A Experiment Setup

The training procedure of ROA-Star includes a supervised learning stage and a 50-day multi-agent reinforcement learning stage. We also implement an experiment with AlphaStar's setup for comparison and trains with the same computation resources. Due to the limitation of budget, the comparison experiment lasts for 10 days which is still a valid baseline as the main agents always get transitive improvement [Vinyals et al., 2019]. Besides, we conduct a few ablation experiments to evaluate the impact of each component in ROA-Star, which are enumerated in Appendix B.2. Each ablation experiment was trained for 5 days. All these experiments are applied in the race Protoss and all the training of reinforcement learning is restricted on the Kairos Junction map. In this section, we introduce the basic settings of ROA-Star.

A.1 Human Dataset

Blizzard is releasing a large number of 1v1 replays played on the ladder. The instructions for how to download the replay files can be found at https://github.com/Blizzard/s2client-proto. We extracted a dataset from these replays which contains 120,938 Protoss vs. Protoss replays from StarCraft II versions 4.8.2 to 4.9.3. These replays were played by human players with MMR scores greater than 4100.

We utilize the dataset of human replays to learn a good initiation checkpoint for reinforcement learning. After 5 days of supervised learning, the model trained on the full game of StarCraft II can defeat the built-in elite AI with a win rate of 90%. We also train an opponent prediction model on this dataset. The opponent prediction model converges after half-day training, its performance on the test set will be exhibited in Appendix C.4.

Human replays are also used to construct the strategy set $\mathcal{D}$ for the league training. In order to select a set that can cover the effective strategies, we extract strategy statistic z from each human replay in the dataset and cluster all z using the edit distance between their build orders. We sample from each cluster equally to ensure the diversity of selected z. Finally, we obtain 193 different z which constitute the strategy set $\mathcal{D}$.

A.2 Reinforcement Learning

In the reinforcement learning stage, we reward the agents with the win-loss outcome, z-related pseudo-rewards and scouting reward. Similar to AlphaStar's configuration, the z-related pseudo-rewards measure the edit distance between executed and target build orders, as well as the Hamming distance between executed and target cumulative statistics on the units, buildings, and technologies. When agents condition on no extra z, we disable all z-related pseudo-rewards. It's worth noting that ERE replace the edit distance reward with the curricular reward on the build orders.

We apply RL techniques similar to those used in AlphaStar. To perform asynchronous and off-policy updates, we use V-trace algorithm [Espeholt et al., 2018], as well as the self-imitation algorithm (UPGO, Oh et al. 2018). We also apply a standard entropy regularization loss and a policy distillation loss distilling from the last reset target, i.e. the historical MA model for EIE, and the supervised model for other agents. We apply an additional strategy-guided loss for ERE to help learn under-explored strategies. The overall loss we used in the reinforcement learning stage is shown below.

$$\mathcal{L}_{RL} = \mathcal{L}_{V\text{-}trace} + \mathcal{L}_{UPGO} + \mathcal{L}_{entropy} + \mathcal{L}_{distill} + \mathcal{L}^{*}_{ERE}$$

A.3 League Setting

ROA-Star consists of four simultaneously training agents in its league: one MA, one ME, and two LE, where the exploiters are categorized into ME and LE by the different ways of getting opponents. The MA trains with strategy statistic z sampled from our strategy set $\mathcal{D}$, and we set z to zero 10% of the time. A frozen copy of MA is added as a new player to the league with a period of every 2×10^8 steps. The LE agent fights with the whole league and adds a frozen copy into the league when it defeats all the players in the league with a win rate above 70% or reaches the timeout threshold of 2×10^8 steps. At this point, its parameters will be reset with a 25% probability. ME aims to find the weakness of MA, it adds the frozen copy in the league and reset parameters when defeating MA in more than 70% of games or after a timeout of 4×10^8 steps.

So far, the league setting is almost the same with AlphaStar. In ROA-Star, we reform both ME and LE to goal-conditioned exploiters. During the training, each exploiter will be reset to various configurations with a proportion of 20% origin unconditional exploiter, 30% EIE, and 50% ERE. EIE reset to the current MA model. It samples from the z set with the top 10% win rate. ERE reset to the supervised model and condition on top 15% z in execution deviation. All the MA and the exploiters combine the opponent strategy embedding into their observations and get rewards from the scouting behaviors.

A.4 APM Limits

There exists a physical limit for human players on the actions per minute (APM) they can execute. To ensure fairness, we set limitations on the operating frequency for AI that the agent should successively execute actions with a minimum decision interval of 3 frames (around 130 milliseconds). The average APM of any of our final agents' models is less than 240 (with a peak APM below 800), which is close to the human players on Battle.net according to AlphaStar [Vinyals et al., 2019].

A.5 Infrastructure

To scale up league training, we utilize a distributed learner-actor framework depicted in Figure 14. Actors are deployed on CPU machines to interact with the StarCraft II environments, perform action inference and generate training samples. Meanwhile, learners are deployed on GPU machines to update the model parameters using these samples. League Manager is distributed across both the GPU and CPU, maintains the win rates of all historical models, and allocates training tasks to learners and actors, including the agent's own model, the opponent's model, and z of both sides.

For each agent, the full scale of computational resources contains 64 NVIDIA v100 GPUs and 4600 CPU cores. Each actor worker occupies two CPU cores. About 2400 StarCraft II environments are used simultaneously to provide training samples for an agent. An agent processes about 11000 environment steps per second.

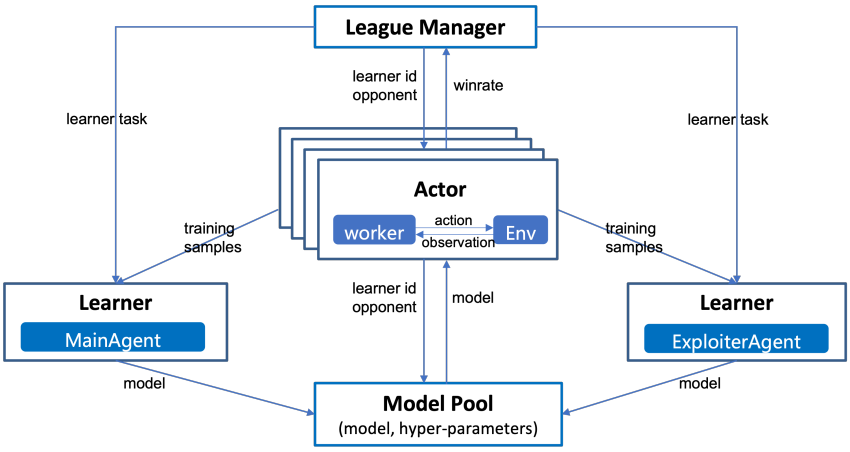

Figure 14: The framework of league training in ROA-Star.

B Evaluation Details

B.1 Human Evaluation

To evaluate the robustness of ROA-Star, we invite three top professional players: Jieshi, Cyan and Macsed to play 20 matches each with ROA-Star. The matches take place on the Kairos Junction map, with both sides using the race Protoss. All of these professional players major in Protoss, and according to Aligulac[4], their world rankings in Protoss are 19, 25, and 39, respectively. They are champions of many StarCraft II professional competitions, including Dreamhack StarCraft II Masters China and StarCraft II World Championship Series China.

The 20 matches against every professional player were divided into 2 times, with an interval of one week in between. 10 matches were played at a time, with a 3-minute break between matches and a 20-minute break after 5 consecutive matches. Professional players could watch replays against ROA-Star and think about their strategy during each break. The average duration of each game was approximately 10 minutes, with the shortest being 3 minutes and the longest being 18 minutes.

To express our gratitude towards the professional players and motivate them to win, we offered them two options for calculating test fees before the test and allowed them to choose the one that suited them best. The first method is that the test fee for each match is 100 RMB, and the second method is based on the result of each match: professional players receive 150 RMB when they win the game, otherwise, they will only receive 50 RMB. Finally, Cyan chose the first method, while Macsed and Jieshi chose the second method.

In the end, we invited herO, the champion of DreamHack SC2 Masters 2022 Atlanta and the second-ranked professional player in Protoss according to Aligulac[4], to play two best-of-three (BO3) matches against our agent as the final benchmark. Prior to the competition, we made an agreement that herO would be rewarded with 100 dollars for every BO3 victory, and no payment would be made if he loses. In the end, we won the first BO3 with a score of 2:0, and lost the second BO3 with a score of 1:2, showing ROA-Star is competitive with the best human player in the world.

B.2 Robustness Evaluation between AIs

It's hard to directly measure the models' robustness because of the vast space of cyclic, non-transitive strategies in StarCraft II. Instead, we apply the Round Robin tournament on the set of models to be evaluated, with any two models in the set playing 100 matches. Based on the performance of the models in these matches, we conducted robustness evaluations on different models and populations.

We conducted tournaments on two model sets. The first set includes the first 5 days' MA models of all the ablation experiments, including:

- **AlphaStar**: The original AlphaStar we replicated.
- **AlphaStar+exploiters with random z**: Reform the unconditional exploiters in AlphaStar to learn random strategies sampled from strategy set D.
- **AlphaStar+EIE+ERE**: Reform the unconditional exploiters in AlphaStar to EIE and ERE.
- **AlphaStar+ERE**: Reform the unconditional exploiters in AlphaStar to ERE.
- **AlphaStar+opponent modeling**: AlphaStar add opponent modeling.
- **Roa-Star**

We demonstrate the robustness evaluations on the first set in Section 5.1, Section 5.2, and Appendix C.1.

The second model set contains all the first 10 days' models in the league of AlphaStar and our ROA-Star, including the models generated by MA and exploiters. The robustness evaluations on the second set are shown in Section 5.3.

[4] http://aligulac.com/periods/343/?page=1&race=p&nats=all&sort=vp

C Supplementary Experimental Results

C.1 The Ablation Study about EIE and ERE

We conduct an ablation experiment to verify both EIE and ERE could contribute to the robustness of MA.

AlphaStar+ERE: During the training, we reset the exploiters to the configuration of ERE with a probability of 80%, and to the original unconditional exploiters with a probability of 20%.

We exhibit the Elo curves of each MA with different exploiters in Figure 8, ERE is superior to random selection in strategy but MA can still benefit from EIE to get further improvement.

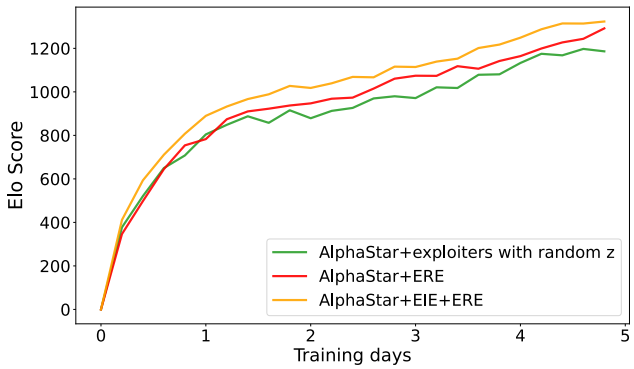

Figure 8: Comparison of the different settings of exploiters.

C.2 Learning Process of Explorative Exploiters

Explorative Exploiters are designed to learn the strict build orders in z, which is especially useful for z that are currently under-explored. In this section, we select a set of z with high execution deviation and compare the learning efficiency of exploiters on these z with various settings. For a specific z, we exhibit the learning process of each entity in the sequence by showing the increase in their execution precision. The execution precision of the n-th entity refers to the ratio of successful executions of the n-th entity after the successful execution of the first $n-1$ entities. We represent each entity in a different color in sequence in the following figures in this section.

Take the strategy Proxy Stargate for example, with its build order as "Gateway->Assimilator->Assimilator->Gateway->Cyberneticscore->Stargate->Adept->Adept->Gateway->Voidray->Shieldbattery->Shieldbattery->Shieldbattery->Voidray->Voidray->Nexus->Voidray". Figure 9 compare the learning efficiency of the exploiter on this strategy with different learning settings. With the help of z-related curricular reward and strategy-guided loss, the agent learns to execute the strategy defined by z effectively and accurately.

In Figure 10, we provide the learning process of the build orders on another six z as shown in Table 4 under the setting of Explorative Exploiters. Within $4 * 10^8$ steps, which is consistent with the training steps of ME, all the entities specified by target z achieve an execution precision above 50%. In Table 4, we also provide a specific comparison of the last entity execution precision of the AlphaStar z-related pseudo-rewards setting and the ERE setting.

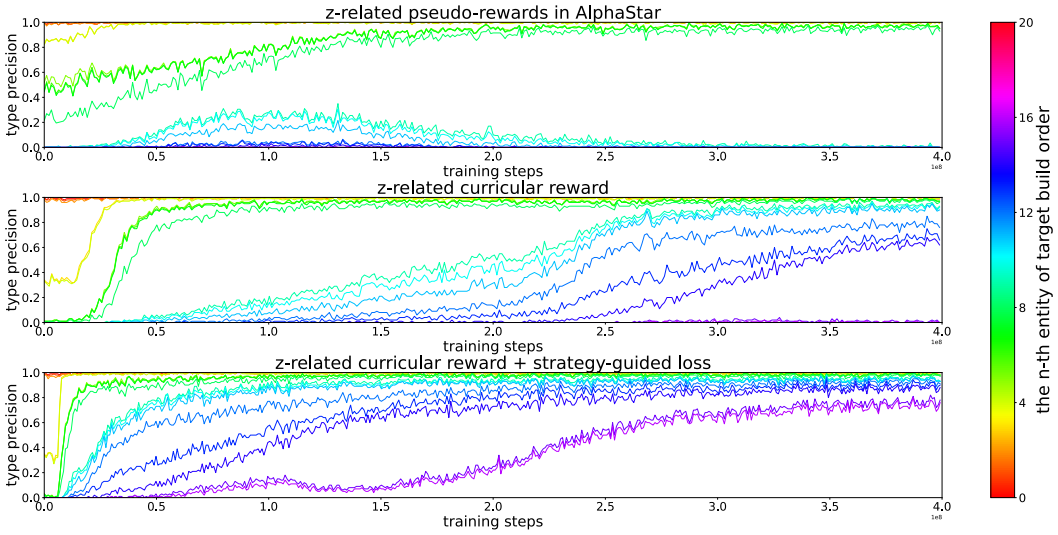

Figure 9: Comparison of the learning process on an under-explored strategy (Proxy Stargate) with different exploiter settings. The top one is the exploiter with the original z-related pseudo-rewards in AlphaStar. The middle one is the exploiter equipped with the z-related curricular reward. The bottom one is the final Explorative Exploiter with the z-related curricular reward and strategy-guided loss.

Table 4: The execution precision of the last entity in target build order after training of $4 * 10^8$ steps. **AlphaStar** refers to training the exploiter with z-related pseudo-rewards as in AlphaStar. **ERE** refers to the exploiter settings in ERE, including the z-related curricular reward and strategy-guided loss.

index	target build order	last entity execution precison	
		AlphaStar	ERE
0	Forge->Assimilator->Gateway->Assimilator->Gateway ->Cyberneticscore->photoncannon->Gateway->Stalker->Stalker ->Stargate->Shieldbattery->Shieldbattery->Shieldbattery ->photoncannon->Stargate->Voidray->Shieldbattery ->Shieldbattery->Voidray	0	0.69
1	Gateway->Assimilator->Assimilator->Gateway->Cyberneticscore ->Roboticsfacility->Stalker->Stalker->Warpprism->Roboticsbay ->Stalker->Stalker->Shieldbattery->Nexus->Disruptor->Stalker ->Stalker->Shieldbattery->Disruptor->Stalker	0	0.69
2	Gateway->Assimilator->Assimilator->Gateway->Cyberneticscore ->Adept->Adept->Adept->Adept->Nexus->Adept->Adept ->Shieldbattery->Adept->Adept->Roboticsfacility->Sentry ->Sentry->Sentry->Sentry	0	0.80
3	Gateway->Assimilator->Assimilator->Gateway->Cyberneticscore ->Twilightcouncil->Adept->Adept->Adept->Adept->Darkshrine ->Gateway->Nexus->Shieldbattery->Darktemplar->Darktemplar ->Roboticsfacility->Stalker->Stalker->Stalker	0	0.50
4	Gateway->Assimilator->Assimilator->Gateway->Cyberneticscore ->Adept->Adept->Stargate->Stalker->Stalker->Shieldbattery ->Stalker->Oracle->Stalker->Stalker->Shieldbattery	0	0.60
5	Gateway->Assimilator->Assimilator->Gateway->Cyberneticscore ->Adept->Adept->Stargate->Stalker->Stalker->Oracle->Nexus ->Oracle->Roboticsfacility->Twilightcouncil->Stalker->Stalker ->immortal->Stalker->Stalker	0	0.67

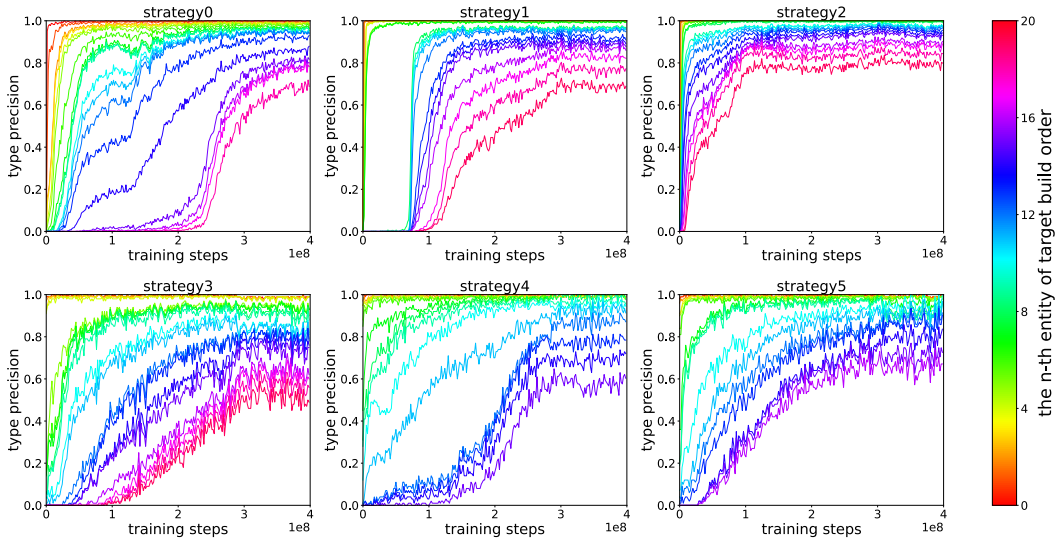

Figure 10: 6 cases of the Explorative Exploiters' learning process on under-explored strategies.

C.3 The Impact of Opponent Modeling on Scouting Ability

We measure the effectiveness of each scouting behavior with the opponent prediction model and reward the agent accordingly. To show the impact of scouting rewards, we compare the time consumed for the agents to discover the opponent's newly-built buildings. We made two 5-day MA models trained with/without opponent modeling play against each other for 2000 matches. To encounter diverse opponents, both two models randomly pick z from set $\mathcal{D}$ to execute. Then we calculate the average interval from the opponent constructing a new building until the agent discovers it. As shown in Figure 11, the scouting reward remarkably reduces the time for discovering the building of opponent under the fog.

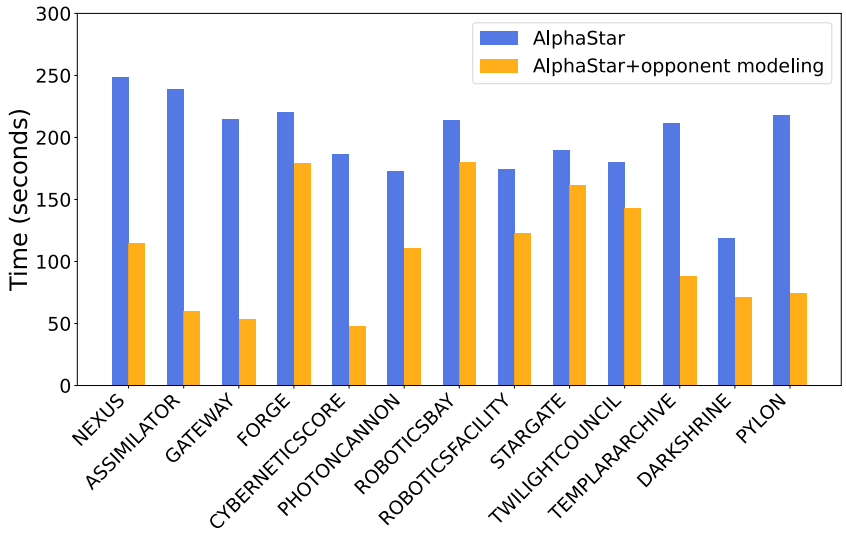

Figure 11: The consumed time to discover opponent's newly-built buildings.

C.4 Opponent Prediction Model Performance

In this section, we exhibit the performance of opponent prediction model on the test set of 3000 human replays.

We make quantity predictions for the current opponent's entities, including those in production. We categorize the quantity of each entity type into buckets, where buildings are grouped into 0, 1, 2, and greater than 2 categories, and military units are divided into 0, 1, 2, 3-4, and greater than 4 categories. We use multi-classification heads to predict these categories and calculate the macro f1-score of these tasks on the test set, as shown in Figure 12.

Another critical piece of the opponent's strategy is its technical route. Once the performance of the opponent military unit is observed, it is possible to directly infer whether the opponent has upgraded a certain technology, such as "Charge" which can increase the attack speed of Zealots. However, at this time the optimal timing to respond is often missed. Instead, it is more meaningful to predict the technologies under research. As they are binary classification tasks with extremely imbalanced data, we measure the ability of the opponent prediction model with average precision (AP), as shown in Figure 12. The mean average precision (mAP) of prediction on all technologies is 0.73.

The construction location of key buildings, specifically whether they are constructed outside of the base, determines if the opponent is using proxy strategies. Therefore, we utilize binary classification to predict whether the opponent is constructing or has already constructed proxy buildings. The AP of each type is shown in Figure 12 and the mAP of all proxy building types is 0.75.

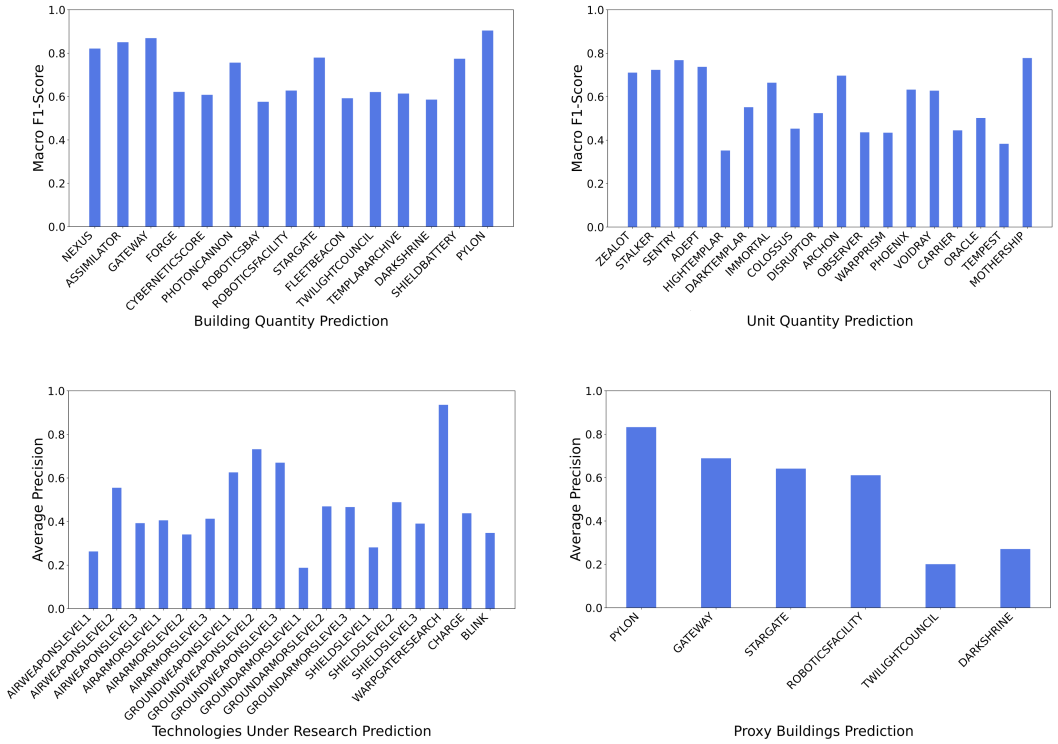

Figure 12: The performance of opponent prediction model on the test set.

C.5 Strategy Diversity in Exploiters

Intuitively, the robustness of MA would benefit from the strategy diversity in exploiters. In this section, we evaluate the strategy diversity in exploiters of two leagues, ROA-Star and AlphaStar, in terms of both qualitative and quantitative measures.

To get a vectorized description of each model generated by the exploiters during the first 10 days, we can have them play 100 matches against a common opponent, such as the 3-day MA model in AlphaStar. For each model, we calculate the average statistics of entities and technologies on the matches to generate a vectorized description. Then we analyze their strategies by applying k-means clustering to these vectors. The clustering result, shown in Figure 13, displays each model as a point in the 2d space after their vector dimensions have been reduced using t-SNE. The exploiters in

ROA-Star can explore more strategies than AlphaStar within the same training time and resources, e.g. the proxy strategies.

We also quantitatively analyze the diversity in exploiters like Determinantal Point Process (DPP, Kulesza and Taskar [2012]) dose. DPP measures the diversity of a candidate set by calculating the determinant of a kernel matrix that describes the similarity between each item pair in the candidate set. To get the similarity matrix, we calculate an L2-distance d, then use the kernel function $\exp(-\frac{d}{T})$ to transform the distance into similarity, T is a scale factor which we set to 3 here. The final DPP scores of ROA-Star and AlphaStar are shown in Table 5. The numerical comparison of the DPP support that exploiters in ROA-star are more diverse than AlphaStar.

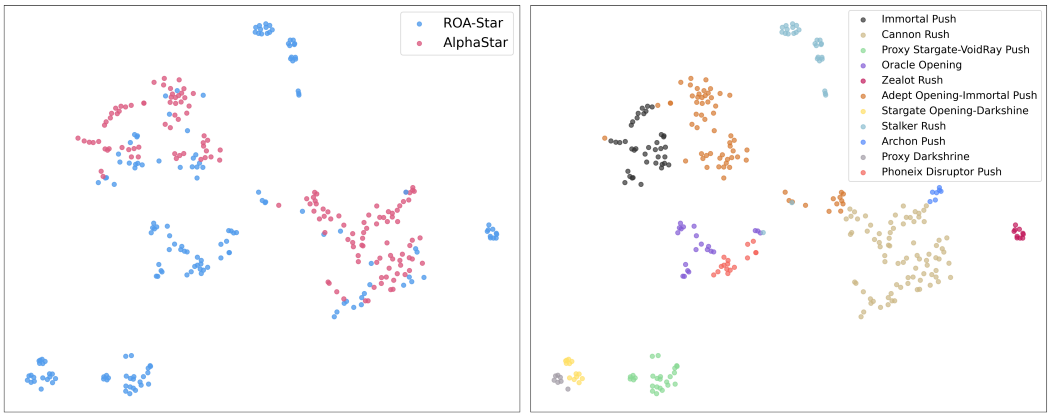

Figure 13: Left visualizes the models in exploiters of AlphaStar and ROA-Star in the 2d space with t-SNE. Right colors the left points with the result of K-means clustering on the models.

Table 5: Diversity in Exploiters

	AlphaStar	ROA-Star
DPP score	0.002	**0.229**