Reinforcement Learning to Play Pokemon VGC competitively

Akkshay Sundara Rajan, David Oprea, Govind Bhuttar, Nikhil Abraham, Nikshith Konnuru, Sai Javvadi, Vladislav Pavlovskii

Introduction

Pokemon VGC (the official competitive Pokemon format) is a two player simultaneous turn based game. It can broadly be split into two phases - teambuilding and playing. This project uses reinforcement learning to tackle the piloting problem - playing out the best moves given a team. The project is built using poke-env [1] on the Pokemon Showdown platform.

Challenges

VGC as a game poses some unique challenges for machine learning solutions

- Partial information - We do not have complete information about opponent's teams. Information gets revealed incrementally as the game goes on. We handle this by encoding only observed information into our observation vector that is fed into the policy network.
- Huge team configuration space - The number of possible different teams is estimated at 10^{139} [2]. This makes training non-generalizable. We handle this by limiting the metagame to 6 teams.
- Nondeterministic moves - Each action taken has variable damage and can proc a secondary effect with a fixed probability.

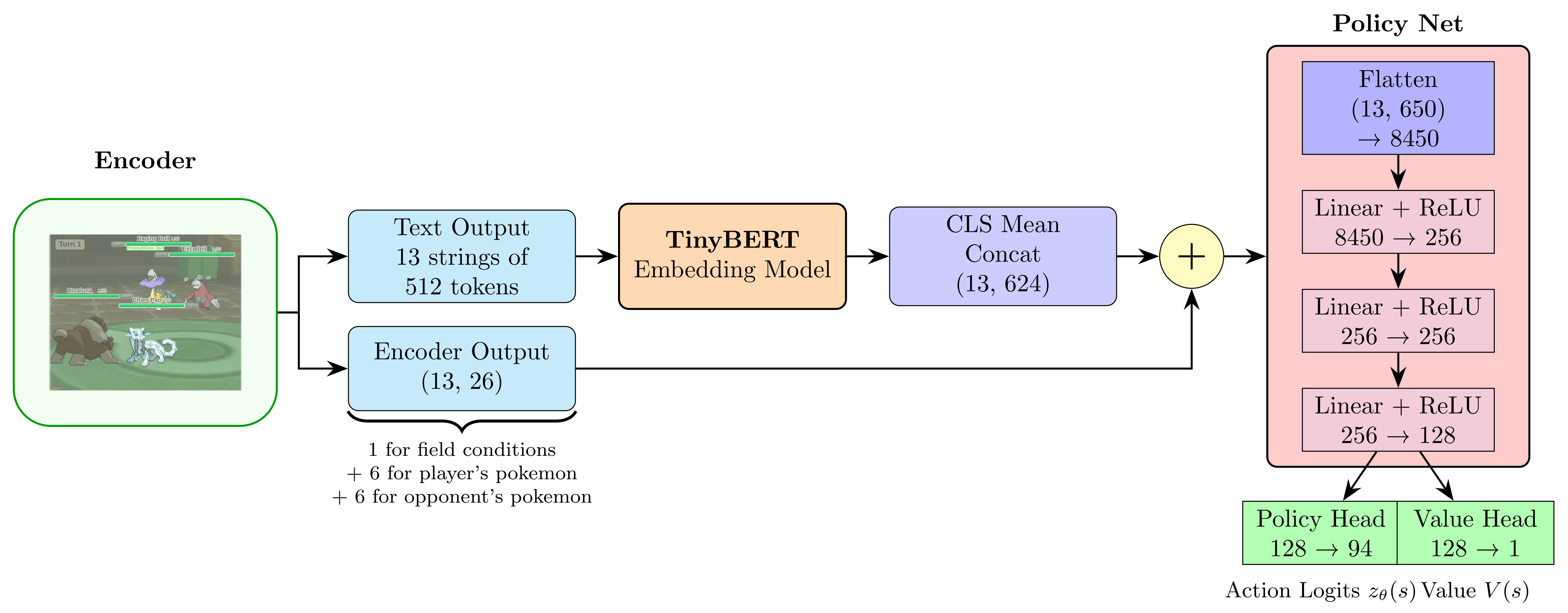
Despite the large variance in the outcome of every turn, VGC is a game that has a factor of skill to it similar to poker (validated by the fact that top players can achieve a win rate $> 70\%$ [3] in tournaments against other professionals).

Other challenges we faced specific to our implementation are:

- Slow CPU Bound training - Majority of the time spent training involves interacting with the Pokemon showdown server.
- Limited interactibility with environment - Poke-env lacks certain features such as saving team preview selections and loading an intermediate state in the battle, which could have allowed for further optimizations.

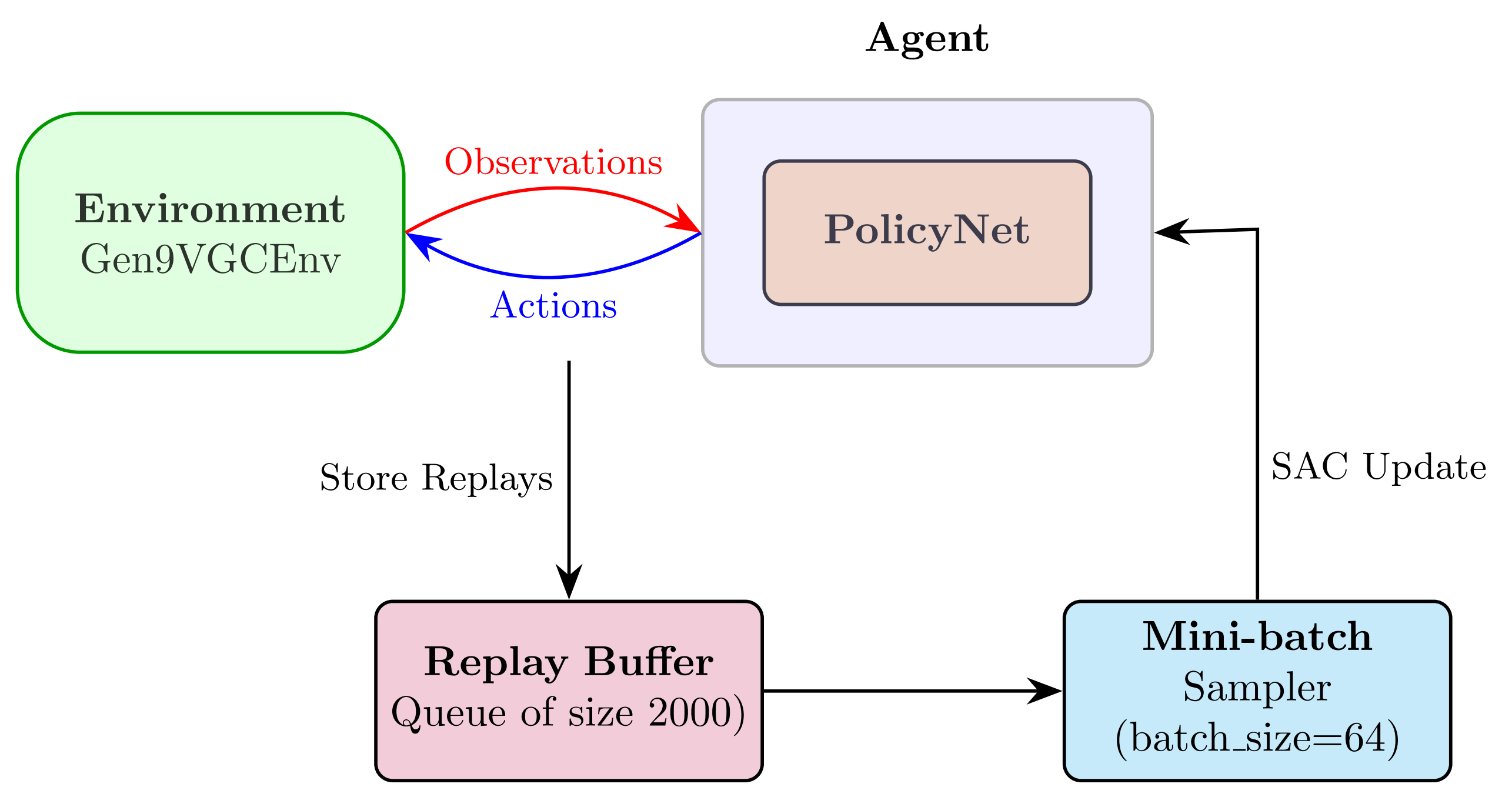
Methodology

Network Architecture



The observation vector uses TinyBERT embeddings to encode Pokemon-specific features, with text indicators for partial observability (e.g., "Moves are unknown"). The encoder also spits out numerical features from the battle such as the boosts / status effect / base stats for each pokemon on the field. The RL agent wraps over the policy net, using the logits + masks of illegal actions to build a probability distribution for the actions that can be taken. The move to be selected is chosen as a random action sampled from the calculated probability distribution. This makes our move selection non-deterministic, which is crucial so that it has the ability to make "reads" instead of chosing the same option given the same state.

Training Approach



We use a simplified Soft Actor-Critic (SAC) with fixed entropy coefficient and off-policy replay buffer for sample efficiency. Two agents with shared policy π_θ engage in self-play with sparse reward signals (± 1 for win / loss), following VGC-Bench's approach [2]. The rationale behind some of the decisions made are as follows:

- **Small Replay Buffer** - Middle ground between the sample efficiency of off-policy methods while still approximating on-policy methods by removing very old replays. Alternatively, we could have used an exponential distribution in the mini-batch sampler that weighs newer replays with a larger probability.
- **Self Play** - With the lack of professional games with the teams selected for usage in the project, methods such as behaviour cloning were not possible. Apart from that, self play training is the standard for most competitive game engines such as Stockfish.

Results

Due to the current implementation having an extremely slow training time, we could only train our model for 1000 battles (for context the VGC-Bench paper trained their model for atleast 0.3 million battles). Despite the minimal training, we see some promising results.

Opponent	Win Rate
Random Move Player	89.2%
Max Base Power Player	38%

It seems that the training has led to a passive playstyle that loses to the offensive max power player. This should be improved by using episodes of training against this heuristic.

Future Plans

- Fine tune the TinyBERT embedding model with relevant Pokemon information to extract more relevant features as part of the input to the policy network.
- Parallelize the training loop to have multiple agents with a global shared policy and global replay buffer which should speed up the CPU bound training loop.
- Expand the network and implement a fleshed out soft actor critic algorithm with a trainable entropy parameter. Due to the discrete nature of the action space, we could also expand our value head to output the Q value for each action instead.
- Expand and increase the depth of the policy network and train the model for a larger number of epochs, employing Raylib for hyperparameter optimization.
- Use a stratified training approach, mixing single agent training against a heuristic with the current self play approach.

Acknowledgement & Citations

We would like to thank the team at ML @ P and Alex for his help throughout this project. This project is also hugely inspired by the VGC-Bench paper and so we are deeply grateful to the authors of it.

[1] Sahovic, H. Poke-env: pokemon AI in python. *https://github.com/hsahovic/poke-env*

[2] Angliss, C., Cui, J., Hu, J., Rahman, A., & Stone, P. (2025). A Benchmark for Generalizing Across Diverse Team Strategies in Competitive Pokémon. *arXiv preprint arXiv:2506.10326.*

[3] PT Ladder VGC Rankings. *https://ptladder.bennbuild.io/ladder/vgc*