Training Super Smash Bros. Melee Agents (Extended Abstract)

Motivation Super Smash Bros. Melee (SSBM), a 2001 crossover fighting video game developed by HAL Laboratory and published by Nintendo for the GameCube, is a fast-paced, multiplayer fighting game with complex physics, high-dimensional continuous controls, and strategic depth. Existing successes of reinforcement learning (RL) on games like Atari, Go, and Chess are not fully applicable to SSBM given its higher temporal resolution and partial observability. Our goal is to explore the extent to which large-scale offline data in the form of replays files from Project Slippi and modern RL techniques can bootstrap competent SSBM agents, and to identify the combination of behavior cloning and reinforcement learning needed to approach human-level performance given data and compute constraints.

Method We first perform offline behavior cloning (BC) on human replay data to initialize agent policies that mimic expert input distributions. We process 525 curated Slippi `.slp` files (where player 1 is always the character Fox and player 2 is another high-skill opponent) into 5-frame state-action windows and train an MLP policy network. Next, we inject the policy into a Dolphin emulator loop for evaluation through key SSBM performance metrics again CPU competition. We then experiment with with two RL policies, namely implicit Q-learning (IQL) and proximal policy optimization (PPO), to refine the policy with reward signals based on win/loss, damage dealt, and stage control in order to learn strategies beyond what can be offered by BC .

Implementation We build infrastructure to process Slippi files and extract frame-wise game state and controller inputs, breaking this dataset 98/1/1 into our train/val/test split. We tested our implementation on smaller MLP models, gradually scaling to a 15 layer and 1024 layer size MLP with leaky-ReLU, optimized via Adam at a learning rate of 1×10^{-4}. For RL finetuning, we employ a parallelized PPO implementation running 32 simultaneous matches, collecting trajectories of length 500 steps. We train on a a c4x4 large and a Google cloud (whatever the specific is). We also built extensive layer between libmelee (an API for interfacing) with an Dolphin emulator in Python with the inputs and outputs of our model architecture.

Results Our BC agent achieves surprisingly impressive performance considering the limited data it was trained on, being able to execute basic combinations and building towards taking off a stock. However, it demonstrates a lack of recovery (taking actions that bring the agent back into the game stage when knocked off) that prevents it from winning against the CPU player (0.375 ± 0.484 kills). Both of our RL techniques (IQL and PPO) make moderate gains above this baseline across a range of performance metrics, but continue to struggle with the issue of recovery.

Discussion Our experiments indicate that BC alone rapidly captures human-like micro-actions but plateaus in competitive performance. RL finetuning yields improvements in strategy and adaptability, validating our hypothesis that hybrid BC + RL pipelines can be effective for training performant agents at smaller scale. Scaling data remain critical, however: deeper architectures and richer reward functions along with that could further enhance performance.

Conclusion We present a framework for training SSBM agents by combining expert offline data with deep reinforcement learning techniques. Our research shows surprisingly promising initial results on trained SSBM agents despite constraints in data and compute, and outlines key challenges such as data scale, reward design, and model architecture design, that will guide future work toward achieving human-level expertise in complex, real-time multi-agent games.

Training Super Smash Bros. Melee Agents

Matthew Lee
Department of Computer Science
Stanford University
mattglee@stanford.edu

William Hu
Department of Computer Science
Stanford University
willhu@stanford.edu

Samuel Do
Department of Computer Science
Stanford University
samdo@stanford.edu

Abstract

Super Smash Bros. Melee (SSBM) is a fast-paced, real-time fighting game with complex physics, high-dimensional continuous controls, and rich strategic depth, characteristics that move beyond traditional reinforcement learning achievements (Atari, Go, Chess). In this work, we explore a hybrid pipeline combining offline behavior cloning (BC) on Project Slippi replay data with deep reinforcement learning (RL) finetuning (IQL and PPO) to train competent SSBM agents under realistic data and compute constraints. We preprocess 525 curated Slippi replays into 5-frame state–action windows, train MLP policies via BC and then refine these policies with RL within a Dolphin emulator using reward signals for wins and damage dealt. While BC agents exhibit strong micro-action imitation, they struggle with recovery and win-rate, which our RL methods partially mitigate to yield moderate performance gains. Our results suggest the potential of BC + RL pipelines for long-horizon, high-frequency control tasks. We highlight key challenges like evaluation, reward design, and model architecture that will guide future efforts toward human-level expertise in complex multi-agent environments.

1 Introduction

Achieving human-like and superhuman performance in games has long served as a benchmark for progress in artificial intelligence. From mastering perfect-information, turn-based board games like Go and Chess with AlphaZero Silver et al. (2017), to tackling high-dimensional visual environments like Atari through Deep Q-Networks and model-based agents like DreamerV2 Hafner et al. (2020), reinforcement learning has proven to be a powerful tool in advancing game-playing AI. However, these successes often depend on environments with low temporal complexity, full observability, and relatively straightforward reward structures. In contrast, real-time fighting games like Super Smash Bros. Melee (SSBM) present unique challenges: high-frequency continuous control, partial observability, sparse and delayed rewards, and complex, multi-agent interactions.

SSBM is a 2001 crossover platform fighter developed by HAL Laboratory and published by Nintendo for the GameCube. Over the past two decades, it has cultivated a devoted competitive scene and an active ecosystem of developers who have built tools for replay logging (Slippi), emulator instrumentation (libmelee), and game state introspection. These advances make SSBM not only a compelling environment for studying advanced decision-making strategies but also an increasingly viable testbed for reinforcement learning research.

Stanford CS224R 2025 Final Report

Despite this promise, prior work in training agents for SSBM has faced significant hurdles. Firoiu et al. Firoiu et al. (2017) pioneered early RL agents by training directly from pixels, but lacked access to large-scale replay data or efficient simulator APIs. More recently, Gu et al. Gu (2025) employed behavior cloning (BC) on billions of frames of expert gameplay, showing strong action prediction but limited in-game success without further reinforcement learning. These works hint at a critical insight: while BC captures expert-like behavior patterns, it struggles with generalization, long-horizon planning, and recovery, all elements crucial to winning in SSBM.

In this paper, we investigate whether combining imitation learning with reinforcement learning under realistic compute and data constraints can produce competitive SSBM agents. Specifically, we explore a hybrid offline learning pipeline: we first train policies using behavior cloning on 525 curated Slippi replays (approximately one million frames) focused on Fox, a fast-paced, combo-oriented character. We then finetune these policies using either Implicit Q-Learning (IQL) or Proximal Policy Optimization (PPO) within a Dolphin emulator loop, using a reward function that balances stocks taken, damage dealt, and game outcomes.

Our contributions are fourfold:

1. We integrate Slippi-Ishiiruka and libmelee into a Gym-style training environment for efficient RL in SSBM.
2. We curate a dataset of Fox vs. high-level opponents and process it into a structured replay buffer for offline learning.
3. We implement and evaluate both offline (IQL) and online (PPO) RL methods atop BC, under tight compute constraints.
4. We conduct quantitative and qualitative analyses of agent performance, identifying key bottlenecks like failure to recover.

Through this work, we aim to understand not only what current RL pipelines can achieve in high-frequency multi-agent environments, but also what is still missing in terms of data, algorithms, and evaluation practices to reach truly expert-level performance.

2 Related Work

2.1 Beating the World's Best at Super Smash Bros. Melee with Deep Reinforcement Learning

Firoiu et al. approached the same task much earlier in 2017, essentially predating the crucial data and infrastructure later efforts were built on (including ours) Firoiu et al. (2017). Without access to a large corpi of expert demonstration data or an API such as libmelee to interact with the SSBM emulator, the authors trained an agent off of the raw pixels of a game frame. They developed their own environment wrappers around the Slippi emulator, collected data via expert replays, and trained using a standard convolutional neural network and reinforcement learning techniques such as PPO. Ultimately, the authors showed that it was feasible to derive vision-based policies for SSBM notwithstanding the key challenges of sample efficiency and frame prediction accuracy.

2.2 Training AI to play Super Smash Bros. Melee

Gu et al. approached training a SSBM bot from a pure behavior cloning perspective (though the authors are currently working on offline and online reinforcement learning methods for the same task) Gu (2025). In particular, Gu et al. used 3 billion frames of SSBM from professional Fox replays to train an agent using behavior cloning. The authors notably discretize controller analog stick input and preprocess the data to transform the task (of predicting controller inputs) to a single-label classification rather than multi-label. For training, the authors used a 20 million parameter decoder-only transformer architecture and set up a harness for closed loop evaluation. While our task and initial approach are extremely similar, we limit ourselves to a much smaller subset of data and computational resources while asking: is it possible to train an agent that does just as good with less data and compute but a smarter approach? Namely, we limit our data set to only contain 525 Slippi .slp files (roughly a million frames) in comparison to the 3 billion Gu et al. used Gu (2025). Furthermore, SSBM poses an especially sparse reward landscape.

2.3 Mastering Atari Games with Discrete World Models

Hafner et al. extended their work in the original DreamerV1 paper (which targeted tasks from the MuJoCo physics simulator) to the task of playing Atari games, achieving state-of-the-art performance at the time Hafner et al. (2020). In DreamerV2, the authors extended previous reinforcement learning methods based on latent-dynamics by learning discrete latent representations for planning and control. They enable the agent to perform long-horizon planning without interacting with the real environment (essentially imagining or dreaming up future states). This proves to be especially sample-efficient since the agent is now able to explore states through its world model rather than wait on a simulator. Furthermore, the agent can attribute positive rewards to actions that, in its imagination, leads to positive future states; this helps address the all-too-common issue of sparse-rewards in reinforcement learning. While we do not adopt a model-based reinforcement learning approach, we grapple with some of the key problems that it aims to address.

3 Method

At a high level, our metholodigcal contributions are fourfold. First, we curate an effective set of train/validation/test data for expert learning of a specific SSBM character. Secondly, we extensively integrate a GameCube emulator into our codebase by building upon open-source libraries towards our specific model constraints so that we can train and evaluate policies. Thirdly, we implement several RL techniques for training agents under relevant policies. Finally, we integrate these pieces into a single train + evaluate workflow.

3.1 Data

3.1.1 Raw Data Format (`.slp` files)

When Melee is played on a certain emulator known as Slippi Dolphin, completed matches are automatically saved as `.slp` (Slippi) files for personal replay and review. These files contain frame-wise game state data, player inputs, etc. without storing the video of the match. `.slp` files are an effective and efficient data representation to which we can apply RL techniques, because the frame-by-frame game state data and player inputs naturally lend them to observation and action samples.

Generating the data ourselves this way would be far too time-consuming, so instead, we rely on a rich existing `.slp` dataset that we believe originated from the project in Gu (2025). This dataset floats around subreddits and Discord servers that research machine learning techniques for Melee, and an anonymous Reddit post notes some important attributes of the dataset:

- "95,102 SLP files."
- "Unzips to 200GB."
- "All tournament sets, with varying skill levels."
- "Pruned to remove handwarmers, doubles, less than 30 second matches."
- "CC0 Licensed, so use it however you want."

The Google Drive link to the dataset is here. This is not our link, so please exercise caution when accessing. The uncompressed form is very large, so we selectively extracted replays using zip command line tools and shell magic.

3.2 Data Curation

The scale of the dataset was beyond what was feasible for us to handle due to training constraints, which is also what inspired our approach for data efficiency and smarter methods. Thus, we pruned the dataset to 525 select replays from the original dataset to train the agents for our project. This also gave us the chance to ensure that our agent was playing the same Melee character each time; the replays were filtered so that Fox was always the first player i.e. Fox versus some other character. We pick Fox as his tactics rely on fast combos with minimal projectiles, traits that we felt would lend to more easily trainable and expressive behaviors.

3.2.1 Data Pipeline

As mentioned, we believe this existing dataset originated from Gu (2025), so we also used Gu's existing replay processor and data loader for processing the `.slp` files in to Numpy arrays of observations and actions per frame. Since Gu was working with the dataset in full, Gu used a data sharding algorithm, but we opted to save the processed data as `.pkl` objects since we were working at a much smaller scale for our project.

The data pipeline saves each processed replay as a `.pkl` object, where each object is a dictionary of fields to Numpy arrays. Each array contains values for that field on each frame. Some example observation fields include `p1_position_x`, `p1_position_y`, `p1_facing`, `p1_action`, etc. Some example action fields include `p1_main_stick_x`, `p1_main_stick_y`, `p1_button_a`, etc.

The replay buffer is then populated by extracting the observations and actions at each frame index from the Numpy arrays.

The default split provided by the data processor was 98% for training, 1% for validation, and 1% for testing i.e. 513 replays for training, 6 for validation, and 6 for testing, knowing the size of our pruned dataset.

3.3 Emulating SSBM

Emulating SSBM is crucial to not only evaluating trained agents on the actual task (i.e. playing SSBM); it is necessary to enable other reinforcement learning techniques beyond behavior cloning. We use a prebuilt Linux emulator AppImage from vladfi1 et al. (2025) of Slippi-Ishiiruka as well as a melee.iso file (that represents a digital copy of the SSBM game). These resources are then specified to libmelee AltF4 et al. (2025), which exposes different functions that enable us to perform game actions (i.e. selecting stages and characters or specifying controller input). In addition to performing actions in the SSBM environment, libmelee also exposes per-frame observations after each timestep. We combine these to form an Gym-like environment for standardized interaction.

Mapping model outputs to controller inputs is simple enough, as we simply map an space of twelve continuous values between 0 and 1 to the twelve GameCube controller inputs relevant to SSBM (main joystick x and y, camera stick x and y, left and right trigger, and six buttons: X, Y, A, B, Z, and D-pad up). Likewise, our model receives as input a vector containing the positions, damage percentages, facing directions, and action states for both player 1 and player 2.

3.4 RL Methods

In order to devise a baseline of performance, we decided to train an agent via our implementation of behavior cloning. We then attempted refining performance with two techniques that we also implemented: implicit Q-learning and proximal policy optimization.

3.4.1 Behavior Cloning

Given the reasonably substantial amount of SSBM replays we have access to, which translates to roughly a million (observation, action) pairs each mapping to a frame of game-play, behavior cloning is a natural option to serve as a good baseline of performance. As mentioned previously, these (observation, action) pairs are grouped into windows of 5 consecutive frames. Groups are then shuffled randomly through the replay buffer, and the model is trained to minimize the negative log likelihood of the ground truth action taken at the last frame based on a normal distribution parameterized by a mean and standard deviation output by the model. The mean-prediction network is a standard MLP with leaky-ReLU as the activation function. The deviation-prediction network is a simple trainable 1D tensor representing the log of the standard deviation.

3.4.2 Reward Function

The reward function is defined as the difference in value between the current and previous states for both players, where the state value incorporates stocks, health, and a large penalty for losing the game. Formally, the state value for a player is:

$$V(s) = 400 \cdot \text{stock}(s) + \text{health}(s) - 10000 \cdot \mathbb{1}_{\text{loss}(s)},$$

where $\text{stock}(s)$ is the number of remaining stocks, $\text{health}(s)$ represents the current percent health (inverted such that lower health implies a higher value), and $\mathbb{1}_{\text{loss}(s)}$ is an indicator function equal to 1 if the player has lost the match in state s, and 0 otherwise.

The reward at timestep t is the change in state value between the current and previous states, computed as the delta between Player 1 and Player 2:

$$r_t = \left(V(s_t^{(1)}) - V(s_{t-1}^{(1)})\right) - \left(V(s_t^{(2)}) - V(s_{t-1}^{(2)})\right),$$

so that higher rewards correspond to improvements for Player 1 relative to Player 2. This reward function was our best attempt at explicitly modeling all the behaviors we thought were important: getting kills, preserving health, and not losing.

3.4.3 Implicit Q-Learning

Implicit Q-Learning (IQL) is an offline reinforcement learning algorithm introduced by Kostrikov et al. (2021) that addresses the distributional shift problem inherent in learning from fixed datasets. Unlike standard actor-critic methods that rely on policy improvement via maximizing the Q-function, IQL avoids explicit policy updates and instead fits a value function and uses it to weight behavior actions. This approach minimizes overestimation and instability from out-of-distribution action queries. The key idea is to train the critic using conservative Bellman updates and compute a value-weighted regression onto actions from the dataset, implicitly biasing learning toward high-value actions without requiring policy gradients.

The Q-function in IQL is trained via a modified temporal-difference loss that omits the policy improvement step:

$$\mathcal{L}_{\text{Q}} = \mathbb{E}_{(s,a,r,s')}\left[\left(Q(s,a) - (r + \gamma V(s'))\right)^2\right],$$

where $V(s) = \mathbb{E}_{a\sim\pi_\beta}\left[\exp\left(\frac{Q(s,a)-V(s)}{\beta}\right)\right]$ implicitly defines the soft value function. IQL's strengths lie in its simplicity, strong empirical performance on offline tasks, and robustness to hyperparameters. However, it may underperform on online or highly exploratory tasks due to its reliance on behavior data and lack of explicit exploration. However, we take advantage of this for SSBM as environmental rollouts are highly costly from a time and compute perspective.

3.4.4 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a widely used on-policy reinforcement learning algorithm proposed by Schulman et al. (2017) that balances learning efficiency and stability. PPO improves upon vanilla policy gradient methods by introducing a clipped surrogate objective that limits how much the policy can change at each update step, thereby preventing destructive policy updates. It is especially favored in simulated environments where reliable exploration is feasible, and sample reuse is limited.

The core objective optimized by PPO is:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ is the advantage estimate. The clipping mechanism controls variance while still allowing sufficient policy updates. PPO is sample inefficient compared to off-policy methods like IQL and struggles in sparse-reward or offline settings. Nevertheless, its stability, simplicity, and performance in high-dimensional continuous control has established it a strong choice for many real-world tasks, and we hypothesized this would generalize to SSBM.

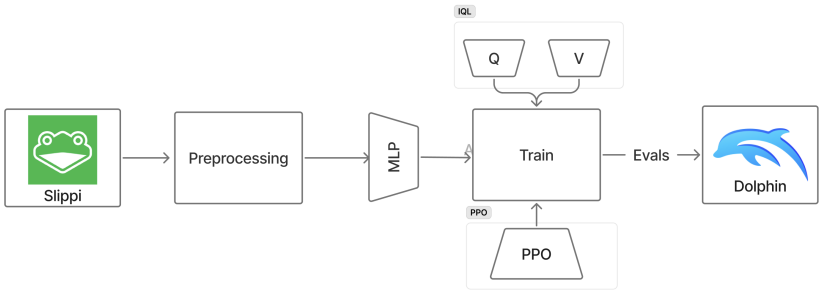


Figure 1: System diagram overview of our method for training SSBM agents.

3.5 System Integration

Visualized in Figure 1, our overall system for training and evaluating policies on SSBM can be described as follows:

- Collect all training Slippi files, preprocessing them and loading them into a replay buffer.
- Use the training data to train a baseline MLP using behavior cloning.
- (Optional) Use RL techniques like IQL or PPO to further refine the MLP policy with the Dolphin emulator.
- Use the Dolphin emulator to evaluation final performance metrics.

4 Experimental Setup

4.1 Behavior Cloning

For BC, we experiment with a number of different model sizes and learning rates for the mean-prediction MLP, iterating between (2, 4, 8) for the number of layers, (128, 256) for the layer sizes, and $(1e-3, 2e-4, 5e-4)$ for the learning rate. As a basic sanity check, we evaluate each BC sub-experiment based on the percentage of predicted actions that matches the ground truth action, up to a set threshold of 0.05, 0.1, and 0.2.

4.2 Evaluating In-Game Performance

After we finished iterating on our baseline model, we proceeded to train three policy classes of agents: (1) BC, (2) BC + IQL, (3) BC + PPO. After training each policy, we play the agent ten times against the default CPU, an effective baseline since it is known for its simple and rather deterministic behavior. We tracked five metrics that are used in SSBM (and in specific, the Slippi project) from these matches to measure performance of our agents:

- **Kills**: The number of stocks that the player is able to take off the other player.
- **Damage**: The percentage damage that the player is able to inflict on the other player.
- **Opening Conversion Rate (OCR) :** The success percentage at turning a hit into another hit.
- **Openings / Kill (O/K) :** How many openings are needed on average before the player takes a stock?
- **Damage / Opening (D/O):** How much damage is done per opening?

Our hyperparameter configuration was as follows: a 15-layer MLP policy network, 1024 hidden units per layer, and a learning rate of 1e-3 with the Adam optimizer, processing 5 frames per observation with a batch size of 512 over 5000 iterations. Training uses a replay buffer of 1 million samples, saves checkpoints every 10 iterations, and validates every 10 iterations.

5 Results

5.1 Behavior Cloning

As shown in Figure 2, a large enough model is capable of predicting sensible actions given game-state observations when the model. Through out experiments, we discovered the key differentiator between models that succeeded and those that did not was the number of layers - MLPs with only 2 layers struggled to output actions that were within 0.1 of ground truth 50% of the time. On the other hand, those with more than 2 layers were within threshold 80% of the time (a MLP trained with 8 layers seemed to learn much faster than the 4-layer variant but tapered off to around the same level of performance). For a threshold of 0.05, models with 2 layers were within threshold only 30% of the time whereas those with more than 2 layers reached roughly 60% as shown in Figure 4. For an even greater threshold of 0.2, 2-layer MLPs were within threshold roughly 75% of the time; MLPs with greater than 2 layers were within threshold 85% of the time as shown in Figure 5. These results ultimately suggest that we need a large enough model to provide it with enough capacity to learn the mapping between observations and actions in SSBM. With that being said, this measure of performance does not necessarily translate to real, in-game performance and motivates more means of evaluation through environment interaction.

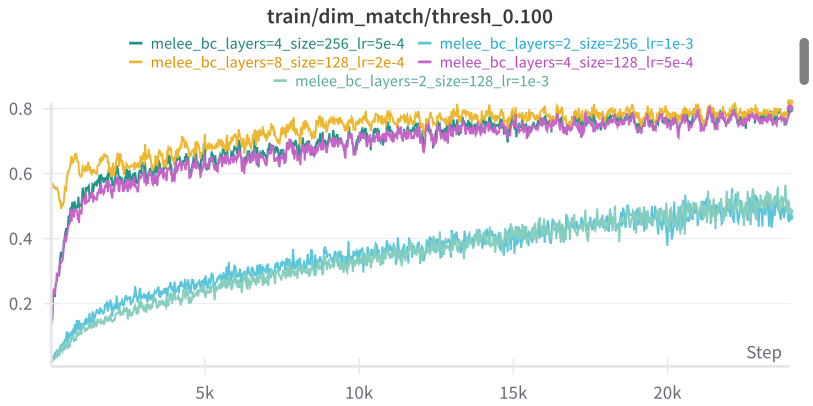


Figure 2: Percentage of action dimensions that match within threshold (0.1)

5.2 Evaluating In-Game Performance

Our results from the performance experiment are summarized in Table 1. The key result, unfortunately, is that our policies were unable to eek out a single win against the CPU across the board.

Approach	Kills	Damage	OCR	O/K	D/O
BC	0.375 ± 0.484	82.00 ± 35.77	$24.54\% \pm 14.33\%$	8.33 ± 2.11	14.40 ± 2.74
+ IQL	0.501 ± 0.471	89.64 ± 39.07	$32.54\% \pm 20.74\%$	6.33 ± 2.05	14.36 ± 3.39
+ PPO	0.528 ± 0.374	92.19 ± 35.04	$33.58\% \pm 18.22\%$	6.45 ± 2.02	14.48 ± 3.60

Table 1: Performance comparison between BC, IQL, and PPO across 10 games.

5.2.1 Statistical Analysis

To expound on specific implications of the statistics reported:

- **Kills and Damage**: These metrics are the most direct indicators of policy performance. Both IQL and PPO yield modest improvements over BC, with IQL increasing kill counts and damage slightly and PPO providing the highest mean performance with lower variance. Given that we are explicitly refining the policy with reward, the marginal gains are surprising.

The answer will be explored in depth later, but suffice to say that recovery is the key barrier to agent success.

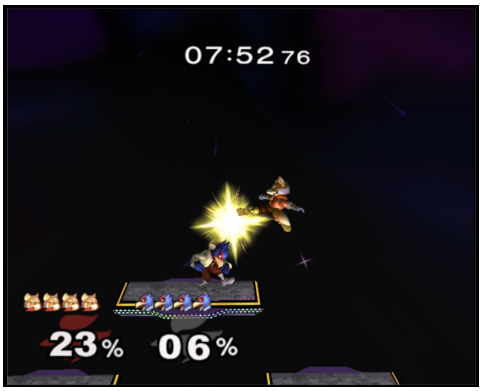
- **Opening Conversion Rate (OCR)**: Under BC, OCR is limited; IQL boosts conversion on punish opportunities but introduces greater inconsistency in out-of-distribution states, while PPO further improves OCR and likely reduces variance through its clipped surrogate updates.
- **Openings / Kill (O/K)**: BC requires the most openings per stock; IQL reduces this by making punish sequences more efficient, and PPO matches or slightly improves on IQL with more stable stock closures.
- **Damage / Opening (D/O)**: This metric stays consistent across BC, IQL, and PPO, indicating that neither offline nor on-policy RL learns higher-damage combos per opening. Overall damage gains arise from better conversion rather than stronger individual sequences, and more complex chains of tactics beyond those learned by imitation in BC are unfortunately not realized.

5.2.2 Briefly on Transformers

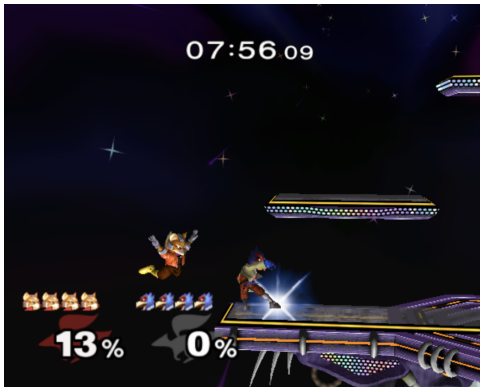
Given these metrics, one might ask: would a Transformer architecture not work much better? After all, the combo-driven nature of SSBM tactics lends itself well to the idea that a Transformer architecture that autoregressively predicts state would do much better than our currently MLP architecture. Indeed, that was our hypothesis; however, it was not what we found. Instead of performance that exceeded that of the MLP, we found action predictions that were closely tied to the mean controller action i.e. zero joystick movement and no buttons pressed. This unfortunately lead to null metrics but makes sense: our Transformer model is drastically underfitting due to the limited amount of data we are providing this architecture. Indeed, as existing reports have shown, scaling the amount of data and compute used to train a Transformer does result in impressive performance, but we simply could not compete on that dimension.

5.3 Qualitative Analysis: Exploring Game Play

Metrics are great, but what implications does this actually suggest in terms of SSBM in-game performance? We decided to probe deeper into the evaluations we ran in order to qualitatively understand the strategies we trained, in particular trying to understand the reason behind low number of kills. In order to do this, we watched every single match produced in the evaluation phase (30 total). What we found were behaviors that were surprising effective but unfortunately all too human.



(a) Fox as Player 1 opens the game by hitting an effective combo against Falco as Player 2, a replica of expert tac.



(b) Player 1 dives over Player 2 onto the left side of the stage, falling without attempting to grab onto the ledge as an expert player would.

Figure 3: Two key moments from our overall set of matches: an early combo and a fatal misstep.

5.3.1 Hitting Combos...

Figure 3a showcases our agents' core micro-strategic success: learning to string together fast follow-up attacks in combos that maximize damage and set up potential kill opportunities. In this example, Fox (Player 1) capitalizes on a small opening against Falco (Player 2) by chaining a short hop aerial forward tilt that then later goes into a series of neutral attacks. This sequence, which we saw in all three policies, demonstrates that both BC and RL-refined policies can capture temporal action dependencies from human replay data. The consistency of these learned combos correlates strongly with gains in OCR and the lack of change in D/O observed in Table 1.

Despite this success, our agents still miss or truncate combos often, especially under IQL and PPO where exploration introduces variability in the resulting performances. Failed combos have a critical failure mode: timing misalignment causing aerial or tilt attacks to miss. These failure patterns suggest that while our models learn the macro-structure of combos, they lack the precise calibration exhibited by expert human play to exploit advantages in position. Addressing these gaps may require augmenting the training data with targeted "combo drills" focused on specific spacings or player positions. As we see below, however, missing attacks is not the only issue.

5.3.2 ...and Jumping Off Stages

Despite the uncannily good micro-strategy of our agents, all agents across all our training methods (BC, IQL, PPO) failed to learn one critical heuristic: falling to your death unprompted is bad, so you should avoid actions that cause it and attempt to recover if you are falling. Figure 3b is a good example of this: Fox as Player 1 is not actively being attacked by the Player 2 (Falco), however, the agent decides to dash over onto the left side and proceeds to fall without making an attempt to grab onto the ledge and recover. Ultimately, this behavior is the single largest barrier to greater success of our agents. At the suggestion of our TA, we attempted curation of specific instances of recovery that we could use within training to improve the performance of our agent. However, a key barrier was that specifying the right bounding box for states involving recovery and then the right specific state/action sequences representing recovery was quite difficult. Even after curation, the overall metrics produced by such a model in BC were not significantly different from the baseline.

There are a few implications that we can draw via observing this behavior. Firstly, for actions where deviation from the correct sequence is highly consequential, we need much more data to learn from. While a few flukes here and there in the middle of the stage are fine and might be representative of noise inherent with the training data, recovery is what could be considered a "must learn" in our BC baseline. Furthermore, these results suggest that we could have built a more robust reward function that better captures the tradeoff of conservatism (trying not to lose a stock) with initiative (going to combos, even if they are risky, knowing that tempo is important to agent success). Of course, the difficulty in specifying such a reward function lends further credence to the idea that training a better reward predictor across a wider range of state/action inputs via a process like PPO is the correct way forward towards scaling performance.

6 Discussion

6.1 Better, More Accurate Evaluation

Accurately evaluating agents in a way that allows us to gauge its actual in-game performance is difficult. Even with an emulator in place and in-the-loop metrics such as percent damage, it is challenging to appropriately reward agent actions. For example, rewarding an agent based on player health retained (essentially penalizing it for losing health) could cause the agent to be adopt an overly defensive play style. Building careful in-the-loop metrics is critical to both understanding how agents perform in complex environments such as SSBM and pave the way forward to training better ones and would be a good line of future work.

6.2 Improving RL Evaluation

Training an agent for SSBM for reinforcement learning is bottlenecked by the latency of the emulator itself, waiting for actions to be taken, and observations and rewards to be returned to the agent. We might take a number of approaches to this. Firstly, we can decrease the latency of the emulator

itself through fast-forward mode, essentially allowing the game to run much faster than normal. Alternatively, we can also parallelize multiple instances of the emulator to increase the throughput of our system and train the agent in aggregate. Though these optimizations were not necessary for our limited amount of data and compute, they will be essential as we scale our model up and would be a critical future line of work. Such findings would also impact the development of techniques for evaluating other complex environments with difficult evaluation procedures like robot behavior learning.

6.3 Scaling is All You Need

More data would have enabled better performance. Although this is an almost intuitive statement in modern deep reinforcement learning, the point is worth repeating as we compare our performance with the CPU baseline and with available results. It is entirely reasonable to hypothesize based on these comparable that dramatically scaling the number of examples seen on tactics and recovery from expert players would have enabled us to train agents with richer and even winning behavior, but such is the constraint of class projects in modern ML.

7 Conclusion

In this work, we demonstrate the viability of training Super Smash Bros. Melee agents using a hybrid approach that combines behavior cloning on expert replay data with reinforcement learning finetuning via IQL and PPO. Despite limited compute and data compared to prior large-scale efforts, our agents learn competent micro-strategies and show modest performance gains with RL. However, persistent failures in recovery and stock preservation highlight the limits of our learning methods and underscore the need for better reward design, targeted data augmentation, and more scalable infrastructure. Our findings reinforce the promise of BC + RL pipelines for complex multi-agent games and point toward several future directions for achieving stronger, more human-like performance.

8 Team Contributions

All members were involved in discussion and implementation review of all facets of the work described in this paper. Specific emphases include:

- **Matthew Lee:** RL methodology implementation and training pipeline.
- **William Hu:** Emulator environment and behavior cloning implementation.
- **Samuel Do:** Data curation infrastructure and training infrastructure.

Changes from Proposal While essentially the same as the proposal in vision, we reduce the scope of performance we wanted to achieve after understanding key constraints in scale. The shifts in responsibility thus reflect the de-emphasis of the Transformer based agents.

References

AltF4 et al. 2025. *libmelee: Open API for Melee AI — Documentation*. Read the Docs. `https://libmelee.readthedocs.io/en/latest/` Version0.2.0 documentation, accessed 2025-06-09.

Vladfi1 et al. 2025. *slippi-Ishiiruka*: Dolphin Emulator fork with Slippi/AI support. `https://github.com/vladfi1/slippi-Ishiiruka`. GitHub repository, accessed 2025-06-09.

Vlad Firoiu, William F. Whitney, and Joshua B. Tenenbaum. 2017. Beating the World's Best at Super Smash Bros. with Deep Reinforcement Learning. *CoRR* abs/1702.06230 (2017). arXiv:1702.06230 `http://arxiv.org/abs/1702.06230`

Eric Gu. 2025. HAL: Training superhuman AI for Super Smash Bros. Melee. `https://github.com/ericyuegu/hal`. GitHub repository. Accessed: 2025-05-23.

Danijar Hafner, Timothy P. Lillicrap, Mohammad Norouzi, and Jimmy Ba. 2020. Mastering Atari with Discrete World Models. *CoRR* abs/2010.02193 (2020). arXiv:2010.02193 `https://arxiv.org/abs/2010.02193`

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. 2021. Offline Reinforcement Learning with Implicit Q-Learning. arXiv:2110.06169 [cs.LG] https://arxiv.org/abs/2110.06169

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR* abs/1712.01815 (2017). arXiv:1712.01815 http://arxiv.org/abs/1712.01815

A Behavior Cloning - Other Figures

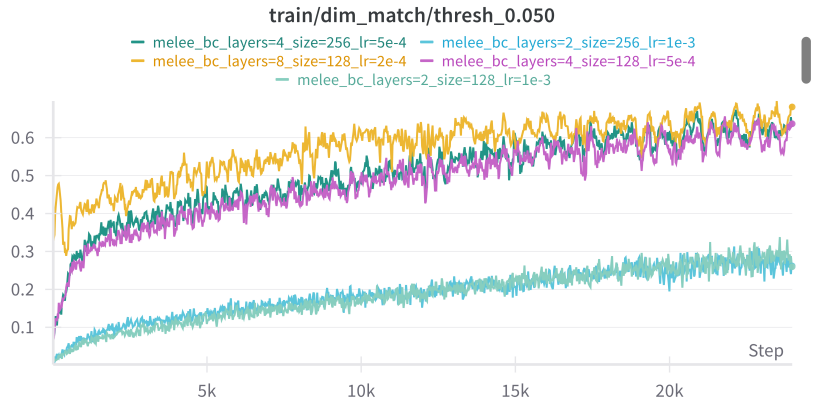


Figure 4: Percentage of action dimensions that match within threshold (0.05)

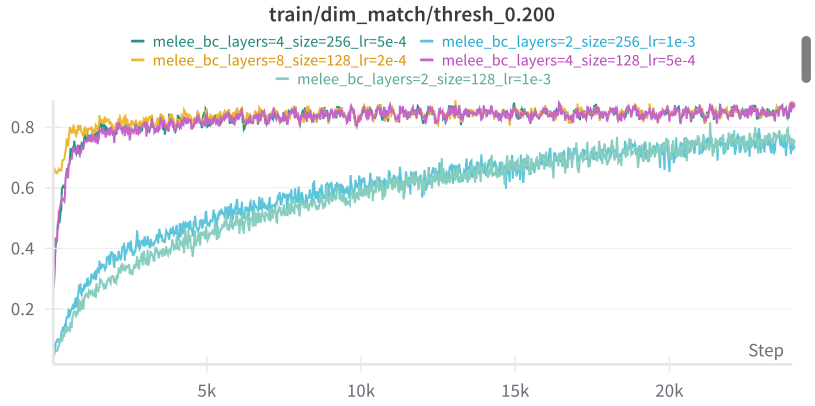


Figure 5: Percentage of action dimensions that match within threshold (0.2)