yellow blocks represent player positions and small blue blocks represent enemy positions.

H. Algorithms

Algorithm 2 lists the pseudo-code of goal-conditioned RL procedures in Stage I, where G_t^{basic}, G_t^{oa} and G_t^{g} represent the basic return, obstacle avoidance return and goal-reaching return from time step t till the termination, respectively.

Algorithm 2 Curriculum Goal-conditioned RL

Input: θ parameterizes policy π and $\zeta = \{\zeta_1, \zeta_2, \zeta_3\}$ parameterizes value heads, goal generators G_{RAND} and G_{HER}
for k=1, 2, ... **do**
  Reset environment with returned initial state s_0
  if goal mode **then**
    Sample a goal: $g \sim G_{RAND}(\cdot) \cup G_{HER}(s_0)$
  else if not goal mode **then**
    Initialize an empty goal: $g \leftarrow \emptyset$
  end if
  Run policy π_{θ_k} in environment until be terminated
  Actors collect trajectories $\mathcal{D}_\tau$ and send them to the Learner
  Update the θ_k to θ_{k+1} with Equation (4)
  Update ζ by $\max_\zeta \mathbb{E}_{s\sim\mathcal{D}_\tau}\left[\|G_t^{basic} - V_{\zeta_1}^{basic}(s_t)\|_2 + \|G_t^{oa} - V_{\zeta_2}^{oa}(s_t)\|_2 + \mathbb{1}_{g_t\neq\emptyset}\|G_t^{g} - V_{\zeta_3}^{g}(s_t)\|_2\right]$
end for

I. Hindsight Goal-generation

Inspired by Hindsight Experience Replay (HER) (Andrychowicz et al., 2017), we adopt a similar method to utilize the collected trajectories for learning a goal generator G_{HER} which accepts a state as input. We conclude its training in two steps: (1) constructing (s, g) pairs with collected trajectories as illustrated in Figure 11(b); (2) supervised training G_{HER} with the above pairs and an MSE loss between the labeled goals and predicted goals. For step (1), we split trajectories into many segments with length of 200 timesteps. Then, we randomly sample a state s from the first 150 steps and sample a state s' from the last 20 steps to derive a goal $g = Proj(s')$, with a distribution proportional to their basic value $V(s')$. For step (2), we train G_{HER} with s, Δt, $V^{basic}(s)$ and $V^g(s)$ as input to generate goals, where Δt the time slot of goal completion, $V^g(s)$ the goal-achieving value, $V^{basic}(s)$ the basic state value.

PPO clip eps	0.2
Optimizer	Adam
Learning rate	0.0001
Batch size	20480
Number of CPUs	5120 (AMD EPYC 7H12 64-Core)
Number of GPUs	2 (A100)
γ $(basic)$	0.995
γ (oa)	0.92
γ $(goal)$	0.993
λ	0.95
Entropy coefficient	0.025
Unroll length	20
Sample max use times	3
Gradient clip threshold	10
Action space size	56
Meta states size	2^{68}

Table 11: Parameter settings for RL.

J. Additional Experimental Results

J.1. Lora Method Experiment

Lora Rank. We evaluate the impact of the rank parameter on performance during Lora fine-tuning of large language model neural networks. Generally speaking, the larger the rank parameter, the more comprehensive and thorough the fine-tuning of the neural network, but the corresponding training time and model footprint will be larger. The experimental results are shown in Table 12. The size of lora rank has little impact on model performance indicators, but a large rank will cause the model training time and the size of the saved parameter file to increase dramatically.

Rank	Precision	Precision (Choice)	Recall	Recall (Choice)	F1	F1 (Choice)	Accurate	Accurate (Choice)
8	0.544	0.672	0.482	0.608	0.502	0.629	0.060	0.124
16	0.550	0.673	0.487	0.601	0.507	0.626	0.070	0.124
32	**0.555**	**0.685**	0.505	0.621	**0.529**	**0.652**	0.065	**0.159**
64	0.547	0.675	0.501	0.616	0.519	0.635	0.070	0.124
128	0.552	0.684	**0.507**	**0.626**	0.524	0.645	**0.075**	0.134

Table 12: Evaluation on lora rank.

Lora Target. We next verified which neural networks in fine-tuning the ChatGLM-6B large language model can achieve the best performance. The experimental results are shown in Table 13. It is worth noting that only fine-tuning the MLP network without fine-tuning the attention network can achieve the best training results. Although generally speaking, the mainstream fine-tuning task of large language models is to fine-tune the attention layer network, but that task usually focuses more on answer semantics. In our task, we pay more attention to the format to meet the metastate parsing requirements, so fine-tuning the MLP network can achieve better results.

Dataset	Precision	Precision (Choice)	Recall	Recall (Choice)	F1	F1 (Choice)	Accurate	Accurate (Choice)
Attention	**0.555**	**0.685**	**0.505**	**0.621**	**0.529**	**0.652**	0.065	**0.159**
Mlp	0.549	0.664	0.482	0.587	0.514	0.620	0.065	0.134
All	0.529	0.642	0.471	0.581	0.485	0.596	**0.069**	0.119

Table 13: Evaluation on lora target.

Index	Meaning
g^1	Average velocity
g^2	Horizontal direction of movement
g^3	Whether seen enemy
g^4	Whether hold a gun
g^5	Whether prone position
g^6	Length of distance moved
g^7	Length of distance from agent to teammate
g^8	Distance with nearest enemy
g^9	Whether seen by enemy
g^{10}	Damage to enemy
g^{11}	Whether have bullets
g^{12}	Horizontal direction of view
g^{13}	Whether follow with the movement direction of teammate
g^{14}	Whether crouch position
g^{15}	Whether have a gun
g^{16}	Whether have medical kits
g^{17}	Whether to restore health
g^{18}	Health level
g^{19}	Whether knock down enemy
g^{20}	Whether target the same enemy as teammate

Table 14: Top 20 sub-goals ranked by frequency.

J.2. Experiment of the goal-generation dataset size.

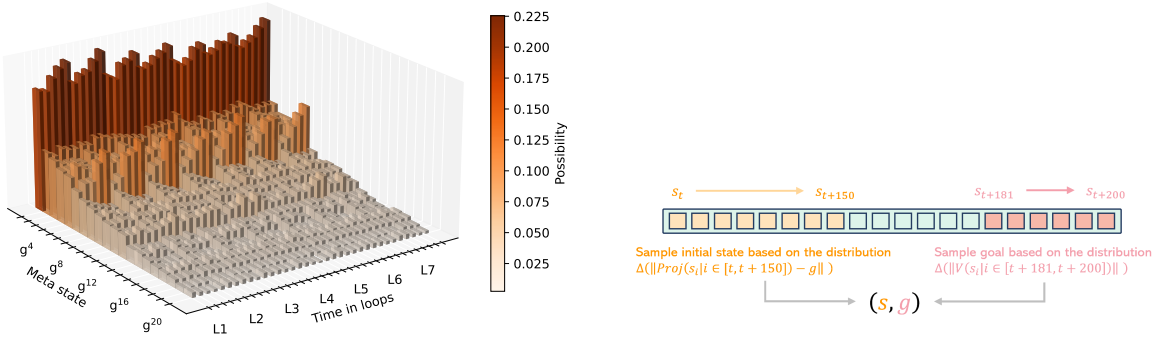
We conduct experiments of various models with four percentages of fine-tuning train set, i.e., 100%, 30%, 10%, 3%, on the goal generation task. The results are shown in Table 15. It can be seen that as the amount of data gradually decreases, the performance of various training indicators gradually deteriorates under various settings. However, the smaller the amount of data, the greater the improvement brought by pre-training of our proposed CoTF method. The results show that the CoTF method we proposed can effectively collect and expand the chain of thought data related to the final goal, thereby avoiding overfitting of the training set in the case of small data.

K. Reward Functions for Co-Training

Agent Feedback Rewards. The calculation of the agent feedback reward is multifaceted, aiming to reflect the degree of completion as feedback for the training of the LLM. Specifically, three aspects are considered to satisfy the requirements, and the total agent feedback reward is given by the sum of them:

- r_g^f **- Minimal Distance to a Goal When Satisfying Environment Termination**. As depicted by Equation (7), the agent progressively reduces the distance between the initial state and the goal, scaling it by the magnitude of the initial state-goal difference:

$$R_g^f = \sum_{t=1}^{T} \| \frac{|g - Proj(s_{t-1})| - |g - Proj(s_t)|}{|g - Proj(s_0)| + \epsilon} \|_1, \text{ where } \epsilon = \text{1e-6} \tag{7}$$



(a) (b)

Figure 11: (a) Sub-goal distribution during co-training. The 20 most frequently occurring goal meta states are filtered out and displayed. The vertical axis represents the probability of the state being output by the language model; (b) For a collected trajectory segment with length $k = 200$, we firstly estimate the basic value for the last $k - j + 1$ states (here $j = 20$) and select one state as the goal with the probability proportional to their values.

- r^f_{keep} **- Reward Indicating How Long the Goal Can Be Kept**. As depicted by Equation (8), upon accomplishing the goal, the agent receives a reward proportional to the cumulative number of steps taken to sustain the goal state, scaled by the count of distinct sub-goals between the initial state s_0 and the goal g, i.e. $n(g \cap Proj(s_0))$:

$$R^f_{keep} = n(g \cap Proj(s_0)) \cdot \sum_{t=0}^{T} \mathbb{1}_{g \cap Proj(s_t) \neq \emptyset} \tag{8}$$

- r^f_{rnd} **- Reward Indicating Whether the Generated Goal Satisfies the Projection of the Current State**. As shown in Figure 12, an RND network is deployed for the evaluation of a state-goal pair. The calculation of r^f_{rnd} indicates that frequently appearing state-goal pairs are feasible, while those that never appear tend to be infeasible in the environment:

$$R^f_{rnd} = -\sum_{t=0}^{T} \|\varphi(E(s_t, g)) - \varphi^\star(E(s_t, g))\|, \tag{9}$$

  where $\varphi^\star$ a target network which shares the same architecture as the RND predictor but the network is non-trainable.

Examination Reward Function. The examination reward function is introduced as an intrinsic signal to encourage the LLM to generate goals with essential sub-goals. We use the SI dataset as the examination set $\mathcal{I}_S$. For each training iteration, a batch of instructions $\mathcal{I}_{train}$ is randomly sampled from the full instruction dataset $\mathcal{I}$, and corresponding goals g are generated. After the agent finishes its rollout, the examination reward for each batch is computed based on the intersection $\mathcal{I}_\cap = \mathcal{I}_S \cap \mathcal{I}_{train}$. For non-empty $\mathcal{I}_\cap$, an examination reward for each instruction in $\mathcal{I}_\cap$ is computed as:

$$r^e(i, g, g_{sub}) = \begin{cases} +2 & g_{sub} \in g \\ -2 & \text{otherwise} \end{cases}, \forall i \in \mathcal{I}_\cap \tag{10}$$

Then, R^e is calculated as $R^e = \frac{1}{|\mathcal{I}_\cap|} \sum_{i \in \mathcal{I}_\cap} r^e(i, g, g_{sub} | g = LLM(i, s))$.

Formatting Reward Function. The formatting reward for each generated goal is calculated by computing an edit distance, utilizing the Wagner-Fischer algorithm (Wagner & Fischer, 1974).

With the defined reward functions, RLAF is applied with a reward function $R = R^f + R^e + R^m$ and Proximal Policy Optimization (PPO) for each data point in a batch.

Dataset Size	Training Method	Precision	Precision (Choice)	Recall	Recall (Choice)	F1	F1 (Choice)
100%	CoTF	51.85%	64.84%	46.68%	57.91%	49.13%	61.18%
	CoTF ->SFT	55.48%	68.52%	50.48%	62.10%	52.86%	65.15%
	SFT	54.70%	65.20%	49.00%	60.20%	51.70%	63.20%
	Improve Rate	1.42%	5.09%	3.02%	3.16%	2.25%	3.09%
30%	CoTF	49.43%	62.66%	44.45%	56.29%	46.81%	59.30%
	CoTF ->SFT	49.92%	62.59%	45.51%	57.65%	47.61%	60.02%
	SFT	46.12%	60.96%	33.68%	45.39%	38.93%	52.03%
	Improve Rate	8.25%	2.68%	35.11%	27.01%	22.30%	15.34%
10%	CoTF	45.58%	60.84%	41.77%	53.92%	43.59%	57.17%
	CoTF ->SFT	48.06%	61.01%	43.15%	54.31%	45.47%	57.47%
	SFT	42.08%	55.31%	30.86%	41.45%	35.61%	47.39%
	Improve Rate	14.20%	10.32%	39.80%	31.03%	27.69%	21.28%
3%	CoTF	39.42%	58.35%	34.28%	50.10%	36.67%	53.91%
	CoTF ->SFT	41.61%	60.40%	34.55%	50.32%	37.75%	54.90%
	SFT	17.66%	38.28%	13.33%	29.47%	15.20%	33.30%
	Improve Rate	135.52%	57.78%	159.15%	70.79%	148.43%	64.88%

Table 15: Language model performance evaluation with different sizes of fine-tuning training set. The underlined "Improve Rate" values represent the improvement percentage of the "CoTF ->SFT" method relative to "SFT" method.

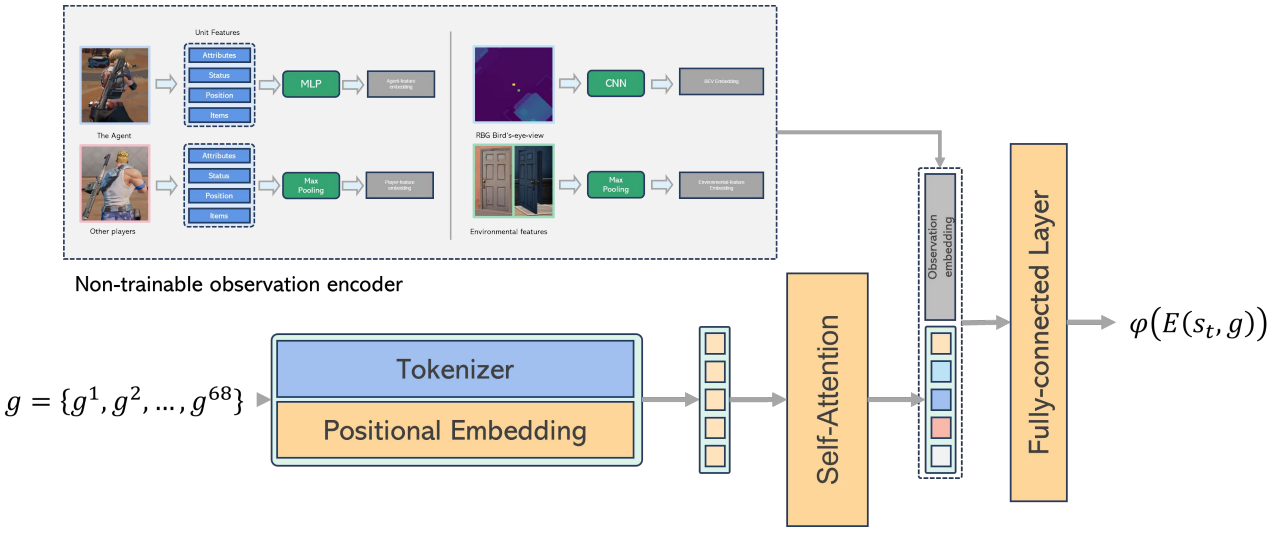


Figure 12: Implementation of the RND predictor network.

In order to complete the command 'You should lie in wait', let us plan the states of the agent step by step using the following template:
1. Analyze the verbal orders of teammates and players, what do you want to do? According to the command, also analysis the relevant states of teammates and enemies that need attention. The verbal command of the teammate player is [Command], which means teammate player wants the agent...
2. Analyze which states of the agents are most relevant to the verbal commands of teammate player. The agents in the unselected states will adjust themselves to complete your plan (analyze the reason first, then select key states one by one as few as possible and as important as possible according to the degree of importance)? According to the teammate's command: 2.1. [Reason1]: [State1] 2.2. [Reason2]: [State2] ...
3. Plan how these key states need to be adjusted (analyze the reason first, and then make adjustments one state by one state, the state can be changed or remain the same, and must be selected from the value range of the game state [Choice 1, Choice 2, ...])? According to the teammate's command: 3.1. [State1]: [Reason1]: [Current_value1] ->[Target_value2] 3.2. [State2]: [Reason2]: [Current_value1] ->[Target_value2] ...
4. Modify the adjustment that may be wrong, and refer to the Rules to analyze which state adjustments may conflict, repeat or be unnecessary, and output the modified adjustment plan: According to the states adjustments in 3... 4.1. [State1]: [Current_value1] ->[Target_value2] 4.2. [State2]: [Current_value1] ->[Target_value2] ...
5. According to the analyze and the planing of the verbal command, further analyze the behavior tendency required in the adjustment process (the proportion of Mobile, Offense, Waiting, Supplies, Scouting, first analyze the reason, and then calculate the percentage) Mobile: [Reason1]: [Percent1] Offense: [Reason2]: [Percent2] Waiting: [Reason3]: [Percent3] Supplies: [Reason4]: [Percent4] Scouting: [Reason5]: [Percent5]
6. Analyze how long the current command needs to be kept (for example, the command of 'killing the enemy' needs to be kept for a 'short term', and the command of 'pay attention to reconnaissance' needs to be kept for a 'long term'. First analyze the reason and then make a judgment). According to the command of the teammate, [Analysis]: The current command needs to be kept by '[XX term]'. If you see phrases like [Context] in answer template, replace the entire phrase according to the meaning of the Context, do not repeat the content; make analogy expansion for '...'; keep ':'; absolutely do not modify others in template.

Table 16: Chain of thought prompt for GPT4.

prompt	system background prompt	We have an agent and a player working together as a teammate in a PUBG game. We hope you can help the agent plan how the agent's game state should change, so as to complete the player's command and help the player win the game.
	teammate state prompt	The state of the agent's teammates can be described as follows:{' Length of distance moved': 'No movement', 'Average velocity': 'Slow', 'Horizontal direction of movement': 'Southeast', 'Horizontal direction of view': 'South', 'Pitch direction of view': 'Medium', 'Health level': 'Empty', 'Whether to restore health': 'False', 'Whether the health is damaged': 'False', 'Whether rescued teammate': 'False', 'Whether prone position': 'False', 'Whether crouch position': 'False', 'Whether have a gun': 'True', 'Whether hold a gun': 'False', 'Whether have bullets': 'True', 'Whether have medical kits': 'True', 'Whether be knocked down': 'False', 'Damage to enemy': 'Zero', 'Whether knock down enemy': 'False', 'Whether seen enemy': 'True', 'Number of enemies have ever seen': 5, 'Whether seen by enemy': 'True', 'Distance with nearest enemy': 'Nearby', 'Whether closer with nearest enemy': 'False', 'ID of teammate player': 2}
	enemy state prompt	The state of the enemy can be described as follows:{'Horizontal direction of movement of enemy': 'Southwest', 'Velocity of enemy': 'Slow', 'Enemy's position relative to agent': 'West'}
	self state prompt	The state of the agent can be described as follows:{'Damage to enemy': 'Zero', 'Whether knock down enemy': 'False', 'Whether kill enemy': 'False', 'Whether seen enemy': 'True', 'Whether seen by enemy': 'True', 'Number of enemies have ever seen': 3, 'Length of distance moved': 'Short', 'Average velocity': 'Fast', 'Horizontal direction of movement': 'West', 'Horizontal direction of view': 'NorthEast', 'Pitch direction of view': 'Medium', 'Health level': 'Full', 'Whether to restore health': 'False', 'Whether the health is damaged': 'False', 'Whether rescued teammate': 'False', 'Whether be knocked down': 'False', 'Whether prone position': 'False', 'Whether have a gun': 'True', 'Whether have bullets': 'True', 'Whether have medical kits': 'True', 'Distance with nearest enemy': 'Nearby', 'Whether closer with nearest enemy': 'True', 'Whether crouch position': 'False', 'Whether hold a gun': 'True', 'Whether seen by teammate': 'True', 'Length of distance from agent to teammate': 'Touch', 'Teammate's position relative to agent': 'Southwest', 'Whether follow with the views of teammate': 'False', 'Whether target the same enemy as teammate': 'False', 'Whether follow with the movement direction of teammate': 'False'}
	command prompt	Teammate player gives you a verbal command based on the current game states:'You should lie in wait.'
	question prompt	In order to complete the command 'You should lie in wait.', how the agent's game state should change?
response	goal meta-state prompt	Whether prone position:True Average velocity:Static Length of distance moved:No movement Whether hold a gun:True

Table 17: **Example of prompt and response.**