Universiteit Leiden

Master Computer Science

Transfer Learning on Character Matchups in Super Smash Bros. Melee: A Fox vs. Fox Case Study

Josef Hamelink

Name:	Josef Hamelink
Student ID:	s2233827
Date:	30/09/2025
Specialisation:	Artificial Intelligence
1st supervisor:	Aske Plaat
2nd supervisor:	Mike Preuss

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Transfer Learning on Character Matchups in Super Smash Bros. Melee: A Fox vs. Fox Case Study

Josef Hamelink
s2233827
j.d.hamelink@umail.leidenuniv.nl

September 30, 2025

Abstract

Many fighting games have a large roster of characters, each with their own unique moves and playstyles. With this project, we investigate the extent to which the matchup profile of training data affects the performance of *Imitation Learning* agents in Super Smash Bros. Melee (SSBM). Taking Fox (the face of competitive SSBM) as the main character for our case study, we train agents on three separate datasets that differ on the types of matchups they contain; *FvF* (Fox vs. Fox) only, *XvX* (non-Fox vs. non-Fox) only, along with a dataset containing replays where just one of the players is Fox (*XvF* and *FvX*). We use a GPT-like transformer architecture to train the agents to predict sets of controller inputs based on fixed-length sequences of game states. We also investigate the effects of finetuning the agents on a relatively small amount of *FvF* data after training on their respective designated dataset. After training, each agent plays a series of *FvF* games against Melee's most competent level CPU (level 9) to evaluate their performance. Not surprisingly, the Fox specialist outperforms the other agents in the *FvF* evaluation games; all other agents' winrates stay below 50%, even after finetuning. However, our (finetuned) true generalist (trained on *XvX*) does secure a respectable second place, largely due to our mixed agents showing remarkably little improvement gained from the *FvF* finetuning. These results suggest that while training on a diverse set of characters can yield decent performance, specializing in the goal matchup directly will still be the most efficient way to prepare an imitation learning agent for that specific scenario. We frame fine-tuning on Fox-vs-Fox as transfer learning, showing that specialization via transfer substantially improves closed-loop performance over purely generalist training.

Contents

1 Introduction

In recent years, the intersection of artificial intelligence (AI) and gaming has allowed researchers to explore complex environments and develop agents capable of learning and adapting to various challenges. Most notably, DeepMind's AlphaGo [1] and AlphaStar [2], and OpenAI's Dota 2 [3] have demonstrated the potential of AI in mastering games through Reinforcement Learning and self-play, reaching superhuman performance levels.

Over two decades ago – in 2001 – Nintendo released Super Smash Bros. Melee (SSBM) [4], a GameCube fighting game that despite Nintendo's lack of support has maintained a dedicated international player base and (competitive) community to this day [5]. The game's enduring popularity can be attributed to its fast-paced gameplay, intricate mechanics, and the high skill ceiling enjoyed by players and spectators alike. SSBM sports a diverse roster of characters, each with unique moves and playstyles, leading to a rich and complex metagame. In 2019, Jas Laferriere (a.k.a. *Fizzi*) started *Project Slippi* [6]; an open-source project that introduced rollback netcode and online multiplayer for emulators such as Dolphin [7]. This development not only allowed all sorts of players to seamlessly play against each other online from the comfort of their own personal computers, it introduced a replay system that records and stores detailed gameplay data in `.slp` files. These replays contain frame-by-frame information extracted directly from the game's memory, such as player positions, actions, and game states. This has opened up new avenues for AI research in SSBM, as these replays can be used to train and evaluate AI agents in a more efficient and robust manner when compared to previous methods that would have had to rely on screen capture and computer vision techniques to interpret the game state.

In this project, we train four different agents that use Imitation Learning (IL) with a GPT-like transformer architecture to predict controller inputs based on sequences of game states. These sequences are extracted (after some preprocessing) from three distinct `.slp` datasets. Each of these datasets contains the same number of replays, but they differ in the types of matchups they contain.

We have chosen Fox to be the subject of our study (see Figure 1). Since he is widely regarded as the best (and most popular), it was relatively "easy" to gather a large amount of fox mirror matches (*FvF*). Finding enough replays for any other mirror match – also referred to as a ditto – would have been much more computationally wasteful, since the character usage distribution in any fighting game is typically heavily skewed towards a select few strong characters [8]. Another benefit to Fox being the focus of our experiments, is that he finds himself a so-called *clone character* in Falco, whose moveset and and playstyle – while far from identical – are similar enough to Fox's that evaluating our agents on the Falco ditto can help us further understand the how well our agents generalize to unseen matchups.

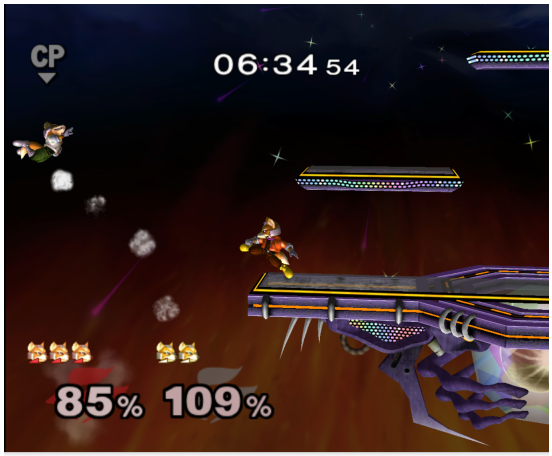


Figure 1: The Super Smash Bros. Melee environment. Shown here is one of our trained agents (in orange) KO'ing the best bot in a Fox vs. Fox evaluation game.

1.1 Research Questions

With this setup, we aim to answer the following research question:

To what extent is matchup generalization of training data advantageous for Imitation Learning agents in SSBM compared to specialization?

In order to answer this question, we want to first answer the following sub-questions:

1. *How does the specificity of IL training data affect validation loss on Fox mirror match data during training?*
2. *How do the different agents perform in live-environment Fox mirror match evaluations after training on their respective datasets?*
3. *Does transfer learning (finetuning) on Fox mirror match data after training on a different dataset significantly improve performance in Fox mirror match evaluations?*
4. *How well do differently trained agents generalize to other evaluation matchups, and does transfer learning still help in this regard?*

Answering these questions will provide insights into the trade-offs between specialization and generalization in training data for IL agents in SSBM, and potentially inform future research and development of AI agents in other fighting games.

To briefly summarize our contributions: we curate and preprocess three distinct datasets of SSBM replays, stratified by matchup type, enabling efficient and targeted imitation learning experiments. We then implement and train transformer-based agents to predict controller inputs from game state sequences, after which we evaluate their performance in emulated Fox mirror matches against SSBM's highest-level CPU.

Together, these contributions form the first systematic investigation into the effects of matchup-specific training data on imitation learning agents in SSBM, providing insights that could inform future AI research in fighting games and beyond.

2 Related Work

Even though (1v1) fighting games theoretically provide a rich testbed for AI research, they have not received nearly as much attention as other genres such as board games [1], real-time strategy games [2], and multiplayer online battle arena games [3]. This is likely due to the inherent complexity of fighting games, which often feature a large roster of characters, each with their own unique moves and playstyles. For instance, SSBM features 26 [1] playable characters, leading to a total of 676 possible matchups (including mirror matches).

2.1 Reinforcement Learning

Other fighting games that have been the subject of AI research include *FightingICE* [9,10], *Blade & Soul* [11], and recently superhuman performance has been achieved in *Street Fighter* [12]. Most, if not all, of these works make use of Reinforcement Learning (RL) techniques, often through self-play, to train their agents. Scientific literature on SSBM is no exception; RL has been the approach chosen by all relevant works we could find on melee AI. Most notably, Vlad Firoiu et al. [13] have been able to train Q-learning and Actor-Critic agents that have been able to beat some of the highest-ranked melee players in the world.

2.2 Imitation Learning

We position our finetuning step as a form of transfer learning, where we take an agent trained on a source domain (e.g., non-Fox matchups) and adapt it to a target domain (Fox mirror matches). Transfer learning – especially in the context of deep learning – has been widely studied in the literature, with many works showing that pretraining on a large and diverse dataset can lead to improved performance on a smaller, more specific dataset [14].
In natural language processing, transformer-based models such as BERT [15] and GPT [16] have demonstrated the effectiveness of transfer learning through pretraining on large corpora followed by finetuning on specific tasks. Specifically GPT-3 [16] has shown that large-scale pretraining can lead to impressive zero-shot and few-shot learning capabilities across a wide range of tasks. Transfer learning across different games has also been explored, most notably by DeepMind in their work on AlphaZero [17], where they demonstrated that an agent trained on chess could be adapted to play shogi and Go with minimal additional training. A prerequisite for successful transfer learning is that the source and target domains share some underlying structure or features, which in the case of different fighting game characters is exactly the case, making it a promising avenue to explore.
Outside of academic context, a machine learning researcher by the name of *Eric Gu* has leveraged a transformer to do supervised learning on a large dataset of SSBM replays. His work is well-documented in a blog post [18] and a public GitHub repository [19]. Inspired by Andrej Karpathy's *nanoGPT* [20], Eric used next-token prediction on a decoder-only transformer to predict, for a sequence of game states with length N, the controller inputs for the next frame $N + 1$, simultaneously for all frames. After training on 3 billion frames of data, his agent was able to beat the level 9 CPU in a Fox mirror match with a winrate of around 95%.
His contributions have been of great help to our own project, serving as a solid foundation to build upon. I.e., we adapted his game state representation and model architecture (including most of the hyperparameters), allowing us to focus on the effects of training data specificity.

[1] While only 25 are available in the selection screen, one of the characters (Zelda) can switch back and forth to another character with an entirely different move set (Sheik) throughout a match.

3 Methodology

In this chapter, we describe our experimental setup used to investigate the research questions posed in Chapter 1.1. We first outline the datasets we curated to train our agents on (Chapter 3.1), along with how the data is pre- and post-processed (Chapter 3.2). Next, we describe the architecture of our model (Chapter 3.3), as well as the training procedure (Chapter 3.4). To conclude this chapter, we briefly describe in what manner we evaluate our trained agents (Chapter 3.5). A brief overview of the entire pipeline is sketched in Figure 2.

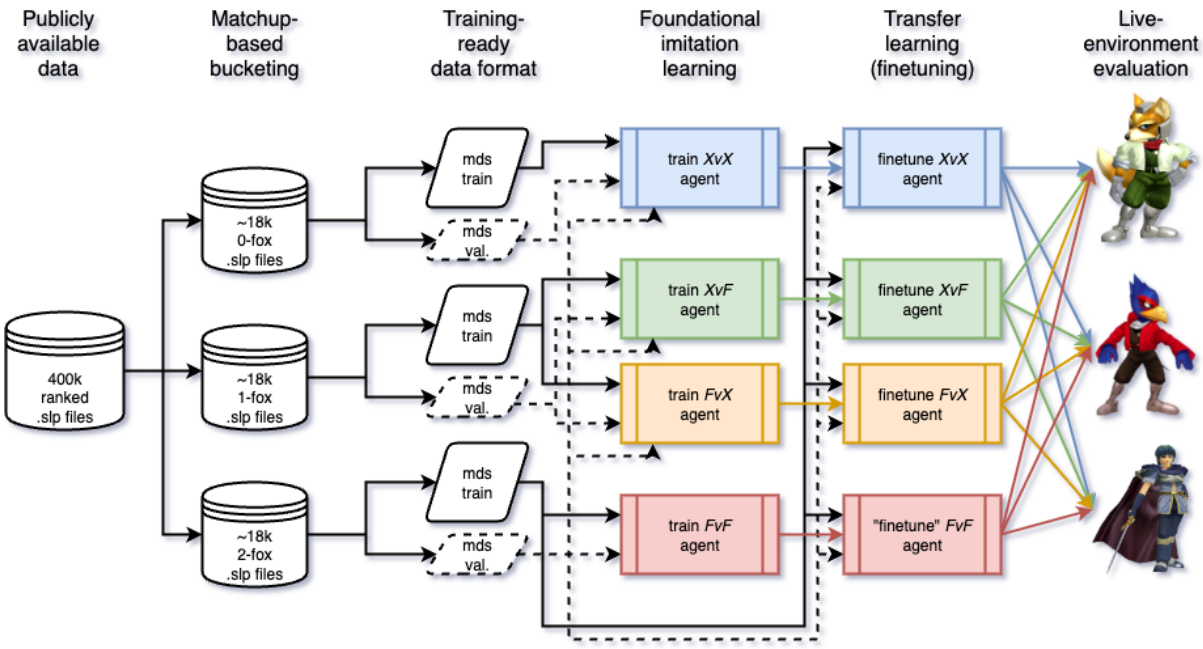


Figure 2: Overview of the complete pipeline built for this thesis.

3.1 Dataset Curation

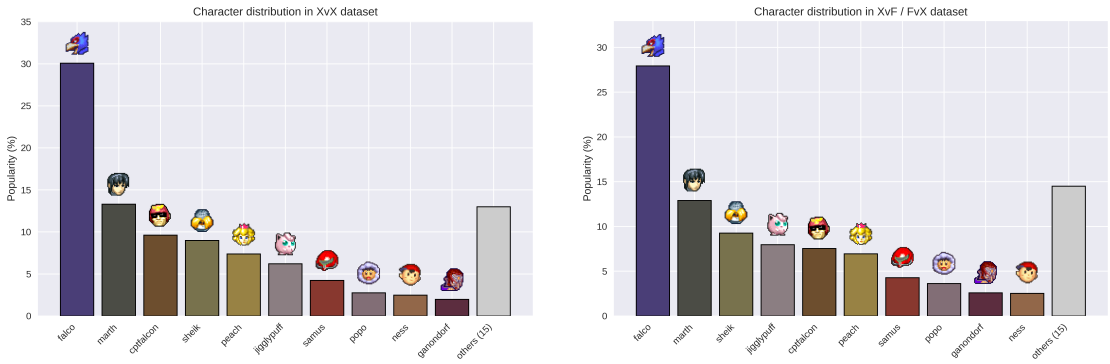
Thousands of games are played online every day, thanks to the rollback netcode introduced by Project Slippi [6]. This year, Vlad Firoiu has published a vast collection (well over half a million) of these melee replays [21]. These games are sourced from the three highest ranks in online SSBM (platinum, diamond, and master), suggesting they are of at least decent quality, and follow a competitive metagame in which Fox is frequently used.

Due to hardware limitations further down the line, we chose to limit the size of our datasets to $18\,432$ replays each. $16\,384$ of these replays were used for training, and 2048 for validation, resulting in a train/val split of approximately 89/11. These numbers are deliberately chosen to be powers of two, as this allows for efficient batching during training. With finding a sufficient amount of Fox dittos being the limiting factor, we still needed to download and parse roughly $400\,000$ `.slp` files to get to our desired dataset sizes, essentially discarding the vast majority of the replays.

To reiterate, we end up with three buckets of data, differing only in the number of players playing as Fox:

- **FvF**: both players are Fox (pure Fox specialist)
- **XvX**: neither player is Fox (true generalist)
- **XvF/FvX**: exactly one player is Fox (mixed agents)

Note how – by shifting the perspective during training – we can use the same dataset for training both *XvF* and *FvX*.
It is imperative to the quality of our experiments that our datasets are balanced in all other relevant aspects, such as the stages on which the games are played, the distribution of other characters where applicable, and the win rates of the players we set our agent's perspective to. In Figure 3, we can see that the character distributions in our datasets are roughly the same; Falco (Fox's clone) dominating the non-Fox character usage, followed by Marth, and some of the other stronger characters. The top 10 most popular characters are also the exact same across both datasets.



(a) (b)

Figure 3: Character distributions in the datasets where Fox is not present (a), and where exactly one player is Fox (b). For the former, we consider both players' characters, while for the latter we only show the characters piloted by the non-Fox player.

In SSBM, there are 29 different stages (in other games also referred to as maps or arenas) that can be played on, each with their own unique layout and features. Being originally designed for 4-player free-for-all matches, most of these stages are not suitable for 1v1 competitive play. Therefore, the competitive community has agreed upon a small selection of 6 stages that are deemed tournament-legal. Slippi online matches are also restricted to these 6 stages, which is reflected in our datasets. Figure 4 shows the stage distributions across all three datasets, which are again roughly the same. One small quirk worth noting is that we see a slightly higher usage of *Yoshi's Story* (YS) and *Pokemon Stadium* (PS) in the *FvF* dataset, at the cost of a proportionally lower usage of *Fountain of Dreams* and *Final Destination*. One can imagine that some characters inherently benefit more from certain stage features than others; in practice we see Fox players often electing to "counterpick" their opponents to YS or PS. In the ditto matchup, any advantage gained from these stages is nullified, yet Fox players still seem to feel more comfortable on these stages.

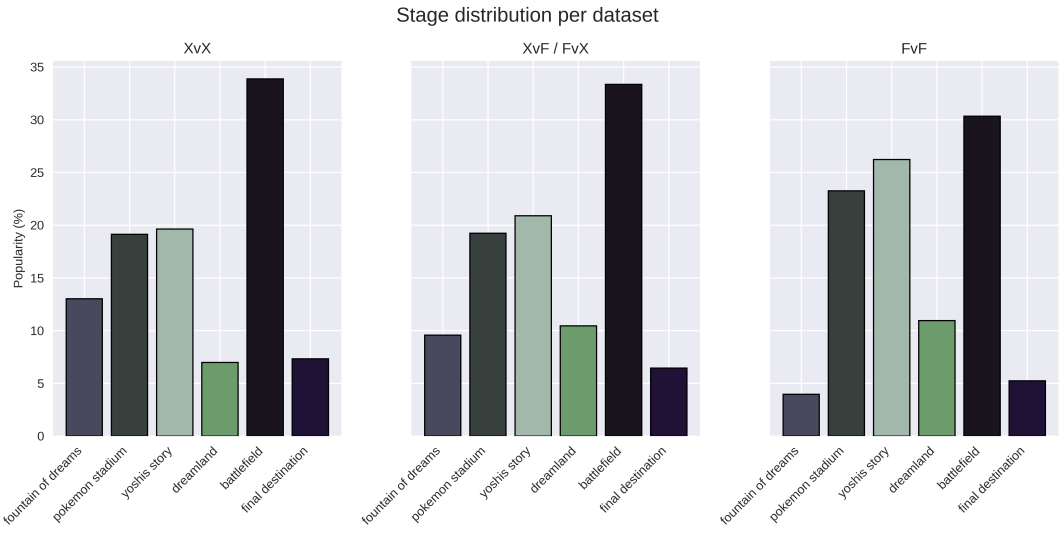


Figure 4: Stage distributions across all three datasets.

As for winrates, we find that our arbitrarily picked player (p1) enjoys win rates of 55.01% and 52.18% in the *FvF* and *XvX* datasets respectively, which is acceptable, but not ideal. More importantly, our mixed dataset is more balanced with a win rate of 50.75% for the Fox player. A bias in this dataset could have seriously favored the *FvX* agent at the expense of the *XvF* agent, or vice versa.

Now that we have three sets of balanced `.slp` files, we need to preprocess them into a format suitable for training our agents. With some slight modifications, we use the same preprocessing pipeline as Eric Gu [19]. For every replay, we extract frame-by-frame game state information, such as player positions, velocities, controller inputs, and various other features. In total, each frame is represented by a vector of 91 features; consisting of 29 game state plus 15 controller features, for each of the two players. The remaining three entries are player-independent metadata; frame index, stage, replay uuid.

The controller inputs are represented as a combination of discrete and continuous values. The GameCube controller (see Figure 5) has two analog sticks; the main control stick which is primarily used for movement, and the C-stick which can be used for attacks. In our representation, both sticks are represented by two continuous values each, in the range $[0, 1]$. Our model's stick predictions will be discretized due to some rather intricate SSBM mechanics that I will not go into here, but are thoroughly explained in Eric Gu's blog post [18]. For our purposes, it suffices to say that 21 discrete (x, y) tuples are used to cover the most relevant stick positions. The shoulder buttons (L and R) are pressure-sensitive, allowing for "light-shielding" which is a relevant technique in high-level play, and is therefore also relevant to model. In a similar fashion to the analog sticks, model predictions regarding shoulder buttons are discretized into three levels; not pressed, light-pressed, and fully pressed. All other buttons are digital, i.e. either pressed or not. It is worth noting that multiple buttons can be pressed simultaneously, and that some buttons have different functions depending on the context. The extracted data is stored in a series of compressed *MosaicML Data Shard* (MDS) files [23], which can be efficiently streamed during training.

Figure 5: The GameCube controller layout, along with the in-game actions each button/stick translates to, courtesy of IGN [22].

3.2 Processing

The processing pipeline serves as the bridge between raw replay features stored in our curated MDS files, and the structured input/target pairs required for training our transformer model. Rather than transforming the entire dataset upfront, we employ an on-the-fly pre- and post-processing pipeline that interacts directly with the model's data loader. This design allows us to flexibly sample fixed-length sequences from full episodes while applying the same transformations consistently during both training and evaluation.

Each frame of gameplay is represented as a `TensorDict` containing both player-specific and global features. Continuous values such as player positions, velocities, and damage percent are normalized or standardized to ensure stable training dynamics, while shield values are inverted and normalized to semantically align with damage percent. As stated before, some controller inputs (the analog sticks and shoulder buttons) are discretized into categorical bins, while others stay as binary values.

Crucially, inputs and targets are temporally offset: for every game state at frame t, our model is tasked with predicting the controller inputs at frame $t + 1$. This offset is enforced during preprocessing, ensuring that the network learns causal mappings from state to action rather than simply memorizing concurrent values.

3.3 Model Architecture

Our agents are parameterized by a GPT-like transformer that maps fixed-length sequences of processed game states to predictions of the next-frame controller inputs. The input layer concatenates continuous features with learned embeddings for categorical variables such as stage, character, and current action, projecting them into a common embedding space. The core of the model consists of a stack of transformer blocks with causal self-attention and relative positional encodings, which preserve temporal order while enabling the network to capture dependencies over long horizons. The shared transformer backbone outputs a contextualized representation for each frame, which is then decoded through four task-specific heads corresponding to the controller components: C-stick, main stick, button presses, and shoulder triggers. These heads are connected in a chained fashion, where each head conditions not only on the transformer representation but also on the detached outputs of previous heads, reflecting the natural hierarchy of controller decisions. This design ensures that interdependencies between controller modalities (e.g., a button press combined with a stick angle) can be modeled without introducing degenerate feedback loops. A schematic overview of how a single frame would be processed through the model is shown in Figure 6.

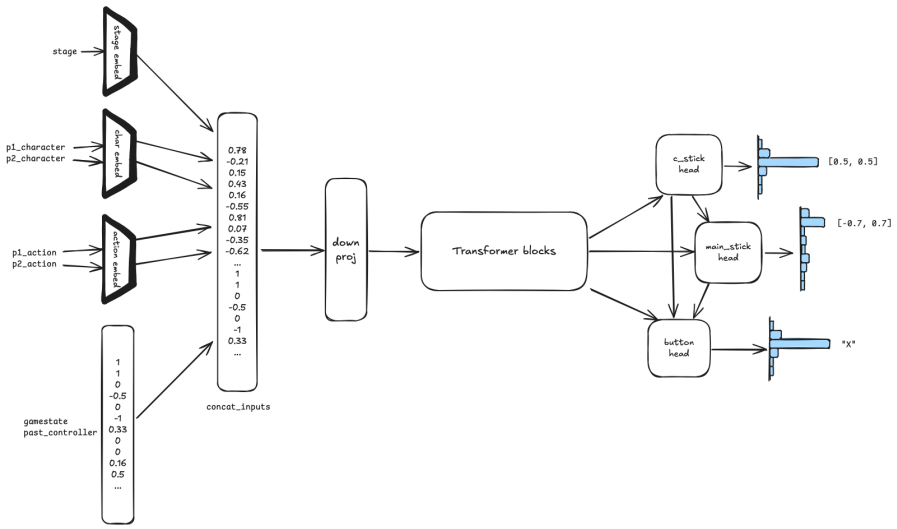


Figure 6: Overview of the model architecture we used, courtesy of Eric Gu [18].

3.4 Training Procedure

Training is carried out using the AdamW optimizer with a cosine annealing learning rate schedule. Each model is trained for a fixed number of epochs on its designated dataset, using batches of subsequences that are sampled and preprocessed on the fly by the streaming dataloader. Within each training step, the transformer produces predictions for all four controller heads, which are compared to the observed next-frame inputs using cross-entropy loss. These losses are summed to obtain the total training loss, ensuring that all aspects of the controller are optimized jointly.

Validation is performed after every epoch on two sets: the validation split of the same matchup type as the training data, and a held-out set of Fox ditto games. This dual validation setup allows us to monitor not only in-domain performance but also how well each agent is tracking the goal evaluation scenario. The best-performing model with the lowest own-type validation loss is saved to disk for later evaluation.

After the initial training run, we perform an additional finetuning phase on the Fox vs. Fox dataset. In this phase, the optimizer is reinitialized with a reduced learning rate and trained for a smaller number of epochs, again with cosine annealing scheduling. To make things as fair as possible, our specialist agent (trained on *FvF*) also undergoes this finetuning step, effectively serving as a control to isolate the effects of the additional training. The idea here is to see how well agents trained on other matchup types can adapt to the Fox ditto scenario with a limited amount of specialized data. Finetuned models are saved separately, allowing us to directly compare agents before and after this adaptation step.

3.5 Evaluation

To evaluate our trained agents, we pit them against the highest-level CPU opponent available in SSBM (level 9). This CPU is known to be quite competent, and has been used as a benchmark in previous works [13, 18]. For each of the best models (before and after finetuning), we run a series of 100 Fox mirror matches against the level 9 CPU, saving the resulting `.slp` replays for further analysis. On top of the Fox mirrors, we are also interested in evaluating our agents on other matchups, such as Falco or Marth mirrors; two other very popular characters in competitive SSBM and our datasets (see Figure 3). Any other evaluation matchup could be easily ran, but would be less relevant to our research questions.

During these evaluation matches, the agent receives the same game state features as during training, and outputs controller inputs at each frame, which are then fed into the emulator to control the Fox character. The CPU opponent operates independently, following its own built-in AI logic. We were unable to find any publicly available implementations of the level 9 CPU AI, but it is widely believed to use some form of hardcoded decision tree logic – rather than machine learning – given the era in which the game was developed.

4 Experiments & Results

Our experimental setup consists of training four different agents, as outlined in Chapter 3.1 and 3.3, each on their respective dataset. The hyperparameters used for training are mostly the same as those used by Eric Gu [18, 19], with some slight modifications to account for our smaller datasets and hardware limitations.
We ended up using 4 layers, 8 attention heads, and an embedding dimension of 256 for our transformer. The AdamW optimizer is initialized with a learning rate of 3×10^{-4}, weight decay of 1×10^{-2}, and β_1, β_2 values of 0.9 and 0.999 respectively. For finetuning, we reduce the initial learning rate to 1.5×10^{-5}, with a similar weight decay and β_1, β_2 values. The course of the learning rate during training and finetuning is visualized in Figure 7.

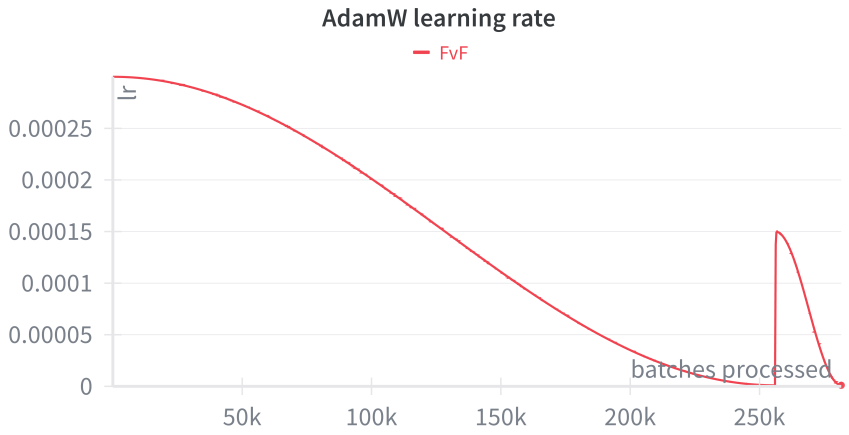


Figure 7: Learning rate schedule over the course of training (1000 epochs) and finetuning (100 epochs).

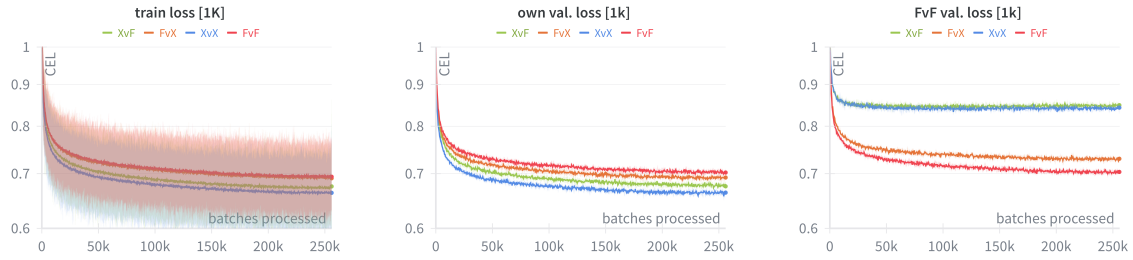
One epoch of training consists of one pass through all $16\,384$ replays in the training set, sampling one random sequence of 256 frames (approximately 4.27 seconds of gameplay) from each replay. We use a batch size of 64, resulting in 256 training steps per epoch. Every agent is trained for a total of 1000 epochs, after which we finetune them for an additional 100 epochs on the *FvF* dataset. Three repetitions were sufficient to verify the consistency of our results, so the learning curves shown in Chapter 4.1 are averaged over these three runs. The burden of training these models was shared between a local machine (RTX 3080Ti) and multiple nodes on the Leiden ALICE cluster (RTX 2080Ti). Each run took approximately 12 hours to complete. This is not too bad, considering we process well over four Billion individual frames [2] in total, not even counting validation.

[2]We train on $4\,613\,734\,400$ frames, to be precise. 256 frames per episode, on $16\,384$ replays, for a total of $4\,194\,304$ frames, over 1100 epochs (training + finetuning).

4.1 Learning Curves

The first thing we want to verify is that the validation loss on an agent's own type of data follows the trend of the training loss, which is a prerequisite for any meaningful training. In Figure 8 we can see that this is indeed the case for all agents, with the agent's own-type validation loss closely tracking the training loss throughout the entire training process, while being less noisy.

We can also see that judging by train loss and own-type validation loss, the *XvX* agent has the easiest time fitting its training data, followed by the agent who faces Fox and plays as a non-Fox character (*XvF*). *FvX* comes in at third place; performing on par with *FvF* on train loss, but slightly better on own-type validation loss. Counterintuitively, the pure specialist (*FvF*) seems to struggle the most achieving low validation loss on its own type of data. One explanation for this could be that – since it has the least diverse training data – it is more prone to overfitting, and therefore has a harder time generalizing to unseen validation replays.

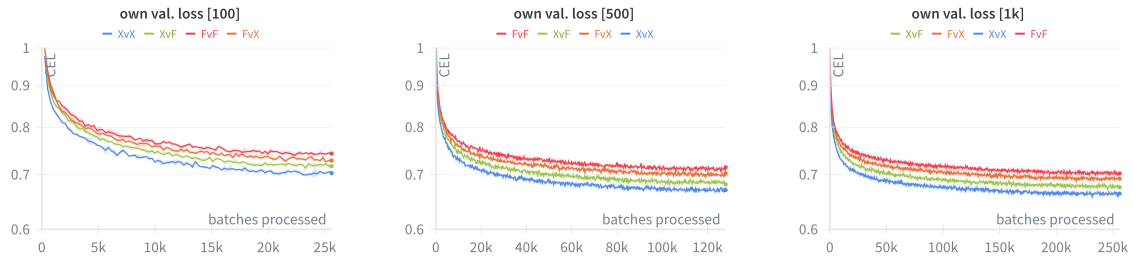


(a) (b) (c)

Figure 8: Training and validation losses over the course of the 1000 training epochs for all four agents. Note that the y-axis shows the total CE loss (summed over all four controller heads) on a log scale.

When we switch our attention to the Fox ditto validation loss (the goal evaluation scenario), we see a different picture emerging. In this scenario, it should come as no surprise that the *FvF* agent performs best, with the lowest validation loss throughout training. The most striking observation here is that our mixed agents seem to resemble their "ego" counterparts more than they do each other; while the learning curves for *XvF* and *XvX* are nearly indistinguishable (and plateau at a high, unfavorable loss), *FvX* seems to be doing significantly better, almost being on par with *FvF*. This suggests that the character an agent plays as – or, more accurately, is tasked to imitate – has a much larger impact on its prediction accuracy, than the character it faces in the matchup. Intuitively this makes sense, as the agent's own character determines the set of possible actions it can take, and therefore has a larger influence on the controller input distribution it needs to learn. The final point of interest in these learning curves is the asymmetry: while the *FvX* agent performs worse than the *FvF* specialist despite being exposed to more varied opponents, the *XvF* agent gains no measurable advantage over the *XvX* agent from having trained against Fox in its training data.

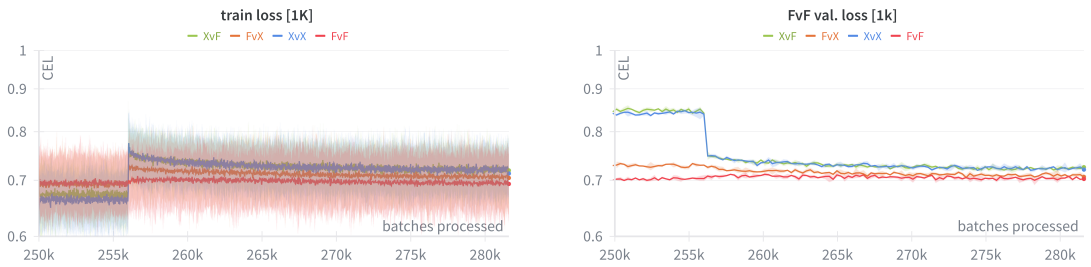
We also carried out similar training runs with a shorter training period of 500 epochs (plus 50 finetuning), and 100 epochs (with 10 finetuning), to see how the learning curves would compare. The results for own-type validation loss are shown in Figure 9. While the exact loss values differ, the overall trends remain the same, with *XvX* and *XvF* performing best on their own-type validation data, and *FvF* lagging behind the most. We can also see that the jump from 500 to 1000 epochs is not as significant as the jump from 100 to 500 epochs, indicating a strong case of diminishing returns.



(a) (b) (c)

Figure 9: Own-type validation losses for all four agents, trained for 100 (a), 500 (b), and 1000 (c) epochs. Note that the y-axis shows the total CE loss (summed over all four controller heads) on a log scale.

After the initial training phase, we finetune all agents on the *FvF* dataset for an additional 100 epochs. We stop tracking the own-type validation loss, as it is no longer relevant, and focus solely on the Fox ditto validation loss. The learning curves for this finetuning phase are shown in Figure 10.



(a) (b)

Figure 10: Training and validation losses over the course of the 100 finetuning epochs for all four agents. Note that the y-axis shows the total CE loss (summed over all four controller heads) on a log scale.

When the finetuning phase starts (at 256k batches processed), we see that – especially for our "*Xv?*" agents – the train loss suddenly spikes up, before slowly decreasing again. This is to be expected, as the model is now exposed to a substantial shift in training data, and needs to adapt its parameters accordingly. While the *FvF* agent is still training on literally the same data, it does sees a small increase in train loss. This phenomenon can be attributed in full to the sudden increase in learning rate (see Figure 7). The validation loss for the *FvF* agent remains mostly unchanged.
The most interesting observations here are the improvements made by the other three agents. The *FvX* agent, which was already performing decently on the Fox ditto validation set, sees a small but consistent improvement throughout the finetuning phase, ending up even closer to the *FvF* agent. *XvF* and *XvX* on the other hand, enjoy much more notable improvements, staying closely together, closing over three-quarters of the gap to the *FvF* agent, mostly in the validation round after the very first finetuning epoch.
Similarly to what we show in Figure 9, we also carried out finetuning runs after training for only 500 epochs (plus 50 finetuning), and 100 epochs (with 10 finetuning). The results for Fox ditto validation loss during finetuning are shown in Figure 11.

(a) (b) (c)

Figure 11: Own-type validation losses for all four agents, trained for 100 (a), 500 (b), and 1000 (c) epochs. Note that the y-axis shows the total CE loss (summed over all four controller heads) on a log scale.

The same trends described above are visible in these shorter finetuning runs as well, with the *Xv?* agents plateauing at a high *FvF* loss before they are exposed to the finetuning data, and then making significant improvements in the first few epochs of finetuning. Since the agents trained on a thousand epochs do show the most promising results, evaluation is only carried out on these models.

4.2 Evaluation Results

As mentioned in Chapter 3.5, we evaluate our trained agents by having them play a series of 100 matches against the level 9 CPU in an emulated SSBM environment. For each of the four agents, we evaluate both the best model after the initial training phase, as well as the best model after finetuning, resulting in a total of eight evaluation runs. The best model is determined by the lowest own-type validation loss for the initial training phase, and the lowest Fox ditto validation loss for the finetuned models. For the evaluation matches, we obviously run the Fox mirror matchup, but also the Falco and Marth mirror matchups, to see how well the agents generalize to other characters. It is important to note that a game in Super Smash Bros. Melee is won by the player who takes all opponent's stocks (lives) first, and that each player starts with four stocks. This allows us to use two metrics to evaluate our agents; win rate (the percentage of games won out of 100), and kill/death ratio (the number of opponent stocks taken divided by the number of own stocks lost). The results of these evaluation runs are depicted in Figure 12, and the raw numbers are also listed in Table 1.

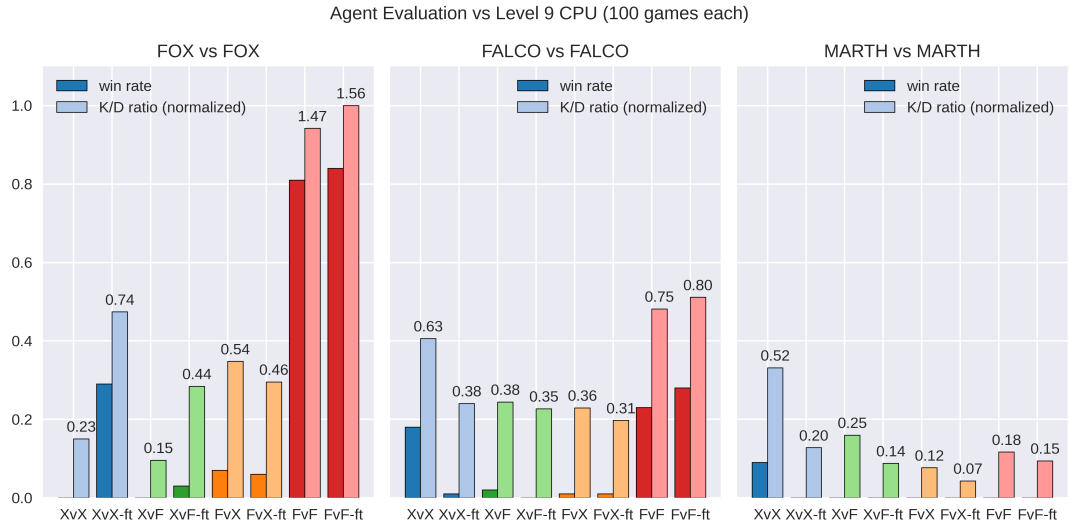


Figure 12: Win rates and k/d-ratios of all four agents, before and after finetuning, against the level 9 CPU in Fox, Falco, and Marth mirror matches. K/D-ratios are defined as the number of kills (opponent stocks taken) divided by the number of deaths (own stocks lost).

There is a lot to unpack in these results, so let's start with the matchup most relevant to our research questions; the Fox mirror matches. The agent trained on *FvF* data dominates on its home turf, achieving a win rate of 81% before "finetuning" (which in this case is just further training on the same data), and 83% after. Looking at the secondary metric (k/d ratio), the same trend is visible, with the *FvF* agents averaging three stocks taken for every two stocks lost.

The runner-up in this scenario is the finetuned *XvX* agent, still managing to win almost a third of its matches against the level 9 CPU, and achieving a quite decent k/d ratio of 0.74. This is a major improvement over its pre-finetuning performance, where it was unable to take even a single win over the CPU, and taking on average less than a single stock per game (it loses all its own four stocks per match).

Looking at the mixed agents, we do see that the agent who has been trained to play *as* Fox (*FvX*) was able to win a handful of matches even before finetuning, while the *XvF* agent was not. Quite counterintuitively however, the finetuned *FvX* agent performs worse on both metrics than its pre-finetuning self, which is something we have a hard time coming up with a convincing explanation for. The finetuned *XvF* agent on the other hand, does go from zero to three wins, and almost triples its k/d ratio, which is more in line with our expectations.
Moving on to the Falco vs. Falco evaluation matches, we would expect our three non-specialist agents to perform better here, as they have all been exposed to Falco gameplay in their training data, while the specialist has not. While they still lag behind the *FvF* agent, the gap is less pronounces than in the Fox mirrors. The fact that the *FvF* agent is still the best here, despite not having seen a single Falco game during training, is quite remarkable, and speaks to the similarity these two "clone" characters share. This time, coming in at a closer second place is the non-finetuned *XvX* agent, with a win rate of 18% and k/d ratio of 0.63. The finetuned *XvX* agent on the other hand, performs quite poorly, winning only a single match, as well as showing a drop in k/d ratio. The mixed agents again perform the worst, showing stats very similar to the finetuned *XvX* agent.
Finally, the Marth vs. Marth evaluation matches feature low scores across the board, with the only agent being able to even win a single match being the non-finetuned *XvX* one. *FvF*'s poor performance here is not too surprising, given that – unlike Falco – Marth is a completely different character from Fox; being more reliant on spacing and fine-grained movement rather than raw speed and pressure. It speaks to the versatility of the *XvX* agent that it is able to eke out a few wins here, despite not having seen a single Fox game during training. One could speculate that, when given the appropriate finetuning data, this agent could potentially adapt to the Marth matchup as well as it did to the Fox matchup, but this is something we did not have the time to investigate.

Agent	**FOX**		**FALCO**		**MARTH**	
	winrate	k/d	winrate	k/d	winrate	k/d
XvX	0.00	0.23	**0.18**	**0.63**	**0.09**	**0.52**
XvX-ft	**0.29**	**0.74**	0.01	0.38	0.00	0.20
XvF	0.00	0.15	0.02	0.38	0.00	0.25
XvF-ft	0.03	0.44	0.00	0.35	0.00	0.14
FvX	0.07	0.54	0.01	0.36	0.00	0.12
FvX-ft	0.06	0.46	0.01	0.31	0.00	0.07
FvF	**0.81**	**1.47**	**0.23**	**0.75**	0.00	0.18
FvF-ft	**0.84**	**1.56**	**0.28**	**0.80**	0.00	0.15

Table 1: Win rates and k/d-ratios of all four agents, before and after finetuning, against the level 9 CPU in Fox, Falco, and Marth mirror matches.

5 Conclusions

Circling back to our research questions posed in Chapter 1.1, we can now provide some answers based on the results presented above.

1. *How does the specificity of IL training data affect validation loss on Fox mirror match data during training?*
   We find that the agent trained specifically on Fox mirror matches (*FvF*) unsurprisingly achieves the lowest validation loss on Fox mirror data during training. More interestingly, the agent trained to play as Fox against a variety of opponents (*FvX*) performs significantly better on this validation set than the agents trained to play as non-Fox characters, regardless of whether they faced Fox in their training data or not (*XvF* and *XvX*). This suggests that the character an agent is trained to imitate has a larger impact on its performance in a specific matchup than the character it faces.

2. *How do the different agents perform in live-environment Fox mirror match evaluations after training on their respective datasets?*
   In line with the validation loss results, the *FvF* agent dominates in Fox mirror match evaluations against the level 9 CPU, achieving a win rate of 81% and a k/d ratio of 1.47. The mixed agent trained to play as Fox (*FvX*) manages to win a few matches, but its performance is significantly lower than that of the specialist. The agents trained to play as non-Fox characters (*XvF* and *XvX*) struggle the most, both failing to win any matches before finetuning.

3. *Does finetuning on Fox mirror match data after training on a different dataset significantly improve performance in Fox mirror match evaluations?*
   Finetuning on Fox mirror match data leads to substantial improvements for the agents initially trained on other datasets. Especially the true generalist (*XvX*) sees a remarkable jump in performance, going from zero wins to a 29% win rate and a k/d ratio of 0.74 after finetuning. It is important to note however, that in some scenarios (as observed with *FvX*), finetuning can also lead to a decrease in performance, indicating that the relationship between initial training data and finetuning effectiveness is complex.

4. *How well do differently trained agents generalize to other evaluation matchups, and does finetuning still help in this regard?*
   When evaluating on the Falco mirror matchup, the *FvF* agent still performs best, despite not having seen this character during training. This highlights the versatility of the specialist agent, likely due to the similarities between Fox and Falco. When it comes to a character more distinct from Fox, such as Marth, all agents struggle, with only the *XvX* agent managing to win a few matches. The other agents, particularly the *XvX* agent, show some ability to generalize, winning a few matches in these scenarios. However, finetuning does not consistently improve performance in these other matchups, and in some cases, it even leads to a decline.

With these sub-questions answered, we can now address our main research question:

To what extent is matchup generalization of training data advantageous for Imitation Learning agents in SSBM compared to specialization?

Our findings suggest that while generalization can provide some benefits, specialization will without a doubt lead to better performance in the specific matchup the agent is trained for. Agents trained on diverse datasets can adapt to specific scenarios through finetuning, but this process is not guaranteed to yield improvements and may sometimes be detrimental. Therefore, the choice between specialization and generalization should be made based on the intended application of the agent, with a clear understanding of the trade-offs involved. The extent to which these findings generalize to other matchups, or even games remains an open question for future research.

6 Discussion

Our experiments confirm that specialization yields the strongest results in Fox ditto evaluations, while generalist agents benefit substantially from finetuning but rarely surpass the specialist. This aligns with intuition: when the evaluation scenario is narrow and well-defined, training directly on that distribution is the most effective strategy. However, our results also highlight that generalist agents are not without merit. The XvX agent, in particular, demonstrated a surprising capacity to adapt during finetuning, despite starting from a weaker baseline. This suggests that broader pretraining can provide a foundation for flexibility, even if it does not translate into immediate performance advantages when evaluating on a specific task.

At the same time, there are clear limitations that temper our conclusions. One major factor was hardware availability. Training on over four billion frames of data pushed the limits of both my personal workstation and the ALICE cluster, forcing compromises on dataset size, batch scheduling, and the number of repetitions. These constraints likely limited the stability of our results and prevented exploration of larger models or longer training runs that could have uncovered additional dynamics.

Another key bottleneck was the emulator itself. While Project Slippi and Dolphin provided a unique opportunity to interact with SSBM at frame-level precision, they also introduced practical hurdles. Running thousands of evaluation games through an emulator was computationally intensive and occasionally unstable, leading to delays and the need for repeated runs. Furthermore, the lack of transparency in the CPU's built-in AI means we cannot be entirely sure what behaviors our agents were exploiting—or failing to exploit—during evaluation. This makes the CPU a useful but still somewhat of a black-box benchmark for agent performance.

Taken together, these challenges underline the difficulty of bringing modern imitation learning methods to a twenty-year-old console game. Yet they also demonstrate the feasibility of the approach: despite the bottlenecks, we were able to curate balanced datasets, implement a transformer-based training pipeline, and conduct systematic experiments. The broader lesson is that even under non-ideal hardware and software conditions, carefully controlled experiments can still yield meaningful insights into the trade-offs between specialization and generalization.

6.1 Future Work

Apart from the general limitations discussed above, there are several specific avenues for future work that could build on our findings. First and foremost, expanding the work to include more characters and matchups would help validate whether the trends we observed with Fox generalize to other scenarios. Perhaps a study could be done on comparing the 10 most popular characters in competitive SSBM, training a true generalist with finetuning, alongside specialists for each character. This would provide a more comprehensive picture of how matchup diversity impacts imitation learning across the board.

Secondly – and perhaps more ambitiously – integrating reinforcement learning into the training pipeline could yield stronger agents. While imitation learning provides a solid foundation, it is inherently limited by the quality and diversity of the demonstration data. Combining IL with RL fine-tuning, as done in other domains, could allow agents to surpass human-level play by exploring strategies beyond those seen in the training set. This would require significant engineering effort to set up a stable RL training loop within the emulator, but it could also lead to groundbreaking results.

Finally, offline RL methods could be explored as a "drop-in" replacement for imitation learning altogether, keeping the same foundations of data curation and evaluation. Offline RL has shown promise in other domains for learning from fixed datasets without online interaction, and could potentially leverage the same replay data we used for IL. This would allow for a more direct comparison between IL and RL approaches in SSBM, shedding light on which paradigm is better suited for learning complex game behaviors from human data.
Overall, while our work has provided valuable insights into the specialization-generalization trade-off in imitation learning for SSBM, there remains a rich landscape of future research opportunities to further explore and expand upon these findings.

7 Closing Remarks

The source code for all experiments done throughout this project is available on GitHub at `https://github.com/Josef-Hlink/fax`, along with a carefully written README file with all instructions needed to reproduce, or even extend, our work. This project would not have been possible without the open-source efforts of many others in the SSBM and machine learning communities, so we want to take this opportunity to thank them for their contributions. In particular, we want to thank Eric Gu for open-sourcing his imitation learning codebase [19], which we heavily built upon, as well as Vlad Firoiu for his pioneering work on reinforcement learning in SSBM [13] and his publication of the ranked netplay dataset [21]. Finally, I want to thank my supervisors, Aske Plaat and Mike Preuss for their patience and guidance throughout this project.

References

[1] David Silver et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[2] Oriol Vinyals et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[3] OpenAI et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[4] Masahiro Sakurai et al. Super smash bros. melee official page (in japanese only). `https://www.nintendo.co.jp/n01/n64/software/nus_p_nalj/smash/flash/index.html`, 2001. Last visited: 2025-09-23.

[5] William Cozens. No nintendo, no problem: How smash bros. continues to thrive without its creator. `https://www.espn.com/esports/story/_/id/20421406/no-nintendo-no-problem-how-smash-bros-continues-thrive`, 2017. Last visited: 2025-09-23.

[6] Jas Laferriere et al. Project slippi - melee online with rollback netcode. `https://slippi.gg/`, 2019. Last visited: 2025-09-23.

[7] Henrik Rydgård et al. Dolphin emulator - gamecube and wii emulator. `https://dolphin-emu.org/`, 2003. Last visited: 2025-09-23.

[8] T. Harper. *The Culture of Digital Fighting Games: Performance and Practice*. Routledge, 1st edition, 2013.

[9] Yoshina Takano et al. Applying hybrid reward architecture to a fighting game ai. In *2018 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–4, 2018.

[10] Emily Halina and Matthew Guzdial. Diversity-based deep reinforcement learning towards multidimensional difficulty for fighting game ai, 2022.

[11] Inseok Oh et al. Creating pro-level ai for a real-time fighting game using deep reinforcement learning, 2020.

[12] Shao-Xiang Go et al. A phase-change memristive reinforcement learning for rapidly outperforming champion street-fighter players. *Advanced Intelligent Systems*, 5(11):2300335, 2023.

[13] Vlad Firoiu et al. Beating the world's best at super smash bros. with deep reinforcement learning. *CoRR*, abs/1702.06230, 2017.

[14] Karl Weiss, , et al. A survey of transfer learning. *Journal of Big Data*, 3(1):9, 2016.

[15] Ashutosh Adhikari et al. Docbert: BERT for document classification. *CoRR*, abs/1904.08398, 2019.

[16] Tom B. Brown et al. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020.

[17] David Silver et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[18] Eric Gu. Training ai to play super smash bros. melee. https://ericyuegu.com/melee-pt1, 2024. Blog post, Last visited: 2025-09-23.

[19] Eric Gu. Hal: Super smash bros. melee ai system. https://github.com/ericyuegu/hal, 2024. GitHub repository, Last visited: 2025-09-23.

[20] Andrej Karpathy. nanogpt. https://github.com/karpathy/nanoGPT, 2023. GitHub repository, Last visited: 2025-09-23.

[21] Vlad Firoiu. Ssbm netplay ranked replays. https://www.dropbox.com/scl/fo/r9qremhl811h6vl6kadfy/AJo-dt9-WC47Qm-s2eRlh9U?rlkey=jn88morgmcy1f1qvc030z5rrd&e=2&st=c6kexo8v&dl=0, 2025. Last visited: 2025-09-23.

[22] Peer Schneider et al. Super smash bros. melee wiki guide: Basic tips. https://www.ign.com/wikis/super-smash-bros-melee/Basic_Tips, 2001. IGN, Last visited: 2025-09-23.

[23] MosaicML. Mosaicml data shards. https://docs.mosaicml.com/projects/streaming/en/latest/preparing_datasets/dataset_format.html, 2022. Last visited: 2025-09-23.