Grasper: A Generalist Pursuer for Pursuit-Evasion Problems (Appendix)

Anonymous Author(s)

ACM Reference Format:
Anonymous Author(s). 2024. Grasper: A Generalist Pursuer for Pursuit-Evasion Problems (Appendix). In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 8 pages.

A MORE DISCUSSION

In this section, we provide more discussion and explanations to further facilitate the understanding of the contributions of this work by answering several questions.

Q1. Significance of PEGs.

(i) As a classical type of game, PEGs have a long history in the field of game theory and provide a general framework that mathematically formalizes many important real-world scenarios such as surveillance, navigation, analysis of biological behaviors, and conflict and combat operations [4, 10, 12, 18, 19, 24, 25]. Although there exist different terminologies in different research such as Princess *vs.* Monster [13], Hunter *vs.* Rabbit [1], Cops *vs.* Robbers [3], Predator *vs.* Prey [6, 23], and Defender *vs.* Attacker [14, 20], they share a similar game model which sets up two players or two teams of players against each other. Moreover, numerous works have been done on solving PEGs under various assumptions, e.g., mobile players, intelligent adversaries, and multiple pursuers and evaders.

(ii) In particular, PEGs on graphs have been used to model the interactions between the pursuer and evader in graph-based environments [5, 7, 14, 16, 17, 26, 27, 29, 30]. As the hardness of solving PEGs on graphs is closely related to the size of the underlying graph, it is important and necessary to develop scalable approaches to solve large-scale PEGs with city-scale road networks.

Q2. The necessity of a new approach when considering solving different PEGs with varying initial conditions.

(i) As mentioned in the main text, given a PEG on a grid map with size 10×10, existing algorithms typically need one to two hours to compute the NE strategy, and the running time increases with the size of the map. Once the initial conditions of the PEG change, we need to re-run the algorithms to compute the NE strategy from scratch, which would be a significant computational challenge.

(ii) In real-world scenarios, the initial conditions of PEGs are not always fixed: a crime can occur at any location of a city at any time, exits can change due to temporary closures and openings, and when the pursuers will catch the evader is not always the same under different situations. Thus, the number of PEGs on a given graph can be enormous. For example, the Singapore map consists of 372 nodes. Suppose there are 5 pursuers and 8 exits, and the time horizon is $6 \leq T \leq 10$. Then, the number of PEGs is $5 \times 372 \times C_5^{371} \times C_8^{371} \approx 8.72 \times 10^{29}$. The number of PEGs is still extremely large even after removing some trivial cases such as some nodes are not the candidate exits in real-world scenarios. Therefore, it is unbearable to solve each possible PEG from scratch with existing algorithms (including classical heuristic methods such as Dijkstra, linear program, CFR, and PSRO).

(iii) The problem of solving different PEGs with varying initial conditions involves the generalizability of an algorithm over different PEGs, which has not been addressed in previous works. In view of this fact and the infeasibility of existing algorithms, there is an urgent necessity to propose a new approach to solve different PEGs with varying initial conditions efficiently.

Q3. More explanations on the game model.

We provide more explanations on the reasonability of the game model in characterizing real-world scenarios.

(i) The pursuers can have the real-time location information of the evader with the help of tracking devices such as GPS-based systems [8] and cameras [21] extensively deployed in the city. For example, the StarChase [8] can provide police officers with real-time (every 3 to 5 seconds) GPS locations of the fleeing vehicle. Therefore, the assumption of the availability of the real-time location of the evader is reasonable and widely adopted in previous works [16, 30].

(ii) In a pursuer's observation, the location information only contains the current locations of the players not the historical locations of the players. Such a setting reduces the size of the observation representation and still carries the entire information as the locations of all players in the next time step depend only on their current locations and their actions (as in [16]).

(iii) A pursuer member can have the location information of the other members (full information), i.e., the observation of a pursuer member consists of the locations of all other pursuer members. This condition does not limit the applicability of the game model because, in real-world scenarios, the pursuer members can easily get the locations of each other through communication (e.g., the walkie-talkies or security command center).

Q4. Did the key ideas have been developed by previous works and the only improvement is attributed to the "generalist" training such that this work seems incremental?

The answer is negative. To our knowledge, this is the first attempt to consider the generalization problem in the domain of PEGs. Previous works are typically tailored to a specific PEG with the given initial conditions and are required to compute the NE strategy from scratch when the conditions change, as mentioned in Section 3.3. Our work is closely related to [16], but the core components proposed in Section 4 are entirely novel, as shown in Figure 1 in the main text. To be clearer, we provide more explanations from two aspects: architecture and training pipeline.

Architecture:

(i) Although the method developed in [16] is the SOTA method for solving *a specific PEG*, it requires re-training the node embedding model and the base policy from scratch once the initial conditions

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org). This work is licenced under the Creative Commons Attribution 4.0 International (CC-BY 4.0) licence.

of the PEG change, which possesses a significant time and computational challenge to solve *different PEGs with varying initial conditions*. With our proposed components, we can easily construct a *SOTA baseline*, MT-PSRO, with minimal and most straightforward modifications to the method in [16].

(ii) To efficiently solve different PEGs, the key is to efficiently leverage the information about the initial conditions, which has not been explored in previous works (note that MT-PSRO does not explicitly take this information into account). To this end, we need to address several closely coupled questions. First, what is the most appropriate approach to represent the initial conditions of PEGs? As the PEG is played on a graph, we propose a graphical representation that is universal for any PEG with any initial condition. Then, naturally, we propose to leverage a GNN to encode these graphs corresponding to different PEGs into low-dimensional hidden vectors. After that, the next question is how to effectively integrate these hidden vectors into policy learning. The most straightforward idea is to directly concatenate these hidden vectors with the output of the observation representation layer to obtain the (augmented) inputs of the policy network, i.e., the baseline MT-PSRO-Aug. However, from our experiments, we can see that such a naive manner is inefficient. To more effectively leverage the initial conditions, we propose (SOTA) Grasper which generates the base policy network conditional on the initial conditions by using a hypernetwork.

In summary, instead of a straightforward extension to previous works, we address several critical challenges, leading to the SOTA Grasper (MT-PSRO $\longrightarrow$ MT-PSRO-Aug $\longrightarrow$ Grasper).

Training Pipeline:

(i) There are several components in our architecture: GNN, observation representation layer, and hypernetwork. The most straightforward manner is to train all these parts together in an end-to-end way. However, as shown in Figure 5 and Table 2 in the main text, this is inefficient. In fact, GNN is only used to extract the features of the initial conditions of PEGs. Therefore, we propose to pre-train the GNN (pre-pretraining stage) before pre-training other parts, which is more efficient than jointly training all the components.

(ii) For the pre-training stage, though the basic principle is similar with [16], there are two main differences. First, the task spaces are different. In our pre-training (Algorithm 2 in the main text), the number of tasks is c_1c_2 while it is c_2 in [16]. Second, in our pre-training, we use heuristic-guided multi-task pre-training (HMP) which is a novel pre-training scheme proposed in our work, while [16] uses the traditional multi-task pre-training. Thus, the training data (the reference policy $\hat{\pi}^p$ is used in our HMP) and the optimization loss are different. To be clearer, we present the pre-training process of [16] in Algorithm 4, and the differences between our pre-training and Algorithm 4 are highlighted in blue in Algorithm 2.

(iii) For the fine-tuning stage, the main difference between our work and [16] lies in that we can quickly obtain the base pursuer's policy π_0^p through the hypernetwork for any given PEG. However, in [16], one must first perform Algorithm 4 to get the base policy π_0^p when solving a new PEG (i.e., learning from scratch). For comparison, we present the fine-tuning process of [16] in Algorithm 5.

In summary, there are three most critical aspects of the training pipeline: the introduction of the pre-pretraining stage which significantly improves the training efficiency, the HMP which effectively addresses the exploration inefficiency, and the game-conditional base policies generation in the fine-tuning stage.

Algorithm 4: Pre-training in [16]

```
1  Given $\mathcal{G}$, initialize pursuer's policy $\pi_\theta^p$ and buffer $\mathcal{D} \leftarrow \emptyset$;
2  for train epoch $= 1, 2, \cdots$ do
3      Randomly generate $c_2$ evader's policies;
4      for each of the $c_2$ evader's policies $\pi^e$ do
5          Sample data using $\pi^e$ and $\pi_\theta^p$;
6          Add the data into the episode buffer $\mathcal{D}$;
7      end
8      Train the networks by optimizing loss $L(\theta)$;
9      Clear the episode buffer $\mathcal{D} \leftarrow \emptyset$;
10 end
```

Algorithm 5: Fine-tuning in [16]

```
1  Require: Pre-trained $\pi_0^p$ for a given PEG $\mathcal{G}$;
2  $\Pi_0^p = \{\pi_0^p\}$, $\Pi_0^e = \{\pi_0^e\}$, $U_0$, $\sigma_0^p$, $\sigma_0^e$;
3  for epoch $k = 1, 2, \cdots K$ do
4      Compute the evader's BR policy $\pi_k^e$ against $\sigma_{k-1}^p$;
5      Initialize the pursuer's BR policy $\pi_k^p \leftarrow \pi_0^p$;
6      Train $\pi_k^p$ against $\sigma_{k-1}^e$ for few episodes;
7      Expansion: $\Pi_k^p = \Pi_{k-1}^p \cup \{\pi_k^p\}$, $\Pi_k^e = \Pi_{k-1}^e \cup \{\pi_k^e\}$;
8      Update meta-game matrix $U_k$ through simulation;
9      Compute $\sigma_k^p$ and $\sigma_k^e$ using a meta-solver on $U_k$;
10 end
11 Return: $\Pi_K^p$, $\Pi_K^e$, $\sigma_K^p$, $\sigma_K^e$
```

Q5. Why do initial conditions include the initial locations?

As the pursuer's observations include the locations of the pursuers and evader, one may come to the conclusion that the initial locations of the pursuers and the evader are not really distinguishers in describing different PEGs. More intuitively speaking, one may expect that the pursuer's policy would learn distinguishable representations so that it can perform well in PEGs with different initial locations of the players if it is trained by using the *data sampled from different PEGs with different initial locations of the players*. However, we note that this is exactly the training procedure of the MT-PSRO which, from our experiments, performs unsatisfying without explicitly taking these initial locations into account.

Q6. More explanations on the evader's policy.

(i) Note that the evader's policy is a *stochastic policy (not a pure policy)* which is defined as $\pi^e : V \rightarrow \Delta(V')$ where $\Delta(V')$ is the probability distribution over all the exit nodes.

(ii) This simplification of the evader's policy definition is called High-Level Actions in previous works, which is a necessity in the PSRO algorithm/fine-tuning procedure (Algorithms 1, 3, and 5) in this work as we only need to estimate the values of the exit nodes through simulations when computing the evader's BR policy. Otherwise, the problem will be more intricate, and more advanced techniques are required, if the evader is also learning-based.

B IMPLEMENTATION DETAILS

In this section, we present details on the environment, hyperparameters, network structure, loss function, and baselines.

B.1 Environment

The environment is implemented following the description of the game model in Section 3. The underlying graphs are shown in Figure 7. The scale-free map is generated using Barabási-Albert preferential attachment [2], which can be easily obtained by leveraging the package "NetworkX" [9]. The Singapore map is from [27]. The Scotland-Yard map is from the board game "Scotland-Yard"[1].

B.2 Hyperparameters

All experiments are performed on a machine with a 24-core Intel(R) Core(TM) i9-12900K and NVIDIA RTX A4000. The hyperparameters related to pre-pretraining, pre-training, and fine-tuning are listed in Table 3. Note that the HMP coefficient α is linearly decayed for 40000 episodes. The hyperparameters related to the network structure are given in the next subsection. We also refer readers to [28] for more details since we use MAPPO as the underlying RL algorithm of both pre-training and fine-tuning processes[2].

B.3 Network Structure

Graph Neural Network (GNN). We use the well-known Graph Convolutional Network (GCN) as the encoder to encode the graphical representation of a PEG into the hidden vector. The implementation follows GraphMAE[3] [11]. For the specific architecture of the GCN, the number of GCN layers is 2, the hidden size is 128, the output size of each node embedding is 32, the dropout rate is 0.5, and the batch size for training the GCN is 32.

Hypernetwork and Policy Network. The policy network contains three fully-connected (FC) layers denoted as $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3)$, each with 128 neurons. The hypernetwork consists of two FC layers each with 128 neurons, followed by three heads each including two FC layers to respectively generate the weights and biases of the corresponding layer of the policy network. That is, the first head generates $\boldsymbol{\theta}_1 = (\boldsymbol{w}_1, \boldsymbol{b}_1)$ and similarly for the second and third heads. The structure of the hypernetwork and policy network is shown in Figure 8. The illustration is inspired by [15]. Different from [15], we choose not to generate the parameter "gain" for implementation convenience as "torch.nn.Linear" only contains weights and biases, which can be easily initialized by copying the generated $\boldsymbol{\theta}$.

Observation Representation Layer. An observation of a pursuer member consists of three parts: the locations of players, the pursuer member's id, and the current time step. We use three components to encode each part of the observation. Each component is a "torch.nn.Embedding" which is extensively used to encode an integer to a compact representation. The sizes of the dictionary for the three components are respectively the number of nodes of graph $|V|$, the number of pursuers n, and the maximum time horizon $\bar{T}$. In our experiments, we have $|V| = 100$ for the grid map, $|V| = 372$ for the Singapore map, $|V| = 300$ for the scale-free map, $|V| = 200$ for the Scotland-Yard map, $n = 5$, and $\bar{T} = 10$. The output

[1]https://boardgamegeek.com/boardgame/438/scotland-yard
[2]https://github.com/zoeyuchao/mappo.git
[3]https://github.com/THUDM/GraphMAE

Table 3: Hyperparameters

Name	Value
Pre-pretraining	
learning rate	$1.5e^{-4}$
weight decay	$1e^{-5}$
training epoch	2000
Pre-training	
learning rate of actor	$3e^{-4}$
learning rate of critic	$5e^{-4}$
MAPPO probability ratio clipping	0.2
MAPPO entropy coefficient η	0.01
MAPPO update epoch	15
discount factor γ	0.99
batch size	256
HMP coefficient α	$0.1 \rightarrow 0.01$
number of games c_1	5
number of evader strategies c_2	5
number of episodes for an RL task	10
Fine-tuning	
learning rate of actor	$1e^{-4}$
learning rate of critic	$5e^{-4}$
MAPPO probability ratio clipping	0.2
MAPPO entropy coefficient η	0.01
MAPPO update epoch	10
discount factor γ	0.99
batch size	32
evader's BR training episode	200
meta-game simulating episode	200

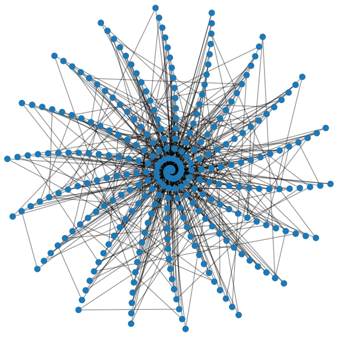
dimension of each component is 16. The outputs of the three components are concatenated to obtain the representation of the pursuer's observation, as shown in Figure 9. Therefore, the total dimension of the pursuer's observation representation is $(n + 3) \times 16$.

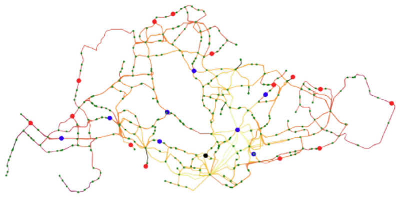
B.4 Loss Function

Let $\pi^p_{\boldsymbol{\theta}}$ and $V^p_{\boldsymbol{\phi}}$ respectively denote the actor and critic in the MAPPO algorithm. Then, the loss function for HMP is:

$$\begin{aligned} & L_t(\boldsymbol{\beta}^{\mathrm{A}}, \boldsymbol{\beta}^{\mathrm{C}}) \\ & = \mathbb{E}\Big[L^1_t(\boldsymbol{\theta} = f_{\boldsymbol{\beta}^{\mathrm{A}}}(\boldsymbol{h}_{\mathcal{G}}, T)) - L^2_t(\boldsymbol{\phi} = f_{\boldsymbol{\beta}^{\mathrm{C}}}(\boldsymbol{h}_{\mathcal{G}}, T)) \\ & + \eta\mathcal{H}(\pi^p_{\boldsymbol{\theta}=f_{\boldsymbol{\beta}^{\mathrm{A}}}(\boldsymbol{h}_{\mathcal{G}},T)}) + \alpha\mathrm{KL}(\pi^p_{\boldsymbol{\theta}=f_{\boldsymbol{\beta}^{\mathrm{A}}}(\boldsymbol{h}_{\mathcal{G}},T)}\|\hat{\pi}^p)\Big], \end{aligned} \quad (1)$$

where $f_{\boldsymbol{\beta}^{\mathrm{A}}}$ and $f_{\boldsymbol{\beta}^{\mathrm{C}}}$ are respectively the hypernetworks for generating the actor and critic networks and parameterized by $\boldsymbol{\beta}^{\mathrm{A}}$ and $\boldsymbol{\beta}^{\mathrm{C}}$ (similarly, $\boldsymbol{\beta}^{\mathrm{A,old}}$ and $\boldsymbol{\beta}^{\mathrm{C,old}}$ respectively denote the old version of the hypernetworks which are periodically updated by copying from $\boldsymbol{\beta}^{\mathrm{A}}$ and $\boldsymbol{\beta}^{\mathrm{C}}$). For notation simplicity, we use $\boldsymbol{\theta}_{\mathcal{G},T}$ and $\boldsymbol{\phi}_{\mathcal{G},T}$ to represent $\boldsymbol{\theta} = f_{\boldsymbol{\beta}^{\mathrm{A}}}(\boldsymbol{h}_{\mathcal{G}}, T)$ and $\boldsymbol{\phi} = f_{\boldsymbol{\beta}^{\mathrm{C}}}(\boldsymbol{h}_{\mathcal{G}}, T)$, respectively (similarly, we use $\boldsymbol{\theta}^{\mathrm{old}}_{\mathcal{G},T}$ and $\boldsymbol{\phi}^{\mathrm{old}}_{\mathcal{G},T}$ to represent $\boldsymbol{\theta}^{\mathrm{old}} = f_{\boldsymbol{\beta}^{\mathrm{A,old}}}(\boldsymbol{h}_{\mathcal{G}}, T)$ and $\boldsymbol{\phi}^{\mathrm{old}} = f_{\boldsymbol{\beta}^{\mathrm{C,old}}}(\boldsymbol{h}_{\mathcal{G}}, T)$, respectively). Then, the first three terms in


(a) **Scale-Free Map**


(b) **Singapore Map**

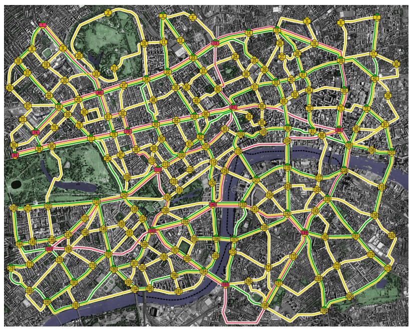

(c) **Scotland-Yard Map**

Figure 7: The underlying graphs used in experiments.

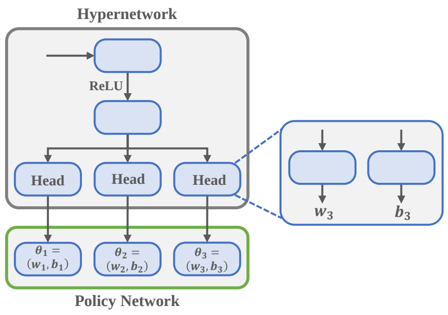


Figure 8: Structure of the hypernetwork and policy network.

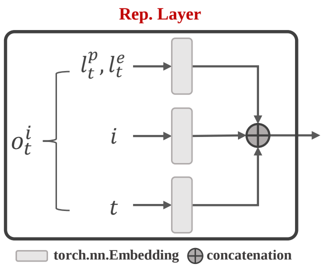


Figure 9: Structure of observation representation layer.

the right-hand-side of Eq.(1) are:

$$L_t^1(\boldsymbol{\theta} = f_{\boldsymbol{\beta}^{\mathrm{A}}}(\boldsymbol{h}_{\mathcal{G}}, T)) = \mathbb{E}\Big[\min\big(\mathrm{ratio}_t\delta_t, \mathrm{clip}(\mathrm{ratio}_t, 1-\epsilon, 1+\epsilon)\delta_t\big)\Big],$$

$$\mathrm{ratio}_t = \frac{\pi^p_{\boldsymbol{\theta}_{\mathcal{G},T}}(a_t^i|o_t^i)}{\pi^p_{\boldsymbol{\theta}^{\mathrm{old}}_{\mathcal{G},T}}(a_t^i|o_t^i)},$$

$$L_t^2(\boldsymbol{\phi} = f_{\boldsymbol{\beta}^{\mathrm{C}}}(\boldsymbol{h}_{\mathcal{G}}, T)) = \Big(V^p_{\boldsymbol{\phi}_{\mathcal{G},T}}(o_t) - \sum_{t'=t}^{T} r^p_{t'}\Big)^2,$$

$$\mathcal{H}(\pi_{\boldsymbol{\theta}=f_{\boldsymbol{\beta}^{\mathrm{A}}}(\boldsymbol{h}_{\mathcal{G}},T)}(\cdot|o_t^i)) = -\mathbb{E}_{a_t^i\sim\pi^p_{\boldsymbol{\theta}_{\mathcal{G},T}}}\log\pi^p_{\boldsymbol{\theta}_{\mathcal{G},T}}(a_t^i|o_t^i),$$

where o_t^i is the observation of the pursuer member $i \in p$, o_t is the global observation shared by all the pursuer members, and δ_t is the TD error [22]:

$$\delta_t = r_t^p + \gamma V^p_{\boldsymbol{\phi}_{\mathcal{G},T}}(o_{t+1}) - V^p_{\boldsymbol{\phi}_{\mathcal{G},T}}(o_t),$$

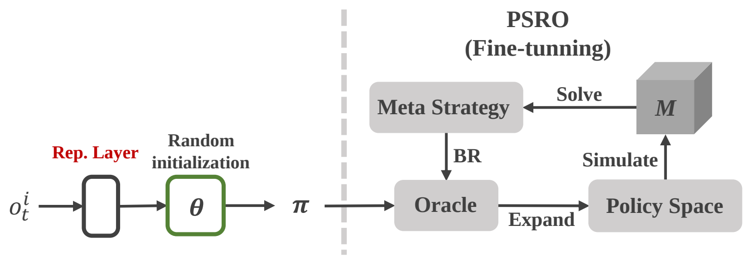
where $r_t^p = 0$ before the game ends and $r_t^p \in \{-1, +1\}$ when the game ends. The expectation is taken on a finite batch of experiences sampled from the episode buffer. For the last term in the right-hand-side of Eq.(1), the action distribution for the reference policy $\hat{\pi}^p$ is constructed by setting the probability of the reference action to 1 while all others to 0. For implementation, we use Dijkstra's algorithm in the package "NetworkX" to derive the reference policy.

B.5 Baselines

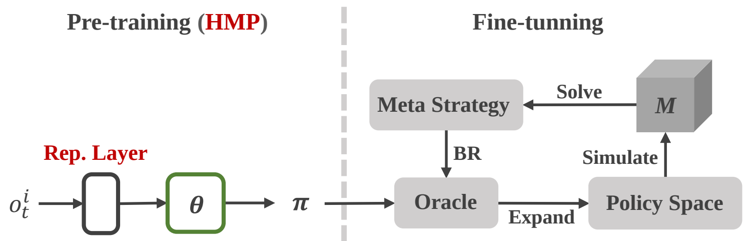
PSRO. See Algorithm 1 in the main text and Figure 10(a) for the standard PSRO algorithm. The only new component is the observation representation layer. It randomly initializes the BR policy at each epoch, i.e., trains the pursuer's BR policy from scratch.

MT-PSRO. The state-of-the-art (SOTA) baseline adapted from [16] by generating a larger number of RL tasks for pre-training. In these tasks, except for the change in the evader's policy, the game's initial conditions also change. The architecture and training pipeline of MT-PSRO is shown in Figure 10(b). MT-PSRO also includes the observation representation layer and employs the HMP method to train the networks. Therefore, it is one of the strong baselines that is expected to possess the ability to solve different PEGs since it is trained by using the data sampled from different PEGs.

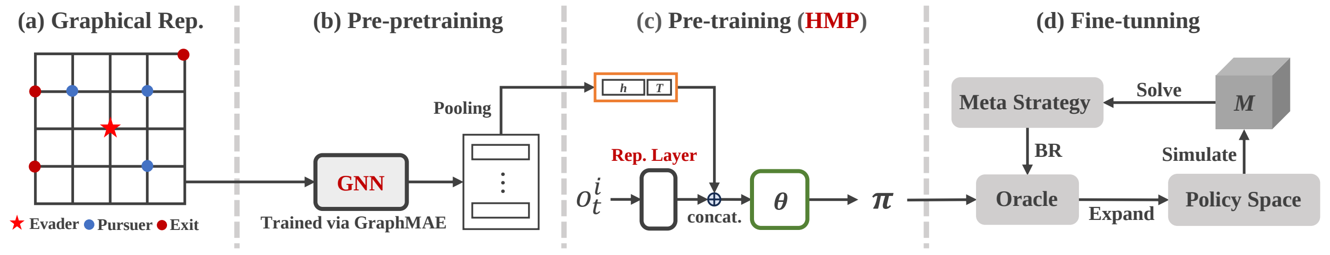
MT-PSRO-Aug. Although MT-PSRO is the SOTA method, comparing it with our Grasper could be somewhat unfair as no information about the initial conditions has been explicitly taken into account in MT-PSRO. To induce a fair comparison, we introduce a novel baseline, MT-PSRO-Aug, as shown in Figure 10(c). MT-PSRO-Aug also employs the same three-stage training scheme as Grasper. Differently, instead of using a hypernetwork to generate the base policy, the hidden vector of the game (obtained by the trained GNN) and the time horizon are concatenated to the output of the observation representation layer to get the (augmented) input to the policy



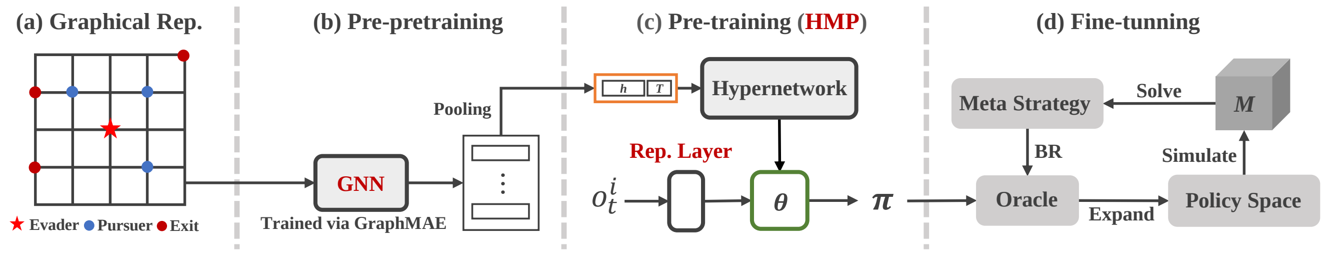
(a) **PSRO**



(b) **MT-PSRO**



(c) **MT-PSRO-Aug**



(d) **Grasper**

Figure 10: Architectures and training pipelines of (a) PSRO, (b) MT-PSRO, (c) MT-PSRO-Aug, and (d) Grasper. This figure clearly shows the differences between these methods. PSRO follows the standard procedure where the pursuer's BR policy is randomly initialized at each epoch. MT-PSRO is a direct adaption from previous works. MT-PSRO-Aug naively leverages the information of the initial conditions to pre-train the base policy. Our Grasper more effectively leverages the information through a hypernetwork which generates the base policy conditional on the initial conditions.

network. As MT-PSRO-Aug also integrates the information about the initial conditions of PEGs, it is another strong baseline that is expected to be capable of solving different PEGs with varying initial conditions. More importantly, comparing Grasper with MT-PSRO-Aug can demonstrate the necessity of introducing a hypernetwork and thus, the superiority of our Grasper in solving diverse PEGs.

Random. Pursuers randomly select actions at any time step.

C ADDITIONAL EXPERIMENTAL RESULTS

In this section, we provide more results to complement the experiments presented in the main text.

C.1 Evaluation Performance

In Figure 4 in the main text, for all methods, we set a fixed number of episodes (10 episodes) for the PSRO procedure (fine-tuning stage). From the results, we can see that given the same fine-tuning budget, Grasper can start from and converge to a higher worst-case utility. Here, we evaluate from another perspective: how long it will take the baselines to achieve a similar utility to Grasper. For this purpose, we keep the number of fine-tuning episodes of Grasper unchanged while increasing the number of fine-tuning episodes of PSRO, MT-PSRO, and MT-PSRO-Aug.

The results for different underlying maps are respectively shown in Figure 11 to Figure 14 and the corresponding fine-tuning budgets are listed in Table 4. As is expected, PSRO requires a much more fine-tuning budget to achieve a competitive performance with Grasper, showcasing the effectiveness of the pre-training and fine-tuning paradigm in accelerating the PSRO procedure, which is also observed in previous work [16]. Furthermore, even though the two strong baselines (MT-PSRO and MT-PSRO-Aug) also (partly or fully) employ the same techniques as Grapser, they are still struggling in solving diverse PEGs with varying initial conditions, since during pre-training, they either entirely ignore or only naively integrate the information about the initial conditions of PEGs. These results again verify the superiority of Grasper over the baselines in terms of solution quality and generalization ability.

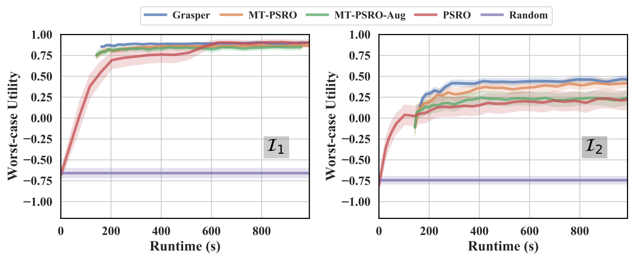


Figure 11: Worst-case utilities of different methods under different fine-tuning budgets in Grid Map.

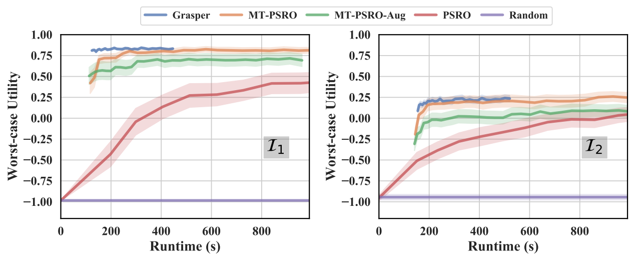


Figure 12: Worst-case utilities of different methods under different fine-tuning budgets in Scale-Free Map.

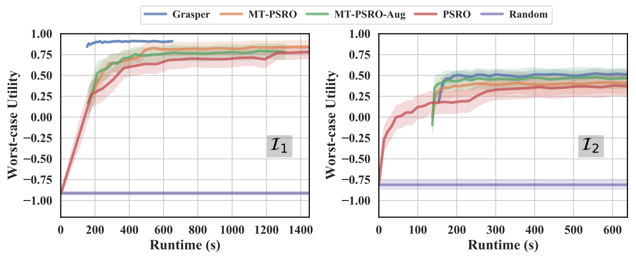


Figure 13: Worst-case utilities of different methods under different fine-tuning budgets in Singapore Map.

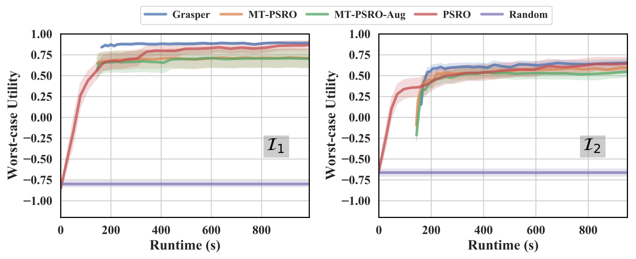


Figure 14: Worst-case utilities of different methods under different fine-tuning budgets in Scotland-Yard Map.

Table 4: The number of episodes for fine-tuning for different methods corresponding to Figure 11 to Figure 14.

		Grasper	MT-PSRO	MT-PSRO -Aug	PSRO
$\mathcal{I}_1$	Grid	10	10	10	1000
	Scale-Free	10	100	100	3000
	Singapore	10	1500	1500	3000
	Scotland-Yard	10	100	100	1000
$\mathcal{I}_2$	Grid	10	25	10	200
	Scale-Free	10	30	30	2000
	Singapore	10	30	30	200
	Scotland-Yard	10	15	10	200

C.2 Effectiveness of Different Components

In Table 1 in the main text, we study the effectiveness of different components in the grid map. In Figure 15 to Figure 18, we plot the evaluation curves for different maps, and in Table 5 to Table 8, we present the final worst-case utilities of different methods in the corresponding maps (Table 5 is a copy of Table 1).

The results clearly demonstrate that the two components – HMP and observation representation layer (Rep.) – are indispensable for Grasper to achieve superior performance. As mentioned in Section 4.1, using mere index numbers of vertices is not a good choice for representing the players' positions in the pursuers' observations. In 5 out of 8 cases (Grid $\mathcal{I}_1$, Singapore $\mathcal{I}_1$ and $\mathcal{I}_2$, Scotland-Yard $\mathcal{I}_1$ and $\mathcal{I}_2$), we can see that using the observation representation layer can significantly boost the performance of Grasper.

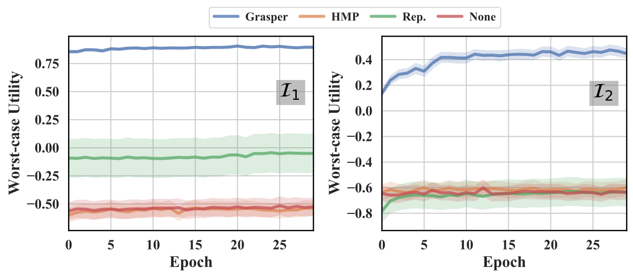


Figure 15: Ablation studies in Grid Map.

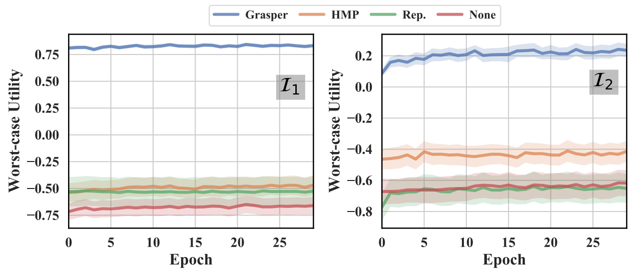


Figure 16: Ablation studies in Scale-Free Map.

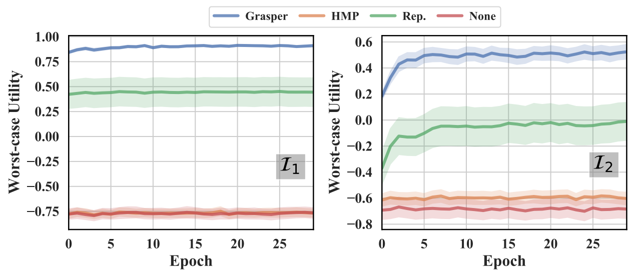


Figure 17: Ablation studies in Singapore Map.

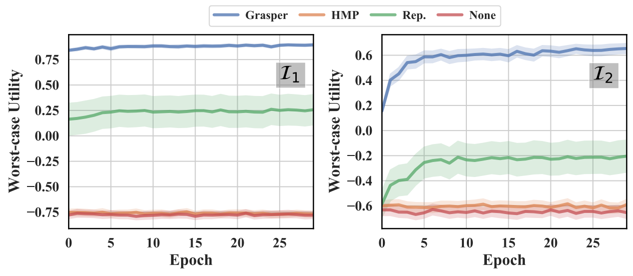


Figure 18: Ablation studies in Scotland-Yard Map.

Table 5: Ablation studies in Grid Map (copy of Table 1).

	HMP	Rep.	Utility
$\mathcal{I}_1$	✓	✓	**0.90 ± 0.01**
	✓		−0.54 ± 0.06
		✓	−0.05 ± 0.17
			−0.52 ± 0.08
$\mathcal{I}_2$	✓	✓	**0.45 ± 0.04**
	✓		−0.60 ± 0.06
		✓	−0.64 ± 0.11
			−0.63 ± 0.06

Table 6: Ablation studies in Scale-Free Map.

	HMP	Rep.	Utility
$\mathcal{I}_1$	✓	✓	**0.83 ± 0.01**
	✓		−0.45 ± 0.09
		✓	−0.52 ± 0.13
			−0.66 ± 0.08
$\mathcal{I}_2$	✓	✓	**0.24 ± 0.04**
	✓		−0.41 ± 0.06
		✓	−0.65 ± 0.09
			−0.62 ± 0.08

Table 7: Ablation studies in Singapore Map.

	HMP	Rep.	Utility
$\mathcal{I}_1$	✓	✓	**0.91 ± 0.01**
	✓		−0.78 ± 0.04
		✓	0.44 ± 0.14
			−0.77 ± 0.05
$\mathcal{I}_2$	✓	✓	**0.52 ± 0.06**
	✓		−0.60 ± 0.05
		✓	−0.01 ± 0.15
			−0.68 ± 0.07

Table 8: Ablation studies in Scotland-Yard Map.

	HMP	Rep.	Utility
$\mathcal{I}_1$	✓	✓	**0.89 ± 0.01**
	✓		−0.77 ± 0.03
		✓	0.26 ± 0.15
			−0.78 ± 0.03
$\mathcal{I}_2$	✓	✓	**0.65 ± 0.04**
	✓		−0.59 ± 0.04
		✓	−0.24 ± 0.13
			−0.65 ± 0.05

C.3 Training Curves

For completeness, we plot the curves of pre-pretraining and pre-training in Figure 19 and Figure 20, respectively.

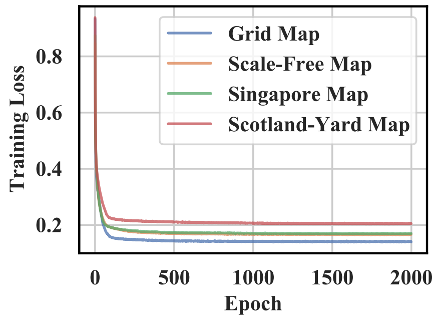
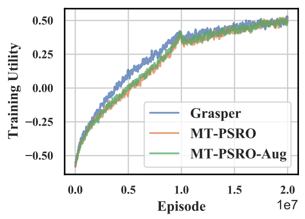
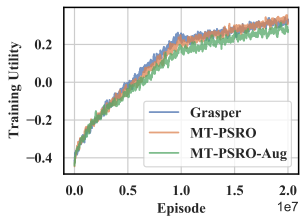


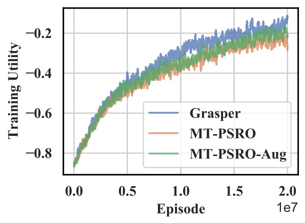
Figure 19: Pre-pretraining curves for different maps.



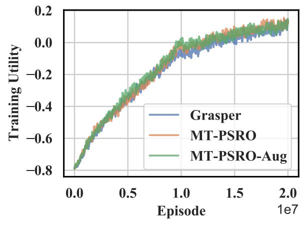
(a) **Grid Map**



(b) **Scale-Free Map**



(c) **Singapore Map**



(d) **Scotland-Yard Map**

Figure 20: Pre-training curves for different maps.

REFERENCES

[1] Micah Adler, Harald Räcke, Naveen Sivadasan, Christian Sohler, and Berthold Vöcking. 2003. Randomized pursuit-evasion in graphs. *Combinatorics, Probability and Computing* 12, 3 (2003), 225–244.

[2] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.

[3] Anthony Bonato. 2011. *The Game of Cops and Robbers on Graphs.* American Mathematical Soc.

[4] Shaunak D Bopardikar, Francesco Bullo, and Joao P Hespanha. 2008. On discrete-time pursuit-evasion games with sensing limitations. *IEEE Transactions on Robotics* 24, 6 (2008), 1429–1439.

[5] Prosenjit Bose, Jean-Lou De Carufel, and Thomas Shermer. 2022. Pursuit-evasion in graphs: Zombies, lazy zombies and a survivor. In *ISAAC*.

[6] Jie Chen, Wenzhong Zha, Zhihong Peng, and Dongbing Gu. 2016. Multi-player pursuit-evasion games with one superior evader. *Automatica* 71 (2016), 24–32.

[7] Harmender Gahlawat, Zin Mar Myint, and Sagnik Sen. 2023. Cops and robber on variants of retracts and subdivisions of oriented graphs. *arXiv preprint arXiv:2307.00584* (2023).

[8] Morgan Gaither, Mark Gabriele, Nancy Andersen, Sean Healy, and Vivian Hung. 2017. Pursuit technology impact assessment, Version 1.1. *Final Report to the US Department of Justice* (2017).

[9] Aric Hagberg, Pieter Swart, and Daniel S Chult. 2008. *Exploring network structure, dynamics, and function using NetworkX.* Technical Report. Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[10] Karel Horák and Branislav Bošanskỳ. 2017. Dynamic programming for one-sided partially observable pursuit-evasion games. In *ICAART*. 503–510.

[11] Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. 2022. GraphMAE: self-supervised masked graph autoencoders. In *KDD*. 594–604.

[12] Linan Huang and Quanyan Zhu. 2021. A dynamic game framework for rational and persistent robot deception with an application to deceptive pursuit-evasion. *IEEE Transactions on Automation Science and Engineering* 19, 4 (2021), 2918–2932.

[13] Rufus Isaacs. 1965. *Differential Games. A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization.* John Wiley & Sons, Inc.

[14] Manish Jain, Dmytro Korzhyk, Ondřej Vaněk, Vincent Conitzer, Michal Pěchouček, and Milind Tambe. 2011. A double oracle algorithm for zero-sum security games on graphs. In *AAMAS*. 327–334.

[15] Pengdeng Li, Xinrun Wang, Shuxin Li, Hau Chan, and Bo An. 2023. Population-size-aware policy optimization for mean-field games. In *ICLR*.

[16] Shuxin Li, Xinrun Wang, Youzhi Zhang, Wanqi Xue, Jakub Černý, and Bo An. 2023. Solving large-scale pursuit-evasion games using pre-trained strategies. In *AAAI*. 11586–11594.

[17] Shuxin Li, Youzhi Zhang, Xinrun Wang, Wanqi Xue, and Bo An. 2021. CFR-MIX: Solving imperfect information extensive-form games with combinatorial action space. In *IJCAI*. 3663–3669.

[18] Xiuxian Li, Min Meng, Yiguang Hong, and Jie Chen. 2022. A survey of decision making in adversarial games. *arXiv preprint arXiv:2207.07971* (2022).

[19] Victor G Lopez, Frank L Lewis, Yan Wan, Edgar N Sanchez, and Lingling Fan. 2019. Solutions for multiagent pursuit-evasion games on communication graphs: Finite-time capture and asymptotic behaviors. *IEEE Transactions on Automatic Control* 65, 5 (2019), 1911–1923.

[20] Arunesh Sinha, Fei Fang, Bo An, Christopher Kiekintveld, and Milind Tambe. 2018. Stackelberg security games: Looking beyond a decade of success. In *IJCAI*. 5494–5501.

[21] Robert Socha and Bogusław Kogut. 2020. Urban video surveillance as a tool to improve security in public spaces. *Sustainability* 12, 15 (2020), 6210.

[22] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement Learning: An Introduction.* MIT press.

[23] Tamás Vicsek. 2010. Closing in on evaders. *Nature* 466, 7302 (2010), 43–44.

[24] Rene Vidal, Omid Shakernia, H Jin Kim, David Hyunchul Shim, and Shankar Sastry. 2002. Probabilistic pursuit-evasion games: Theory, implementation, and experimental evaluation. *IEEE Transactions on Robotics and Automation* 18, 5 (2002), 662–669.

[25] Yuanda Wang, Lu Dong, and Changyin Sun. 2020. Cooperative control for multi-player pursuit-evasion games with reinforcement learning. *Neurocomputing* 412 (2020), 101–114.

[26] Wanqi Xue, Bo An, and Chai Kiat Yeo. 2022. NSGZero: efficiently learning non-exploitable policy in large-scale network security games with neural Monte Carlo tree search. In *AAAI*. 4646–4653.

[27] Wanqi Xue, Youzhi Zhang, Shuxin Li, Xinrun Wang, Bo An, and Chai Kiat Yeo. 2021. Solving large-scale extensive-form network security games via neural fictitious self-play. In *IJCAI*. 3713–3720.

[28] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. 2022. The surprising effectiveness of PPO in cooperative multi-agent games. In *NeurIPS Datasets and Benchmarks Track*. 24611–24624.

[29] Youzhi Zhang, Bo An, Long Tran-Thanh, Zhen Wang, Jiarui Gan, and Nicholas R Jennings. 2017. Optimal escape interdiction on transportation networks. In *IJCAI*. 3936–3944.

[30] Youzhi Zhang, Qingyu Guo, Bo An, Long Tran-Thanh, and Nicholas R Jennings. 2019. Optimal interdiction of urban criminals with the aid of real-time information. In *AAAI*. 1262–1269.