Reinforcement Learning Methodologies for DeepMind's StarCraft-2 Mini-game Environment

Dimitrij Krstev
Ss. Cyril and Methodius University,
Faculty of Computer Science and Engineering
Skopje, North Macedonia
dimitrij.krstev@students.finki.ukim.mk

Kristijan Panchevski
Ss. Cyril and Methodius University,
Faculty of Computer Science and Engineering
Skopje, North Macedonia
kristijan.panchevski@students.finki.ukim.mk

Sonja Gievska
Ss. Cyril and Methodius University,
Faculty of Computer Science and Engineering
Skopje, North Macedonia
sonja.gievska@finki.ukim.mk

Martina Tosheska
Ss. Cyril and Methodius University,
Faculty of Computer Science and Engineering
Skopje, North Macedonia
martina.tosheska@finki.ukim.mk

***Abstract*—This paper presents a reinforcement learning (RL) approach for the BuildMarines mini-game in StarCraft II, leveraging Dataset Aggregation (DAgger) followed by pure RL fine-tuning. Initially, the agent is trained using DAgger on an imperfect expert, enabling efficient policy initialization and mitigating compounding errors in sequential decision-making. The pre-trained model is then further optimized with reinforcement learning, allowing it to surpass the expert's performance through self-improvement. This hybrid approach significantly accelerates training while achieving superior final rewards compared to baseline methods that rely solely on RL.**

Our results demonstrate that combining imitation learning with reinforcement learning leads to more sample-efficient training and improved performance in complex real-time strategy environments.

I. Introduction

Real-time strategy (RTS) games present significant challenges for artificial intelligence due to their complex decision spaces, partial observability, and the need for both tactical and strategic reasoning. Several game environments have emerged as benchmarks for AI research in this domain, with StarCraft II (SC2) becoming a prominent standard. For our research, we will be focusing specifically on DeepMind's PySC2 environment, which provides a Python interface for SC2, including customized mini-game environments, each focusing on certain part of SC2's game mechanics.

More specifically, this research focuses on applying Dataset Aggregation (DAgger), an iterative imitation learning algorithm, combined with Proximal Policy Optimization (PPO) Reinforcement Learning (RL) to train agents in a custom PySC2 mini-game environment. Our approach leverages DAgger's ability to address distribution mismatch problems in Behavioral Cloning (BC) while using PPO for policy refinement, with hyper-parameter optimization handled through Optuna. This hybrid methodology aims to overcome common challenges in RTS AI, such as the complexity of sequential decision-making, the need for strategic planning, and the challenge of learning from expert demonstrations.

The implementation specifically targets the "Build Marines" mini-game scenario, providing a focused test case for evaluating the effectiveness of combining imitation learning with reinforcement ledarning in a complex, partially observable environment.

II. Related Work

While undoubtedly a novel field of study, in recent years there have been a numerous significant papers specifically relating to RL models in RTS games, such as StarCraft II. One of the main studies focused on the broad implementation of LSTM-CNNs, as demonstrated by Vinyals et al. [2], which served as a primary inspiration for our research. The complexity of the StarCraft II environment has made it an excellent testbed for advancing AI capabilities in difficult-to-master real-time strategy scenarios.

Following these foundational developments, significant advancements have emerged in StarCraft II AI research. Liu et al. [3] demonstrated efficient RL approaches for the full-length game, while Hu and Wang [4] focused on implementing and benchmarking the SC2LE environment using actor-critic methods. A notable hierarchical approach was presented by Xu et al. [5], incorporating human expertise for subgoal selection, while Kelly and Churchill [6] explored transfer learning between different RTS combat scenarios.

In terms of learning methodologies, two particularly relevant approaches informed our research direction. Daniels et al. [7] advanced the field by building upon the DAgger algorithm for lifelong reinforcement learning in StarCraft II, while Gleave et al. [8] developed robust implementations of imitation learning that have proven valuable for RTS game environments. These approaches form the methodological foundation for our current research.

III. Methodology

This section outlines our approach, environment configuration, and key implementation details for training our reinforcement learning model, which incorporates imitation learning through DAgger.

Initially, we experimented with pure RL but observed no meaningful progress even after 11,520,000 timesteps. This prompted us to adopt a more structured three-stage process: (1) DAgger imitation learning from an imperfect expert, (2) hyperparameter fine-tuning using Optuna, and (3) continued refinement through reinforcement learning. This core idea of this structure is for the agent to initially build a solid policy from a good reward-averaging expert, and then eventually surpass the expert policy with continued RL.

A. StarCraft II Environment Setup

A pivotal part in this model's training was DeepMind's PySC2 environment, which provides a Python interface to StarCraft II. This framework exposes a comprehensive API that translates the continuous game statistics into actionable observations and allows for a suitably-defined action space. The "Build Marines" mini-game focuses on resource management and production, requiring agents to construct buildings and train marine units within a fixed time frame. The specific setup includes:

- **Observation Space**: Multi-dimensional feature maps representing unit types, player resources, and other game state information.
- **Action Space**: A structured action space comprising $[6, 84, 84]$ dimensions, representing 6 discrete action types that can be performed at various screen coordinates within an 84×84 grid.
- **Reward Structure**: A sparse reward signal based strictly on the number of marines trained by the end of an episode.

The feature map observation space is optimized for convolutional neural networks (CNNs). However, since we use Stable-Baselines3 (SB3) for model training and RL, we opt for the RGB observation map instead. This choice aligns with SB3's NatureCNN model, which requires standard image-based inputs. Additionally, the StarCraft II environment is wrapped inside a Gymnasium interface to ensure seamless integration with SB3's training framework.

B. DAgger Imitation Learning Implementation

The core of our methodology is the Dataset Aggregation (DAgger) algorithm, which played a central role in overcoming the original shortcomings of our initial traditional purely reinforcement learning approach. It achieves this by iteratively collecting imperfect expert demonstrations that consistently achieve an award of 78 and training a policy on an aggregated dataset:

1) **Expert Policy Definition**: Our expert policy follows a clearly defined structured decision-making process of 6 chosen actions out of thousands of possible environment actions of which not all are available at a given state, consisting of the following, in order of selection importance:
   - Builds supply depots when population cap is near
   - Trains a marine unit (Only action that leads to reward after full completion)
   - Select barrack (Prerequisite for marine training)
   - Selects worker SCVs (Prerequisite for building actions)
   - Constructs barracks for training marines
   - No action (Conserve resources)
2) **DAgger Training Process**: Our approach follows the standard DAgger algorithm workflow where an initial policy generates trajectories, the expert provides corrective actions for these states, and the dataset is iteratively expanded with these state-action pairs. The policy is then retrained on this augmented dataset, with expert intervention gradually reduced over time through a decay factor β.
3) **Beta Schedule**: We implemented a high β value of 0.9 to maintain strong expert reliance during training, gradually allowing the learned policy to make more decisions over time.

C. Policy Optimization with PPO

Following the DAgger training phase, we employed Proximal Policy Optimization (PPO) for further refinement:

1) **Model Architecture**: Uses a CNN policy architecture suited to processing the spatial observations from StarCraft II.
2) **Hyperparameter Optimization**: We used Optuna for systematic hyperparameter tuning, searching over the following parameters:
   - Learning rate: 1×10^{-5} to 1×10^{-3} (log scale)
   - Entropy coefficient: 1×10^{-8} to 1×10^{-1} (log scale)
   - Discount factor (γ): 0.5 to 0.9999
3) **Training Process**: Our PPO training workflow leverages the DAgger-pretrained agent as a starting point. It then systematically explore different hyperparameter configurations using Optuna to optimize performance. Each candidate model undergoes extensive training for 1,000,000 timesteps, followed by evaluation across 10 complete episodes to measure effectiveness. After identifying the most promising model based on mean reward metrics, we conduct additional refinement through 144,000 more timesteps of training (equivalent to 10 full episodes).

D. Experimental Configuration

Our experiments were conducted with the following specifications:

1) **Training Duration**: DAgger pretraining for 825,000 timesteps, followed by PPO optimization for up to 1,440,000 timesteps.

2) **Evaluation Metrics**: Performance was measured primarily by the score (number of marines produced) and success rate in completing the mini-game scenario.
3) **Optuna Trials**: 20 complete trials with different hyper-parameter configurations, each saving model checkpoints for comparison, after an additional training of 35,000 timesteps and a 10 episode evaluation.

This comprehensive approach allows us to effectively transfer expert knowledge into a reinforcement learning framework, addressing the challenge of sparse rewards and complex state spaces in the StarCraft II environment.

The full source-code for all mentioned methodologies can be found at the following link:

https://github.com/Kristijan885/ABS-SC2-RL-Bot/tree/master/mini_games/buildMarinest

IV. Results and Analysis

A. Imitation learning

The initial DAgger-trained PPO model achieves an average reward of 4.2 with a standard deviation of 6.16 in 50 episodes. It was trained on 858,000 timesteps with a gradually reduced beta schedule leading to higher expert reliance early on in training and encouraging further exploration there after. The BC trainer is initialized with a batch size of 64 and 1×10^{-4} l2 weight. The beta schedule is reduced by N / 100000 for each N step, with a minimum of 0.1

B. Hyper-parameter Tuning

The optimal hyper-parameter configuration identified through our search was:

- **Learning rate**: 0.00013646368493582003
- **Entropy coefficient**: 1.656328137928305e-06
- **Gamma**: 0.9360938386281064

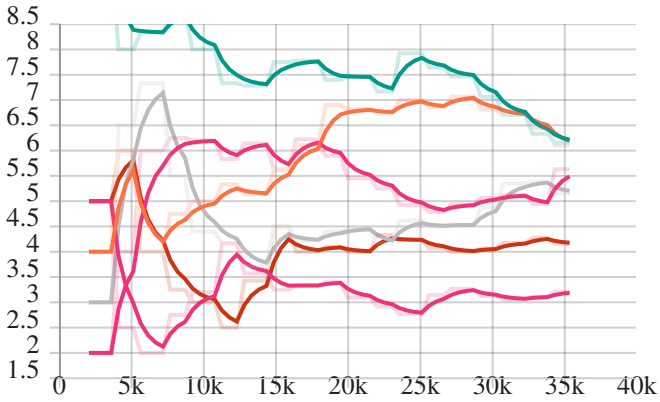

Fig. 1. Mean trial reward per environment step

Figure 1 represents a few notable trial instances and their mean reward yields per environment timestep. Optuna recommends hyper-parameters depending on the previously gained average trail rewards. In light of this, the best trial was trial 12 (in teal) with a mean reward of 6.22 and a slightly better, comparable performance to trial 14's (in orange) 6.12, which in hindsight seems to have had a more stable learning experience.

C. Reinforcement Learning

With the previous best-reward averaging model loaded from the Optuna trials, we continue with an additional 1,440,000 training timesteps, which lead to the following training metrics:

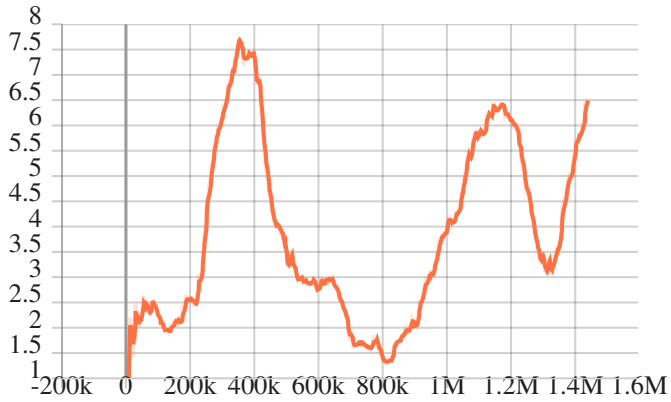

Fig. 2. Mean reward per environment step

The agent initially learns well, reaching a performance peak around 400k timesteps, but experiences a decline by 800k, possibly due to exploration-exploitation trade-offs or suboptimal policy updates. However, performance recovers and starts improving significantly just before training stops at 1.4 million timesteps, suggesting that the transition to pure RL fine-tuning was effective.

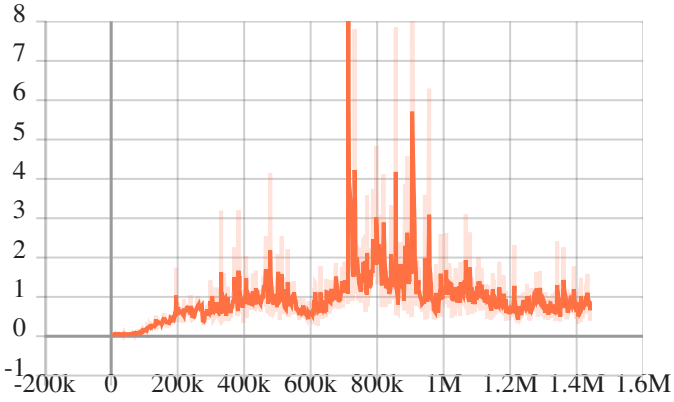

Fig. 3. Approximate Kullback-Leibler divergence per environment step

Figure 3 shows mostly stable KL divergence indicates consistent policy updates. However, the sharp spike at 700k and a smaller peak at 900k suggest moments where the updated policy diverged significantly from previous iterations, likely due to larger learning rate adjustments or drastic policy shifts during training, possibly where RL adjusted the agent's strategy significantly.

Entropy loss measures the agent's action uncertainty. The initial peak at 400k suggests high exploration, which drops at

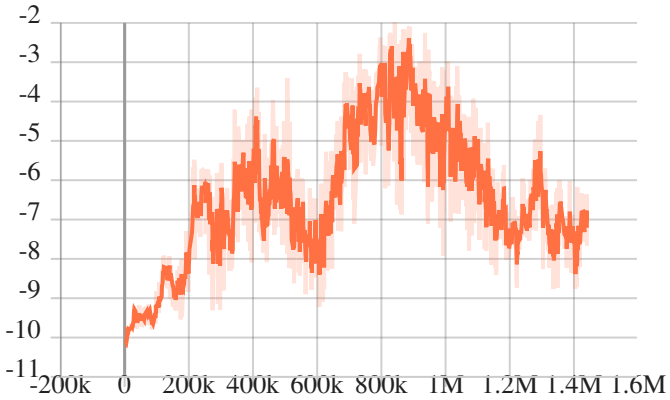

Fig. 4. Entropy loss per environment step

600k as the policy stabilizes. The second peak at 800k could indicate renewed exploration, followed by a steady decline, suggesting that the agent is becoming more confident in its decisions and converging to a more deterministic policy.

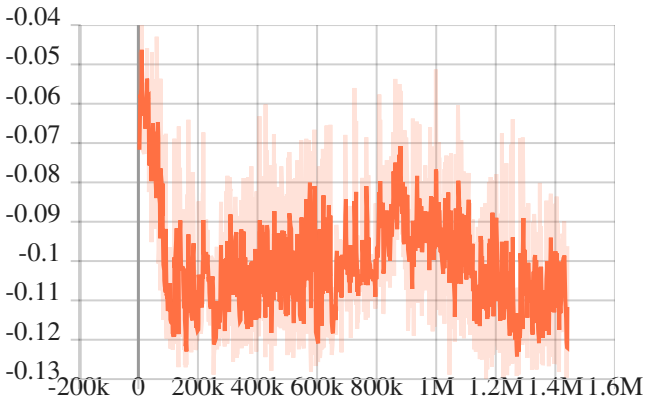

Fig. 5. Policy gradient loss per environment step

The high initial policy loss aligns with the early training phase when large policy updates occur. It drops and stabilizes, indicating reduced gradient updates as the policy matures. The small peak at 900k may correspond to a policy correction phase, possibly related to the earlier KL divergence spike.

All graphs have a scalar smoothing factor of 0.6 for improved readability. The results are gathered from 1.4 million environment time-steps, totaling to 24 hours of training time.

V. Conclusion

In this paper, we presented a three-stage training pipeline for reinforcement learning in DeepMind's StarCraft II Build Marines mini-game, combining DAgger imitation learning, Optuna-based hyperparameter fine-tuning, and PPO reinforcement learning. This approach allowed us to accelerate early training, optimize key hyperparameters, and further refine the agent's decision-making policy through direct environment interactions.

Initially, DAgger imitation learning enabled the agent to learn from an imperfect expert, providing a strong baseline policy. However, due to a limited number of DAgger episodes, the agent struggled to reach an expert-level reward before transitioning to RL, which could have improved sample efficiency.

To enhance training efficiency, we employed Optuna for hyperparameter tuning, aiming to find optimal learning rates and PPO parameters. However, the small trial count constrained the search space, potentially leaving better configurations undiscovered.

Finally, we fine-tuned the agent using PPO reinforcement learning, allowing it to surpass the limitations of imitation learning and achieve superior long-term performance. Compared to similar research relying solely on RL, our approach achieved a higher final reward in significantly fewer training steps, demonstrating the effectiveness of integrating imitation learning and automated hyperparameter optimization.

Future work should focus on increasing the number of DAgger episodes to ensure the agent reaches a comparable reward to the expert before RL training, as well as conducting a more extensive hyperparameter search with Optuna to maximize PPO training efficiency. These improvements could further reduce training time and enhance final agent performance.

Acknowledgment

The authors would like to thank the Faculty of Computer Science and Engineering at Ss. Cyril and Methodius University for their support of this research.

References

[1] S. Ross, G. Gordon, and D. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 627–635.

[2] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, and others, "StarCraft II: A New Challenge for Reinforcement Learning," arXiv preprint arXiv:1708.04782, 2017.

[3] R. Liu, Z. Pang, Z. Meng, W. Wang, Y. Yu, and T. Lu, "On efficient reinforcement learning for full-length game of StarCraft II," Journal of Artificial Intelligence Research, vol. 75, pp. 213–260, 2022.

[4] H. Hu and Q. Wang, "Implementation on benchmark of SC2LE environment with advantage actor–critic method," in 2020 International Conference on Unmanned Aircraft Systems (ICUAS), 2020, pp. 362–366.

[5] X. Xu, T. Huang, P. Wei, A. Narayan, and T.-Y. Leong, "Hierarchical reinforcement learning in StarCraft II with human expertise in subgoals selection," arXiv preprint arXiv:2008.03444, 2020.

[6] R. Kelly and D. Churchill, "Transfer Learning Between RTS Combat Scenarios Using Component-Action Deep Reinforcement Learning," in AIIDE Workshops, 2020.

[7] Z. Daniels, A. Raghavan, J. Hostetler, A. Rahman, I. Sur, M. Piacentino, and A. Divakaran, "Model-free generative replay for lifelong reinforcement learning: Application to StarCraft-2," arXiv preprint arXiv:2208.05056, 2022.

[8] A. Gleave, M. Taufeeque, J. Rocamonde, E. Jenner, S. H. Wang, S. Toyer, M. Ernestus, N. Belrose, S. Emmons, and S. Russell, "imitation: Clean imitation learning implementations," arXiv preprint arXiv:2211.11972, 2022.