Generative Modeling of Individual Behavior At Scale

Anonymous Author(s)
Affiliation
Address
email

Abstract

There has been a growing interest in using AI to model human behavior, particularly in domains where humans interact with this technology. While most existing work models human behavior at an aggregate level, our goal is to model behavior at the individual level. Recent approaches to *behavioral stylometry*—or the task of identifying a person from their actions alone—have shown promise in domains like chess, but these approaches are either not scalable (e.g., fine-tune a model for each person) or not generative, in that they cannot generate actions in the style of each person. We address these limitations by casting behavioral stylometry as a multi-task learning problem—where each *task* represents a distinct *person*—and using parameter-efficient fine-tuning (PEFT) methods to learn an explicit *style vector* for each person. Style vectors are generative: they selectively activate shared "skill" parameters to generate actions in the style of each person. They also induce a latent style space that we can interpret and manipulate algorithmically. In particular, we develop a general technique for *style steering* that identifies a subset of players with a desired style property, and steers a new player towards that property. We apply our approach to two very different games, at unprecedented scale: chess (47,864 players) and Rocket League (2,000 players).

1 Introduction

The rapid advances in machine learning in recent years has made it increasingly important to find constructive ways for humans to interact with this technology. Even in domains where AI has achieved proficiency, it is often important to understand how humans approach these tasks. Such an understanding can help identify areas for improvement in humans, develop better AI collaborators or teachers, create more human-like experiences, and more.

A common method for capturing human behavior is behavioral cloning (BC), a form of imitation learning [Schaal, 1996] that applies supervised learning to fixed demonstrations collected for a given task. While traditionally used in domains such as robotics [Florence et al., 2022] and self-driving vehicles [Pomerleau, 1988], BC has seen increasing use in gaming, such as in Counter-Strike [Pearce and Zhu, 2022], Overcooked [Carroll et al., 2019], Minecraft [Schäfer et al., 2023], Bleeding Edge [Jelley et al., 2024], and chess McIlroy-Young et al. [2020].

The above work focuses on modeling human behavior in aggregate, with the goal of developing better AI partners, opponents, and training tools. However, we believe that the most value for such goals can be derived by modeling human behavior at the individual level. To that end, recent results in chess have shown the most promise. McIlroy-Young et al. [2020] used behavior cloning to create a set of models called Maia that match human play at 9 aggregate skill levels. By fine-tuning these models on the data of 400 individual players, they created 400 personalized models that achieve 4-5% higher move-matching accuracy on average [McIlroy-Young et al., 2022]. The authors use these models to perform *behavioral stylometry* with high accuracy, where the goal is to identify which person played a given query set of games; in this case, they simply apply each of the 400 models to the query set

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

and output the one with the highest accuracy. McIlroy-Young et al. [2021] propose a more scalable approach of training a Transformer-based embedding on the games of each player, and use this to perform accurate stylometry across 2,844 players; in this case, they compute the embedding of the query set of games and match it to the closest player's embedding.

These approaches have different merits. The individual model approach creates a generative model for each player, but it is not scalable and shares only initial (base model) knowledge across the players; adding a new player requires fine-tuning a separate model. The embedding approach is much more scalable: it learns a compact (single-vector) representation of each player in a shared style space, and supports few-shot learning to embed a new player in this space. It cannot be used to generate moves, however, and hence cannot reason about player behavior in practice.

An ideal solution would combine these properties: generative, scalable, shared knowledge, compact representation. Our key insight for achieving this is to view behavioral stylometry as a multi-task learning problem, where each *task* represents an individual *person*. The goal here is to generalize across an initial set of players (tasks) while supporting few-shot learning of new players (tasks). To do this efficiently, we leverage recent advances in parameter-efficient fine-tuning (PEFT) [Ponti et al., 2023, Caccia et al., 2022]. Specifically, we augment an existing BC model with a set of Low Rank Adapters (LoRAs) as well as a routing matrix that specifies a distribution over these adapters for each player. Unlike approaches that train a separate LoRA for each task, this modular design allows players to softly share parameters in a fine-grained manner. We apply this adapter framework to two very different game models (which we create): a modified version of the Maia model for chess, and a Transformer-based BC model for Rocket League, a 3D soccer video game played by cars in a caged arena. (Our models scale beyond the prior art and may be of independent interest.) Our methodology first trains the BC models to convergence across all player data, and then fine-tunes the adapters and routing matrix on per-player data. This encourages the adapters to learn different *latent skills* that explain the variance between players, while each row of the routing matrix induces a weight distribution over these skills. We call each row the *style vector* for the corresponding player.

Style vectors are versatile and powerful. They support few-shot learning which enables stylometry at scale. They induce a generative model for each player that we can run and observe. They induce a shared style space that we can interpret and manipulate algorithmically. Leveraging these properties, we develop a general technique for *style steering* that identifies a subset of players who exhibit a desired style property, and steers a new player towards that property. Our main results include:

1. We perform behavioral stylometry at an unprecedented scale for chess (47,864 players, 94.4% accuracy) and Rocket League (2,000 players, 86.7% accuracy), using a query set of 100 games.
2. Our per-player generative models achieve move-matching accuracy in the range 45-69% for chess and 44-72% for Rocket League, even for players with very few (e.g., 50) games.
3. Style vectors capture a wide diversity of playing styles and strengths. They can be combined, interpolated, and steered, while reflecting consistent changes to play style and strength.

2 Background and Framing

We frame behavioral stylometry and per-player generative modeling as a multitask learning problem, to which we apply PEFT methods. In multitask learning [Caruana, 1997, Ruder et al., 2019], we are given a collection of tasks $\mathcal{T} = \big(\mathcal{T}_1, \ldots, \mathcal{T}_{|\mathcal{T}|}\big)$, each task $\mathcal{T}_i$ associated with a dataset $\mathcal{D}_i = \big\{(x_1, y_1), ..., (x_{n_i}, y_{n_i})\big\}$. Multitask learning exploits the similarities among related training tasks by transferring knowledge among them; ideally, this builds representations that are easily adaptable to new tasks using potentially few target examples. The premise of this paper is that modeling individual human behavior from a pool of players can be interpreted as a multitask learning problem. In other words, each task $\mathcal{T}_i$ consists of modeling the behavior of a specific player i; and dataset $\mathcal{D}_i$ corresponds to the sequence of game actions taken by player i. Specifically, an (x, y) tuple denotes a game state x at a specific point in time during game, along with the action y that player i took in this state. For the rest of the paper, we use the notion of *tasks* and *players* interchangeably.

2.1 Parameter-efficient fine-tuning

Popularized in NLP, parameter-efficient fine-tuning (PEFT) [Houlsby et al., 2019, Hu et al., 2022, Liu et al., 2022] approaches have emerged as a scalable solution for adapting Large Language Models to several downstream tasks. Indeed, standard finetuning of pretrained LLMs requires updating (and storing) possibly billions of parameters for each task. PEFT methods instead freeze the pretrained model and inject a small set of trainable task-specific weights, or "adapters".

One such approach is the use of Low Rank Adapters (LoRA) [Hu et al., 2022], which modify linear transformations in the network by adding a learnable low rank shift

$$h = \left(\boldsymbol{W}_0 + \Delta \boldsymbol{W}\right) x = \left(\boldsymbol{W}_0 + \boldsymbol{A}\boldsymbol{B}^T\right) x. \tag{1}$$

Here, $\boldsymbol{W}_0 \in \mathbb{R}^{d \times d}$ are the (frozen) weights of the pre-trained model, and $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^{d \times r}$ the learnable low-rank parameters of rank $r \ll d$. With this approach, practitioners can trade off parameter efficiency with expressivity by increasing the rank r of the transformation.

2.2 Polytropon and Multi-Head Adapter Routing

Standard PEFT methods such as LoRA can adapt a pretrained model for a given task. In multitask settings, training a separate set of adapters for each task is suboptimal, as it does not enable any sharing of information, or *transfer*, across similar tasks. On the other hand, using the same set of adapters for all tasks risks *negative interference* [Wang et al., 2021] across dissimilar tasks. Polytropon [Ponti et al., 2019] (`Poly`) addresses this transfer/interference tradeoff by softly sharing parameters across tasks. That is, each `Poly` layer contains 1) an inventory of LoRA adapters

$$\mathcal{M} = \{\boldsymbol{A}^{(1)}\boldsymbol{B}^{(1)}, \ \ldots \ , \ \boldsymbol{A}^{(m)}\boldsymbol{B}^{(m)}\},$$

with $m \ll |\mathcal{T}|$, and 2) a task-routing matrix $\boldsymbol{Z} \in \mathbb{R}^{|\mathcal{T}| \times m}$, where $\boldsymbol{Z}_\tau \in \mathbb{R}^m$ specifies task τ's distribution over the shared modules. This formulation allows similar tasks to share adapters, while allowing dissimilar tasks to have non-overlapping parameters. The collection of adapters $\mathcal{M}$ can be interpreted as capturing different facets of knowledge, or *latent skills*, of the full multitask distribution.

At each forward pass, `Poly` LoRA adapters for task τ are constructed as follows:

$$\boldsymbol{A}^\tau = \sum_i \alpha_i \boldsymbol{A}^{(i)}; \ \boldsymbol{B}^\tau = \sum_i \alpha_i \boldsymbol{B}^{(i)} \tag{Poly}$$

where $\alpha_i = \texttt{softmax}(\boldsymbol{Z}[\tau])_i$ denotes the mixing weight of the i-th adapter in the inventory, and $\boldsymbol{A}^{(i)}, \boldsymbol{B}^{(i)}, \boldsymbol{A}^\tau, \boldsymbol{B}^\tau \in \mathbb{R}^{d \times r}$. Here, the τ-th row of the routing matrix $\boldsymbol{Z}$ is effectively selecting which adapter modules to include in the linear combination. In our setting, where each task consists of modeling an individual, $\boldsymbol{Z}[\tau]$ specifies which latent skills are activated for user τ; we call this their *style vector*. As per Eqn 1, the final output of the linear mapping becomes $h = \left(\boldsymbol{W}_0 + \boldsymbol{A}^\tau(\boldsymbol{B}^\tau)^T\right) x$.

In `Poly`, the module combination step remains *coarse*, as only linear combinations of the existing modules can be generated. Caccia et al. [2022] propose a more fine-grained approach, called Multi-Head Routing (`MHR`), which is what we use in our work. Similar to Multi-Head Attention [Vaswani et al., 2017], the input dimension of $\boldsymbol{A}$ (and output dimensions of $\boldsymbol{B}$) are partitioned into h heads, where a `Poly`-style procedure occurs for each head. The resulting parameters from each head are then concatenated, recovering the full input (and output) dimensions. See A.1 for more details.

Routing-only fine-tuning. While LoRA adapters can reduce the parameter cost from billions to millions [Liu et al., 2022], training the adapters for each new task can still be prohibitive when dealing with thousands of tasks. To this end, Caccia et al. [2022] proposed routing-only finetuning, where after an initial phase of pretraining, the adapter modules are fixed, and only the routing parameters $\boldsymbol{Z}$ are learned for a new task. This reduces the parameter cost for each additional task by several orders of magnitude, while maintaining similar performance. We use this method for few-shot learning.

3 ML Methodology

In this section, we detail our methodology for creating a generative model of individual behavior that enables our style analyses. Our methodology applies to any behavior cloning scenario with access to human demonstrations from multiple individuals. To demonstrate this generality, we apply it to two very different games: chess and Rocket League. We start with a base model for each and apply the `MHR` adapter framework to it, and then discuss model training and evaluation.

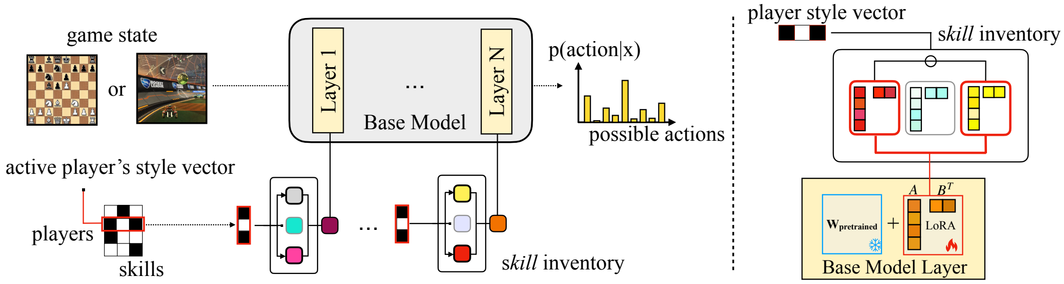

Figure 1: (left) Our overall architecture. We augment a base model with a set of MHR adapters and a routing matrix composed of each player's style vector. (right) Detailed view of an MHR layer, showing a skill inventory of adapters shared across players. The player's style vector specifies which skills are active (in this case, the first and third) to generate the final low-rank weight shift that is applied to the (frozen) base model layer.

3.1 Model architecture

For chess, we follow McIlroy-Young et al. [2022] and use the Squeeze-and-Excitation (S&E) Residual Network [Hu et al., 2018] as a base model, but with a deeper and wider configuration (see A.2). At every residual block, an additional 2-layer MLP rescales the residual output along the channel dimension to explicitly model channel interdependencies. The input is a 112-channel 8×8 image representation of the chess board; the output is the predicted move represented as a 1858-dimensional vector. The total parameters is 15.7M. For Rocket League, we use the GPT-2 architecture from Radford et al. [2019] with a dimensionality of 768, 12 attention heads, and 12 layers. The input is a 49-dimensional vector with game physics information; the output is 8 heads: 5 with 3 bins of [-1, 0, 1] and 3 binary. The model has no embedding layer, as the game data points are passed directly as tokens after processing. The total parameters is 87.7M.

To enable user-based adaptation, we incorporate the `MHR` adapters described in §2.2 into our base models, as illustrated in Fig. 1. In chess, for every linear transformation in the MLP used for channel-wise rescaling, we add an `MHR` layer built of LoRA adapters with rank 16, for a total of $12\times2=24$ `MHR` layers. We use an adapter inventory of size 32 and a multi-head routing strategy with 8 heads. Therefore, for each user we must learn $32\times8=256$ routing parameters as their style vector. This yields 5M additional parameters. For Rocket League, we attach the adapters to the fully connected layer of each transformer block, resulting in 12 `MHR` layers of LoRAs with rank 16. We use an inventory size of 16 and 64 heads. This yields 13.8M additional parameters. To facilitate interpretability and style analysis, we use the same routing (style vector) across all `MHR` layers.

3.2 Data collection and partitioning

We use data from the largest open-source online chess platform, Lichess.org [Duplessis, 2021], which boasts a database of over 4.8 billion games. We collected Blitz games played between 2013 and 2020 inclusive—these are games with 3 or 5 minutes per side, optionally with a few seconds of time increment per move—and applied the same player filtering criteria as McIlroy-Young et al. [2022]. The resulting dataset comprises 47,864 unique players and over 244 million games. (See A.2 for a discussion on data imbalance.) For Rocket League, we collect data from a large open-source replay database, Ballchasing.com [CantFlyRL, 2024]. We use 2.2 million 1v1 replays from 2015 to mid-2022, totalling several decades of human game play hours at 5 minutes per game. After parsing, each Rocket League game state is a vector holding the player's 3D position, linear and angular velocity, boost remaining, rotation, and team; we also include the opponent's state and the position, linear and angular velocity of the ball. Given a game state, we have to predict the user's throttle, steer (while grounded), pitch, yaw, roll (while aerial), jump, boost, and handbrake. Additional processing was needed to correct for missing aerial controls and inconsistent sampling rates (24-27hz). Our full data processing procedure, including the challenges we faced, are detailed in A.3.

We divide the set of players into a few subsets to support our training methodology. The *base player* set comprises all data and is used to train the base models. The *fine-tuning player* set is used to fine-tune the `MHR` architecture shown in Fig. 1. (For both, we split each player's data into 80/10/10 for train/test/validation.) The *few-shot player* set is used for few-shot learning based on a reference set of

100 games per player. For our chess experiments, to enable a direct comparison with prior work, we create an additional fine-tuning player set consisting of the same 400 players used in those studies. Currently, we treat each player's data holistically, but in principle one could partition a player's data in different ways to perform a finer analysis of their playing style. We explore this in A.4.

3.3 Model training and evaluation

Base model. We train our base Maia model for chess using data from a base player set of all 47,864 players, treating this as a classification task of predicting human move y made in chess position x, given a datapoint (x, y). We use the same loss functions and evaluation criteria as the original Maia work: Maia's policy head uses a cross entropy loss while the value head uses MSE; the output of the policy head is used to evaluate the model's move-matching accuracy.

We train our Rocket League model using a base player set of over 800,000 players, though the vast majority of players have 5 games or fewer. We discretize the actions into 3 bins for throttle, steer, pitch, yaw, and roll, as most of this data is close to 0, -1, or 1. We use binary outputs for jump, boost, and handbrake. A next-move prediction is labelled correct if and only if all of the outputs are correct.

`MHR` **fine-tuning.** To train the `MHR` LoRA adapters, we adopt the methodology used in Caccia et al. [2022]: namely, we freeze the base model and fine-tune the `MHR` layers and routing matrix using data from a fine-tuning player set. Recall that the routing matrix $\boldsymbol{Z}$ has a row (style vector) for each player in the fine-tuning set. Following Ponti et al. [2019], we use a two-speed learning rate, where the style vectors' learning rate is higher than the adapters', to enable better specialization.

For chess, we use two fine-tuning player sets in our experiments, creating two separate `MHR`-Maia models. The first set comprises all 47,864 players and is used to evaluate behavioral cloning and stylometry at very large scale. The second set is comprised of the same 400 players used by McIlroy-Young et al. [2022], which we use to compare few-shot learning and stylometry results. For Rocket League, we train an `MHR`-Rocket model on a fine-tuning set of 2,000 players with 100 games each.

Few-shot learning. To perform few-shot learning on our `MHR` models, we perform the "routing-only fine-tuning" described in section 2.2 that additionally freezes all `MHR` LoRA adapters. Given a few-shot player, we add a (randomly-initialized) new row to $\boldsymbol{Z}$ and fine-tune it on the player's reference set of games, eventually learning a style vector for the player. Using this style vector, we can invoke a generative model of the player and use it to evaluate move-matching accuracy, as described above. To perform stylometry, if the player is a *seen* player (i.e., part of the fine-tuning set), then a matching style vector already exists in $\boldsymbol{Z}$, and we can find it using cosine similarity. Otherwise, if the player is *unseen*, then we simply repeat the few-shot learning process on a query set of games (from the same player), and compare this new style vector to the entries in $\boldsymbol{Z}$.

For chess, (unless stated otherwise), all of our few-shot experiments use the `MHR`-Maia model fine-tuned on the 400-player set from McIlroy-Young et al. [2022]. For Rocket League, the few-shot player set consists of 1,000 of the 2,000-player set used to fine-tune `MHR`-Rocket.

Evaluation. We evaluate a fine-tuned `MHR` model in two ways. First, we measure its move-matching accuracy, similar to how we evaluate the base models. However, since our `MHR` models provide a generative model for each player (invoked through their style vector), we can separately evaluate each player's model by applying it to their test set and measuring move-matching accuracy. The overall move-matching accuracy for the model is simply the average of these per-player accuracies.

Our second evaluation method uses the model to perform behavioral stylometry among all players in the fine-tuning set. This is done by leveraging our few-shot learning methodology (above). That is, given a query set of games from some player, we learn a new style vector in $\boldsymbol{Z}$ for those games via few-shot learning, and compare this vector to every other vector in $\boldsymbol{Z}$. Using cosine similarity as our distance metric, we simply output the player with the highest cosine similarity to the query set vector.

4 Style methodology

The style vectors in $\boldsymbol{Z}$ represent distinct distributions over latent skills that give us a starting point for comparing player styles. For example, our stylometry method above uses the cosine similarity of

Method	$\|Query\|$	$\|Universe\|$	$\|Query\ Games\|$	Random (%)	Acc. (%)
Seen few-shot players					
McIlroy-Young et al. [2022]	400	400	100	0.25	98.0
McIlroy-Young et al. [2021]	400	400	100	0.25	99.5
MHR-Maia	400	400	100	0.25	**99.8**
McIlroy-Young et al. [2022]	400	400	30	0.25	94.0
MHR-Maia	400	400	30	0.25	**98.8**
MHR-Maia	10000	47864	100	0.002	**94.4**
Unseen few-shot players					
McIlroy-Young et al. [2021]	578	2844	100	0.035	79.1
MHR-Maia (100 games)	10000	10000	100	0.01	87.6

Table 1: Stylometry accuracy results. *Seen* few-shot players are a subset of the fine-tuning player set, unlike *unseen* players. Numbers for McIlroy-Young et al. [2022] and McIlroy-Young et al. [2021] are borrowed from their respective papers.

these vectors to determine how similar or different players are. However, style vectors also enable much more powerful capabilities, such as the ability to synthesize new (human-like) styles.

To begin, we measure the intra-player consistency of style vectors by splitting a player's dataset into disjoint subsets of varying size, and few-shot learning a style vector for each subset. We then investigate inter-player consistency by merging the datasets of two players and seeing if the style vector trained on the merged dataset is similar to the average of the two player's style vectors.

The latter method actually creates a new playing style that is human-like and yet previously unseen in the world. This suggests a more general approach to style synthesis: interpolate between players using a convex combination of their style vectors. To determine the playing strength of these new players, we can simulate games between them and the players they are derived from. The results of these games can be used to calculate a win rate, which can then be converted to a strength rating.

Currently, our advanced style synthesis techniques focus on chess, where there is a robust mapping between win rates and playing strength (the Elo rating system), and simulating games is cheap. Rocket League simulations are quite costly at present, but in principle the same methodology should apply and we plan to reduce these costs in future work.

In order to make style comparisons more human-understandable, we again exploit the generative nature of our MHR models. Inspired by the concept probing technique used to analyze AlphaZero (a deep RL chess engine) [McGrath et al., 2022], we use a set of human-coded heuristic functions found in Stockfish (a traditional chess engine) to evaluate a player's model. These functions capture concepts such as: king safety, material imbalance, piece mobility, and so on. By invoking a player's model on a fixed set of chess positions and seeing which move they select, we can use this to summarize how much emphasis the player places on the corresponding concepts.

Finally, we combine the above methods to design a simple but general method for *steering* a player's game style towards a specific attribute a, such as increasing their king safety, while limiting the changes on other attributes (so as to preserve their style). To achieve this, we first collect a set players X who exhibit high values for attribute a—determined, for example, by running their generative models on a fixed set of game states. We then extract the common direction among these players, by averaging their style vectors and subtracting the population average. This yields a *style delta vector* that can be added to any player's style vector to elicit the desired change.

5 Experiments

In this section, we demonstrate two main findings. First, MHR-Maia performs competitively with prior methods for behavior cloning and stylometry in chess, while achieving unprecedented scale. We also show that our approach can be applied to Rocket League, for both stylometry and move prediction. Second, we show that explicitly capturing style vectors allows us to reason about and perform arithmetic operations on generated behaviors.

5.1 Behavioral Stylometry

In this section, we show that our models perform competitively with previous behavioral stylometry methods for both seen and unseen players. Here, the goal is to predict the player who produced a given set of games. We compare to individual model fine-tuning [McIlroy-Young et al., 2022], fitting a pre-trained Maia to the data from a single player, and to a Transformer-based method [McIlroy-Young et al., 2021], which embeds players in a 512-dimensional style space based on their gameplay. All reported accuracies are top-1 unless stated otherwise.

To perform stylometry on a query set of games, McIlroy-Young et al. [2022] suggest measuring the move-matching accuracy of each available fine-tuned model and selecting the best performing model. As seen in Table 1, this procedure works well, but is tremendously expensive—requiring computationally intensive inference calls on the entire query set for every candidate player.

In contrast, both the Transformer-based method and `MHR`-Maia scale much better to large numbers of players. The Transformer-based method needs only to condition on these games to produce a vector, while `MHR`-Maia needs only to fit a new vector. In either case, the produced vectors need only be matched to those in the player set, e.g., using cosine similarity. Table 1 compares both approaches, showing that `MHR`-Maia performs competitively or better, on a much larger universe. To do this, we use few-shot learning to compute style vectors for 10,000 players based on their 100 game reference sets, then fit new style vectors for each player based on their respective query sets. Note that the individual model fine-tuning method is omitted from the larger few-shot study due to scalability reasons. The Transformer-based method can scale, but it is not a generative model.

For Rocket League, to the best of our knowledge, we are the first to attempt stylometry. We report player identification results averaged over the few-shot player set. For each prediction, our `MHR`-Rocket approach must correctly identify each of the 1,000 players among a pool of 2,000 players. Yet, it reaches an accuracy of **86.7%** (random: 0.05%), showcasing the validity of our approach.

5.2 Move generation

Here we compare the efficacy of our method to using individually fine-tuned models for each player. Fine-tuning individual models generally results in superior results compared to PEFT methods, as the increased parameter count produces more expressive models. However, they are also more computationally intensive to train and store. That said, in the domain of modeling individual behavior in chess, `MHR`-Maia is able to perform comparatively well despite using a much smaller parameter budget. Figure 2 shows that `MHR`-Maia matches individual model fine-tuning over a wide range of game counts. The base model is frozen for all game counts in `MHR`-Maia. The model has already learned the set of skills required to differentiate the players, all that is needed with very few-shot learning is to find a proper recombination of the learned skills within the new style vectors. The Transformer-based method is omitted, as it is incapable of generating moves.

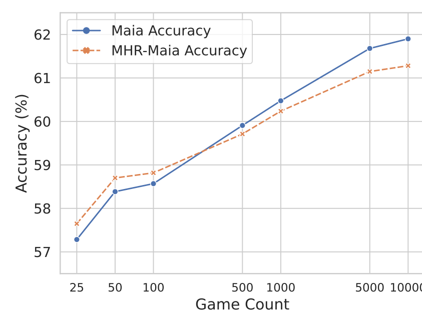

Figure 2: Accuracy at various game counts of the individual models (Maia) and our method (`MHR`-Maia).

For Rocket League, we compare the next move prediction of our base model, with `MHR`-Rocket, to validate that our user-based conditioning generates better predictions. We find that, on average, `MHR`-Rocket increases the next move prediction from **53.1%** to **56.1%**.

5.3 Analysis of style vectors

In this section, we explore the consistency of our style vectors across different players and datasets.

Consistency across a single player. To showcase intra-player consistency, we first partition 50 players' datasets into disjoint subsets. We use 50 splits for chess and 20 for Rocket League. The subsets are sampled across a wide range of dates, opposing players, and playing sessions. Next, we train a style vector for every split across all players. We find that vectors corresponding to the same player will be similar to each other, and have low similarity with the other players and general population. This is visualized in Figure 3. This suggests that our neural network is able to find

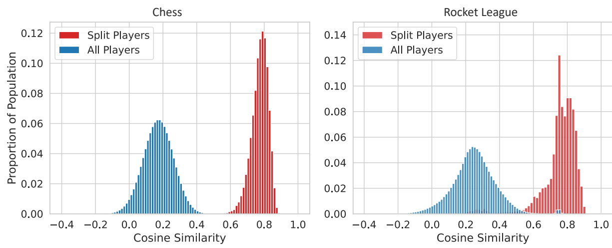

Figure 3: Cosine similarity between style vectors learned from different partitions of the same player (red) vs across different players (blue).

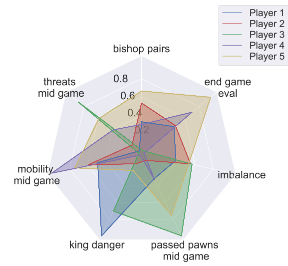

Figure 4: Comparing player styles using human-interpretable evaluation metrics.

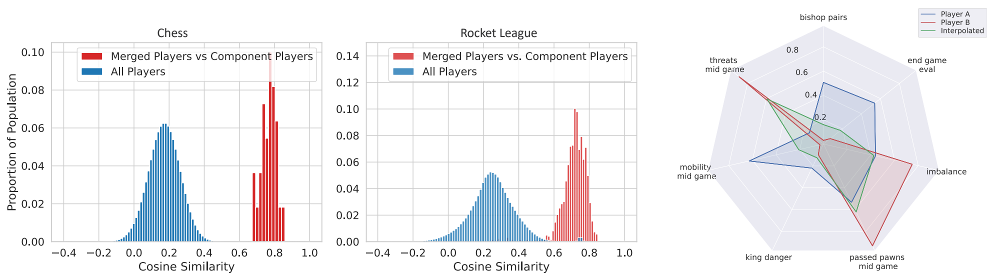

Figure 5: Cosine similarity between averaged style vectors of two players, and the learned style vectors on their merged datasets (red) vs across the full population (blue). The style of an intermediate player (green) is shown along with the two component players (blue and red) on the right.

distinct tendencies for each player. To confirm, we sampled 5 random chess players, predicted their preferred move across 2^{17} positions, and measured a series of Stockfish evaluation metrics per player. Figure 4 shows the distribution of these metrics for each player, demonstrating that these vectors store a wide diversity of styles.

Consistency across merged players. To parse out whether we can generate new styles using this information, we merged two players' datasets together to generate a new set with the tendencies of both players, measuring inter-player consistency. We then compared this new set of vectors to a different set of vectors generated by simply averaging the style vectors of the player pair. As seen in Figure 5 (left and center), vectors with the same two source players have very high similarity in both chess and Rocket League. We then sampled a random pair in the merged dataset, created a new player by averaging the two players' vectors, and recorded their gameplay according to the previous section. The results are visualized in Figure 5 (right), showing that the new player (green) has an intermediate playing style to the source players (red, blue).

5.4 Synthesis of new styles

Convex combinations. We show that interpolating between skill vectors results in a player whose level is a weighted average of the interpolated players. Here, we take 100 pairs of learned player vectors, such that one item in the pair corresponds to a strong player and the other to a weaker player. We then gradually interpolate between the weak and strong player as $(1-\lambda)u_w + \lambda u_s$, $0 \leq \lambda \leq 1$, where u_w and u_s are respectively vectors representing the weak and strong player. For each value of λ we simulate 1,000 games between the interpolated vector and u_s, the stronger player.

Figure 6 plots the win rate of the interpolated player as a function of λ for each player pair we considered. This plot demonstrates that win rate progresses in a roughly linear fashion, starting off winning infrequently against the stronger player and eventually winning roughly half the time as the interpolated player converges to the stronger player.

Directly steering player style. Finally, we directly control the playing style of a player by creating skill vectors according to the procedure described in 4. We choose players in our chess dataset with high (>2 std) bishop pair utilization, and separately players with high king danger. Figure 7 shows

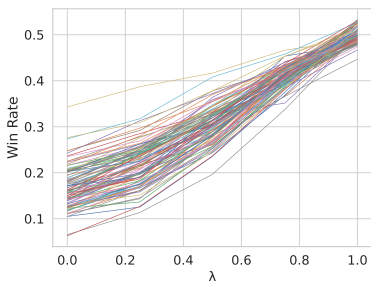

Figure 6: Winrate as a weaker player is interpolated with a stronger player as a function of λ.

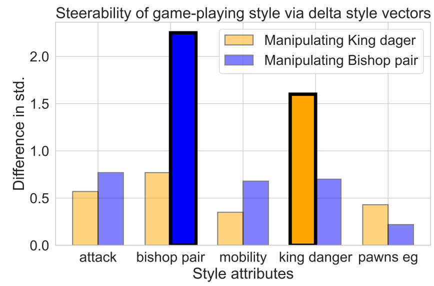

Figure 7: Modifying the a player's style towards a specific attribute

the change in 2,000 randomly sampled player's stockfish evaluations after adding the skill vector corresponding to each heuristic to their style vectors. Indeed, we see that the player's style is steered towards the attribute in question, with model impact on other attributes.

6 Related Work

Stylometry and player style modeling. Originally referring to performing author attribution via statistical analysis of text [Tweedie et al., 1996, Neal et al., 2017], stylometry has since come to refer to the general task of identifying individuals given a set of samples or actions, and has found broad application for tasks such as handwriting recognition [Bromley et al., 1993], speaker verification [Wan et al., 2018], identifying programmers from code [Caliskan-Islam et al., 2015], determining user age and gender from blog posts [Goswami et al., 2009], and identifying characteristics of authors of scientific articles [Bergsma et al., 2012]. In the context of gaming (covered in the introduction), stylometry is closely related to playstyle modeling, where the goal is to associate a player with a reference style, such as by building agents representative of different playstyles and find the closest behavioral match [Holmgård et al., 2014], or gathering gameplay data and applying methods such as clustering [Ingram et al., 2022], LDA [Gow et al., 2012], Bayesian approaches [Normoyle and Jensen, 2015] and sequential models [Valls-Vargas et al., 2015] to identify groups of players with similar styles. Unlike our work, these approaches focus on aggregate playstyles, and do not learn generative models that can be conditioned on an individual's style.

Our method for style synthesis is inspired by earlier work on vector arithmetic with embeddings [Church, 2017], as well as recent work on steering multiask models with task vectors [Ilharco et al., 2023]. Finally, our steering method is reminiscent of Radford et al. [2016], which manipulates the model's latent space to generate images containing specific attributes.

Parameter-efficient adaptation Approaches for efficient adaption of a pretrained model can be broadly grouped in two categories. Adapter based methods inject new parameters within a pretrained model, and only updates the newly inserted parameters while keeping the backbone fixed. Houlsby et al. [2019] defines an adapter as a two-layer feed-forward neural network with a bottleneck representation, and are inserted before the multi-head attention layer in Transformers. Similar approaches have been used for cross-lingual transfer [Pfeiffer et al., 2020]. Adapters have also been used in vision based multitask settings [Rebuffi et al., 2017]. More recently, Ansell et al. [2022] propose to learn sparse masks, and show that these marks are composable, enabling zero-shot transfer. Lastly, Hu et al. [2022] learn low-rank shifts on the original weights, and [Liu et al., 2022] learns an elementwise multiplier of the pretrained model's activations. Adapters have also been used in multitask settings. Chronopoulou et al. [2023] independently trains adapters for each task. In order to transfer to new tasks, the authors merge the parameters of the adapters of relevant training tasks.

7 Conclusion

We show that individual player behavior can be modeled at very large scale in games as different as chess and Rocket League. We cast this problem in the framework of multi-task learning and employ modular PEFT methods to learn a shared set of skills across players, modulated by a distinct style vector for each player. We use these style vectors to perform behavioral stylometry, analyze player styles, and synthesize and steer new styles.

References

A. Ansell, E. Ponti, A. Korhonen, and I. Vulić. Composable sparse fine-tuning for cross-lingual transfer. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1778–1796, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.125. URL `https://aclanthology.org/2022.acl-long.125`.

S. Bergsma, M. Post, and D. Yarowsky. Stylometric analysis of scientific articles. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 327–337, 2012.

R.-A. Braaten. Rl-rpt - rocket league replay pre-training. https://github.com/Rolv-Arild/replay-pretraining, 2022.

J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

L. Caccia, E. Ponti, L. Liu, M. Pereira, N. L. Roux, and A. Sordoni. Multi-head adapter routing for data-efficient fine-tuning. *arXiv preprint arXiv:2211.03831*, 2022.

A. Caliskan-Islam, R. Harang, A. Liu, A. Narayanan, C. Voss, F. Yamaguchi, and R. Greenstadt. De-anonymizing programmers via code stylometry. In *24th USENIX security symposium (USENIX Security 15)*, pages 255–270, 2015.

CantFlyRL. Ballchasing.com. https://ballchasing.com/, 2024.

M. Carroll, R. Shah, M. K. Ho, T. Griffiths, S. Seshia, P. Abbeel, and A. Dragan. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems*, 32, 2019.

R. Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997.

A. Chronopoulou, M. Peters, A. Fraser, and J. Dodge. AdapterSoup: Weight averaging to improve generalization of pretrained language models. In *Findings of the Association for Computational Linguistics: EACL 2023*, pages 2054–2063, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-eacl.153. URL `https://aclanthology.org/2023.findings-eacl.153`.

K. W. Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

T. Duplessis. Lichess. http://lichess.org, 2021. Accessed: 2021-01-01.

L. Emery. Rlgym - the rocket league gym. https://rlgym.org/, 2021.

P. Florence, C. Lynch, A. Zeng, O. A. Ramirez, A. Wahid, L. Downs, A. Wong, J. Lee, I. Mordatch, and J. Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

S. Goswami, S. Sarkar, and M. Rustagi. Stylometric analysis of bloggers' age and gender. In *Proceedings of the International AAAI Conference on Web and Social Media*, volume 3, pages 214–217, 2009.

J. Gow, R. Baumgarten, P. Cairns, S. Colton, and P. Miller. Unsupervised modeling of player style with lda. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(3):152–166, 2012.

C. Holmgård, A. Liapis, J. Togelius, and G. N. Yannakakis. Evolving personas for player decision modeling. In *2014 IEEE Conference on Computational Intelligence and Games*, pages 1–8. IEEE, 2014.

N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, pages 2790–2799, 2019. URL `http://proceedings.mlr.press/v97/houlsby19a/houlsby19a.pdf`.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

G. Ilharco, M. T. Ribeiro, M. Wortsman, L. Schmidt, H. Hajishirzi, and A. Farhadi. Editing models with task arithmetic. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=6t0Kwf8-jrj.

B. Ingram, B. Rosman, C. van Alten, and R. Klein. Play-style identification through deep unsupervised clustering of trajectories. In *2022 IEEE Conference on Games (CoG)*, pages 393–400. IEEE, 2022.

A. Jelley, Y. Cao, D. Bignell, S. Devlin, and T. Rashid. Aligning agents like large language models, 2024. URL https://openreview.net/forum?id=kQqZVayz07.

H. Liu, D. Tam, M. Muqeeth, J. Mohta, T. Huang, M. Bansal, and C. Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning, 2022. URL https://arxiv.org/abs/2205.05638.

T. McGrath, A. Kapishnikov, N. Tomašev, A. Pearce, M. Wattenberg, D. Hassabis, B. Kim, U. Paquet, and V. Kramnik. Acquisition of chess knowledge in alphazero. *Proceedings of the National Academy of Sciences*, 119(47):e2206625119, 2022.

R. McIlroy-Young, S. Sen, J. Kleinberg, and A. Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, page 1677–1687, 2020.

R. McIlroy-Young, Y. Wang, S. Sen, J. Kleinberg, and A. Anderson. Detecting individual decision-making style: Exploring behavioral stylometry in chess. *Advances in Neural Information Processing Systems*, 34:24482–24497, 2021.

R. McIlroy-Young, R. Wang, S. Sen, J. Kleinberg, and A. Anderson. Learning models of individual behavior in chess. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, page 1253–1263, 2022.

T. Neal, K. Sundararajan, A. Fatima, Y. Yan, Y. Xiang, and D. Woodard. Surveying stylometry techniques and applications. *ACM Computing Surveys (CSuR)*, 50(6):1–36, 2017.

A. Normoyle and S. Jensen. Bayesian clustering of player styles for multiplayer games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, pages 163–169, 2015.

T. Pearce and J. Zhu. Counter-strike deathmatch with large-scale behavioural cloning. In *2022 IEEE Conference on Games (CoG)*, pages 104–111. IEEE, 2022.

J. Pfeiffer, I. Vulić, I. Gurevych, and S. Ruder. MAD-X: An Adapter-based framework for multi-task cross-lingual transfer. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7654–7673, Nov. 2020. URL https://aclanthology.org/2020.emnlp-main.617.

D. A. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Advances in neural information processing systems*, 1, 1988.

E. M. Ponti, H. O'Horan, Y. Berzak, I. Vulić, R. Reichart, T. Poibeau, E. Shutova, and A. Korhonen. Modeling language variation and universals: A survey on typological linguistics for natural language processing. *Computational Linguistics*, 45(3):559–601, 2019. URL https://watermark.silverchair.com/coli_a_00357.pdf.

E. M. Ponti, A. Sordoni, Y. Bengio, and S. Reddy. Combining parameter-efficient modules for task-level generalisation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, pages 687–702, Dubrovnik, Croatia, May 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.eacl-main.49.

A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*, 2016.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.

S.-A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. *Advances in neural information processing systems*, 30, 2017.

RLBot. Rlbot. https://github.com/RLBot/RLBot, 2017.

S. Ruder, M. E. Peters, S. Swayamdipta, and T. Wolf. Transfer learning in natural language processing. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Tutorials*, pages 15–18, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-5004. URL https://aclanthology.org/N19-5004.

SaltieRL. Carball. https://github.com/SaltieRL/carball, 2024.

S. Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

L. Schäfer, L. Jones, A. Kanervisto, Y. Cao, T. Rashid, R. Georgescu, D. Bignell, S. Sen, A. T. Gavito, and S. Devlin. Visual encoders for data-efficient imitation learning in modern video games, 2023.

F. J. Tweedie, S. Singh, and D. I. Holmes. Neural network applications in stylometry: The federalist papers. *Computers and the Humanities*, 30:1–10, 1996.

J. Valls-Vargas, S. Ontanón, and J. Zhu. Exploring player trace segmentation for dynamic play style prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, pages 93–99, 2015.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

L. Wan, Q. Wang, A. Papir, and I. L. Moreno. Generalized end-to-end loss for speaker verification. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4879–4883. IEEE, 2018.

Z. Wang, Y. Tsvetkov, O. Firat, and Y. Cao. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=F1vEjWK-lH_.

Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li. On the continuity of rotation representations in neural networks, 2020.

A Appendix

A.1 Multi-Head Adapter Routing

In Poly, the module combination step remains *coarse*, as only linear combinations of the existing modules can be generated. Caccia et al. [2022] propose a more fine-grained module combination approach, referred to as Multi-Head Routing (MHR). Similar to Multi-Head Attention [Vaswani et al., 2017], the input dimension of $\boldsymbol{A}$ (and output dimensions of $\boldsymbol{B}$) are partitioned into h heads, where a Poly-style procedure occurs for each head. The resulting parameters from each head are then concatenated, recovering the full input (and output) dimensions. This makes the module combination step *piecewise linear*, with a separate task-routing matrix $\boldsymbol{Z}$ learned for each head.

Formally, a MHR layer learns a 3-dimensional task-routing tensor $\mathsf{Z} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{M}|\times h}$. The 2D slice $\mathsf{Z}_{:,:,k} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{M}|}$ of the tensor Z denotes the distribution over modules for the k-th head, and $\boldsymbol{W}[k] \in \mathbb{R}^{\frac{d}{h}\times r}$ the k-th partition along the rows of the matrix $\boldsymbol{W} \in \mathbb{R}^{d\times r}$. The adapter parameters $\boldsymbol{A}^\tau \in \mathbb{R}^{d\times r}$ for task τ, and for each adapter layer, are computed as (similarly for $\boldsymbol{B}^\tau$):

$$\begin{aligned} \boldsymbol{A}_k^\tau &= \sum_j \alpha_{i,k} \cdot \boldsymbol{A}_j[k] \quad \text{with} \quad \boldsymbol{A}_k^\tau \in \mathbb{R}^{\frac{d}{h}\times r}, \\ \boldsymbol{A}^\tau &= \texttt{concat}(\boldsymbol{A}_1^\tau, \ldots, \boldsymbol{A}_h^\tau), \end{aligned} \tag{MHR}$$

where $\alpha_{i,k} = \texttt{softmax}(\boldsymbol{Z}\texttt{[}\tau\texttt{,:,k]})_i$. Importantly, the number of LoRA adapter parameters does not increase with the number of heads. Only the task-routing parameters linearly increase with h for MHR vs. Poly. However, this cost is negligible as the parameter count of the routing matrices is much smaller than for the LoRA modules themselves.

A.2 Maia Architecture/Data

Our base Maia architecture follows McIlroy-Young et al. [2022] and uses the Squeeze-and-Excitation (S&E) Residual Network of [Hu et al., 2018]. At every residual block, channel information is aggregated across spatial dimensions via a global pooling operation. The resulting vector is then processed by a 2-layer MLP, with a bottleneck representation compressing the number of channels by r. The output of this MLP is a one-dimensional vector used to scale the output of the residual block along the channel dimension. We use 12 residual blocks containing 256 filters, and a bottleneck compression factor of $r = 8$. We note that this differs from the base Maia model in McIlroy-Young et al. [2022], which uses 64 filters and 6 residual blocks.

While our dataset has a median game count of 3,479 games, many players may have as few as 10-50 games, implying some degree of data imbalance. Our evaluation of few-shot learning shows that 100 games is sufficient to learn the style vector of an unseen player. However, one might still ask how accurately such a style vector is given a very small number of games. To explore this, we first split a player into disjoint sets of 10, 25, 50, 100, 500, and 1,000 games. We then train a style vector on each set. As a baseline, we train a style vector on 10,000 games and track the cosine similarity of the smaller-set style vectors relative to this baseline vector. We show the results in Figure 8.

A.3 Rocket League Architecture/Data

The 1v1 replays dataset was scraped over the course of several weeks from the Ballchasing.com API using the Grand Champion subscription tier, though the API does have a slower free tier. This API yields raw game replays, which are uploaded by users either manually or using a community-made plugin for the game. The replays are in a binary format which must be parsed using community-made projects such as Carball [SaltieRL, 2024].

The Carball library allows us to convert the binary replay format to a more standard CSV format, which we save to a Cloud binary blob storage. The data present in both is a lossy reconstruction of game states, and requires some processing to be usable. In particular, the data is sampled at an inconsistent rate (varying between 24hz and 27hz), contains repeated physics ticks, and is missing action data for aerial controls (pitch, yaw, roll).

We resolve the issue of sampling rate and repeated ticks by removing repeated ticks, and doing a time-weighted resampling and interpolation to a standard 10hz for model training, though we found

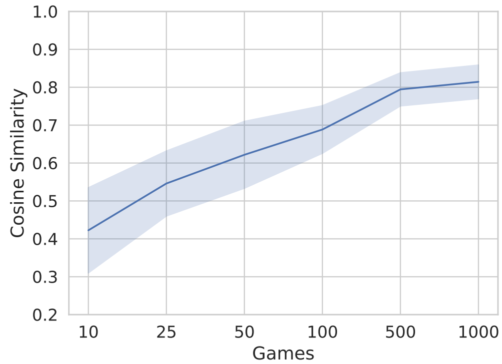

Figure 8: Cosine similarity of style vectors trained with varying game sizes compared to a style vector trained with 10,000 games, run on 50 players.

that 30hz also works well. Note that the actual game physics ticks occur at 120hz, so any value aligned with this should work. Without these changes, the model performs extremely poorly and is unable to navigate the arena.

We resolve the issue of missing aerial controls through the physics-based solver present in the Carball library. The estimation of these controls is not perfect, but it is sufficient for our purposes. Some previous community work has used inverse dynamics [Braaten, 2022] trained from rollouts of in-game bots to solve for these actions, though we opted to not use this due to the inconsistency in replay data sampling.

The data returned by the CSVs are fairly large, messy, and inconsistent. We apply the following transformations to the dataframe to bring the values closer to 0:

- Divide position by 2300
- Divide linear velocity by 23000
- Divide angular velocity by 5500
- Divide boost by 255
- Encode rotation Euler angles according to Zhou et al. [2020]

Additionally, when turning the data into tokens for use in our model, we add in an extra dimension to represent the team, and concatenate the opponent's data points along with the position, linear and angular velocity of the ball. We complete all of these transformations at runtime.

We also have to align the data returned by the simulators for Rocket League with the data used to train the model, RLBot [RLBot, 2017] and RLGym [Emery, 2021]. Along with including an extra dimension to represent the team, we apply the following transformations to all samples obtained from the game:

- Divide position by 2300
- Divide linear velocity by 2300
- Divide angular velocity by 5.5
- Divide boost by 100

The skill distribution of the players in our dataset can be found in Figure 9.

A.4 Implicit Stationarity Assumptions

Most of the existing work in chess assumes that a player remains stationary over time and across gameplay situations. However, in reality, a player's style may depend on the type of opponent they are facing, which opening is used, which stage of the game they are in (opening, middle, endgame), and so on. For instance, McIlroy-Young et al. [2021] observe that stylometry accuracy drops when

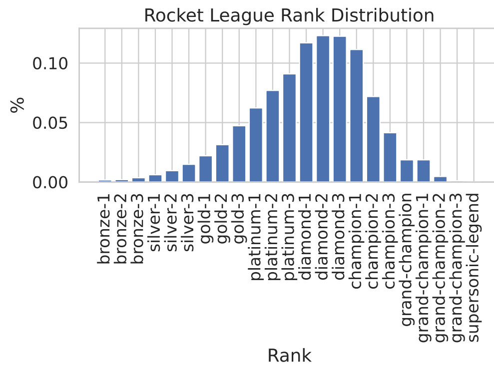

Figure 9: Skill distribution of Rocket League players in our dataset.

removing the opening (e.g., the first 15 moves) moves, suggesting that the opening has an outsized effect on style identification. Our approach does not rely on these assumptions and can in principle be applied to arbitrary subsets of a player's data. For instance, one could split a player's data into opening, middlegame, and endgame moves and train a separate style vector for each. One could further split the data based on which defense the opponent uses, what time of the day it is, etc.. Despite treating players holistically and avoiding any splits of their data, we are still able to capture the peculiarities of each individual's playing style and perform stylometry with high accuracy. This also enables us to compare our results to those of prior work, which also treats player data holistically.

A.5 Delta Style Vector Computation

Algorithm 1 Style Delta Vector computation

Input:
X : Style vectors of top-k players for attrib. a;
P : Style vectors of all players in population
Output Δ_a: Style delta vector for attr. a

$\boldsymbol{V}_a = \text{mean}(X, \texttt{axis} = \text{‘players’})$
$\boldsymbol{V}_P = \text{mean}(P, \texttt{axis} = \text{‘players’})$
$\Delta_a = \boldsymbol{V}_a - \boldsymbol{V}_P$
Returns Δ_a

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract directly summarizes the key results of the paper, which focus on performing behavioral stylometry at scale in games (chess and Rocket League)

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Please see Related work and explanation of our limited style synthesis/steering results for Rocket League.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: we dont use proofs as a contribution

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide thorough implementation details, some of which appear in the appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We will open source our data and models upon publication.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification: See Appendix and main paper for full experimental details.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: While we do not use error bars, our methodology is properly described and clarifies the significance or our results.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We talk about the exact model parameter sizes, and use standard models that are very small. Due to the use of standard base models, information on computational resources required to train them based on token count is readily available.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Upon reading the code of Ethics, the paper conforms to the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: This paper extends prior work on behavior cloning of individual behavior, but it is not the first to perform such fine-grained behavior cloning or observe their societal implications. Prior work by McIlroy-Young et al. discusses the implications of mimicking individual behavior with high fidelity (see "Mimetic Models: Ethical Implications of AI that Acts Like You" in AIES '2022).

    Guidelines:

- The answer NA means that there is no societal impact of the work performed.
- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Papers and codebases are properly cited.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.