ACM SIGGRAPH / Eurographics Symposium on Computer Animation 2024
M. Skouras and H. Wang
(Guest Editors)

COMPUTER GRAPHICS *forum*
Volume 43 (2024), Number 8

Learning to Move Like Professional Counter-Strike Players – Supplemental Material

D. Durst[1] F. Xie[2] V. Sarukkai[1] B. Shacklett[1] I. Frosio[3] C. Tessler[3] J. Kim[3] C. Taylor[2] G. Bernstein[4] S. Choudhury[5] P. Hanrahan[1] K. Fatahalian[1]

[1]Stanford University [2]Activision Blizzard [3]NVIDIA [4]University of Washington [5]Cornell University

This document includes

1. A complete list of the features in the CSKNOW dataset
2. A detailed description of DTWADE, our algorithm for labeling the derived strategy feature
3. A detailed description of the rule-based execution modules used by MLMOVE and RULEMOVE
4. A description of de_dust2's bombsites
5. Additional results including:
   a. More user study analysis
   b. A detailed description of how we compute offense flanking and defense spreading
   c. An explanation of how we compute EMD
   d. A model size abalation
   e. An evaluation of MLMOVE using traditional trajectory metrics
   f. An evaluation of our derived strategy feature accuracy

Additionally, the zip file contains videos and prompts from the user study.

1. Dataset

The dataset contains the following features. All "prior time steps" features extend 3s into the past at a 125ms frequency.

Overall Features

1. CS:GO demo file - string
2. game id - int
3. round id - int
4. tick id - int
5. bomb plant site - categorical
6. bomb fuse time - float

Per Player Features

1. player id - int
2. alive or dead - bool
3. team - bool
4. health - float
5. armor - float
6. helmet - bool
7. weapon id - int
8. walking - bool
9. crouching - bool
10. view angle at current time step - float
11. position at current and prior time steps - vector of floats
12. velocity at current time step and prior time steps - vector of floats
13. crosshair distance to nearest enemy (up to 30 degrees) at current and prior time steps - float
14. distance to nearest enemy and teammate at current and prior time steps - float
15. hurt in last 5s - float
16. fire in last 5s - float
17. enemy visible in last 5s (without considering FOV) - float
18. enemy visible in last 5s (considering FOV) - float
19. hurt in next tick - bool
20. kill in next tick - bool
21. killed in next tick - bool
22. fire weapon on current tick - bool
23. decrease distance to bomb over next 5s/10s/20s - bool
24. movement commands as described in paper

Per Grenade Throw Features

1. thrower id - int
2. grenade type - int
3. throw tick id - int
4. active tick id - int
5. expired tick id - int
6. destroy tick id - int
7. positions during throw trajectory - list of vector of floats

1.1. Derived Features

A limitation of the raw CS:GO log files is that they do not store audio signals and communication between teammates. Skilled CS:GO players make decisions based not only on their knowledge of the de_dust2 map (e.g., intuition about where enemies are likely to be), but also on information they receive while playing. Players track the following team interactions to know where enemies will be and which teammates need help: is a teammate about to shoot an enemy, is an enemy about to shoot a teammate, or was a teammate recently hurt by an enemy's shot? We enrich the CSKNOW dataset with auxiliary player features that can approximate this information.

We derive these auxiliary player features $d_{i,t} = [cd_{i,t}, ts_{i,t}, th_{i,t}, vi_{i,t}, ve_{i,t}, s_t]$ using game events and player

states. We identify when a player is about to shoot an enemy with $cd_{i,t} \in [0,1]$, the distance from a player's crosshair to their nearest enemy (up to 30 degrees). We track the time since they last shot (up to five seconds) with $ts_{i,t} \in [0,1]$, and the time since a player is last hurt (up to five seconds) with $th_{i,t} \in [0,1]$. We track the time since an enemy was last visible (up to five seconds) with $vi_{i,t} \in [0,1]$ and the time since a player might be seen by an enemy (up to five seconds) with $ve_{i,t} \in [0,1]$. We describe strategy s_t below.

Strategy Control. CSKNOW contains a wider set of long-term strategies than those employed in Retakes. All Retakes rounds are independent, so players always push for winning every round. The professionals in the CSKNOW dataset play a different game mode with inter-round dependencies. As a result, they sometimes employ a strategy of intentionally losing the current round to improve the odds of winning future rounds. We account for the strategy (s_t) mismatch by labeling each data point with whether the players are trying to win the round at that time with $s_t \in \{\text{Push}, \text{Save}\}$.

We create the strategy labels using the following steps.

1. A CS:GO expert manually labeled 323 of the 17216 rounds in the dataset (1.9%). Each round is labeled with a continuous value between 0.0 (save-only round) and 1.0 (push-only round), where intermediate values indicate the time of a switch from pushing to saving during the round. For example, 0.25 indicates a push round in the first 25% of its length. This transition is unidirectional. Players may flip from push to save. However, once they opt for saving, they do not have time to re-engage given the short length of Retakes rounds.
2. The remaining 98.1% of the data is labeled using a nearest-neighbor classifier: the aggressiveness of unlabeled rounds is determined by finding the most similar round that is labeled and in the train dataset. Computing similarity between two rounds is hard because player trajectories may end early (upon death) and there are many possible permutations of the mapping between players in two rounds. To address these challenges, we introduce Dynamic Time Warping Average Displacement Error (DTWADE) in Section 2. At a high level, DTWADE first searches along all possible player permutations to find the best match between the two rounds. Then it refines the estimate of the distance by applying Dynamic Time Warping (DTW) [SC78] to the players' trajectories to find a monotonic transformation of the time axis that minimizes the distance between the two sets of temporal sequences.
3. Once each round in the CSKNOW dataset has been labeled, we propagate these labels to the ticks in each round. All game ticks receive a binary s_t label based on their temporal position in the round. The per-tick labels are passed as conditioning input to the movement module of MLMOVE during its training.

Since humans always use a push strategy in Retakes, the 1430 test rounds in the evaluation all have a strategy round value of 1.0 (all ticks in the round are labeled Push). The eight rounds in the user study are drawn from these 1430 rounds.

2. Dynamic Time Warping Average Displacement Error

Metrics to evaluate the distance between two sets of N trajectories of length T, $r_i = \{x_{i,n}(t)\}_{n=0\ldots N-1}$ for $i = [0,1]$ and $t = [0,\ldots,T-1]$, have been proposed in the field of social behavior study; for instance, the Average Displacement Error (ADE) [PESV09; AGR*16] is the average displacement between the position of corresponding actors in the two sets:

$$\text{ADE}(r_0, r_1) = \frac{1}{NT} \sum_{n=0}^{N-1} \sum_{t=0}^{T-1} \left|\left| x_{0,n}(t) - x_{1,n}(t) \right|\right|_2 . \tag{1}$$

This measure has recently been extended to measure the distance between a ground truth set of trajectories and the ones created by a generative model (see the Joint Average Displacement Error, JADE [WHRK23]). Unfortunately, these metrics are unfit for our problem as (a) they assume the mapping between trajectories, players within the game, in the two sets to be known, and (b) they require all trajectories to be the same length T.

In CS:GO, players may be eliminated early (i.e., each trajectory $x_{i,j}$ has a different length $T_{i,j}$). In addition, the mapping between players in different rounds is unknown, therefore computing the distance between r_0 and r_1 requires testing multiple mapping permutations, which calls for a computationally efficient implementation of the metric.

We propose Dynamic Time Warping Average Displacement Error (DTWADE), to address these limitations when computing the distance between two rounds. First, for each round, all the trajectories are extended to the length of the longest trajectory in that round. Recall that a trajectory ends early when a player is eliminated. We assume an eliminated player remains at the same location until the round ends.

Determining the correct mapping between agents requires testing all of the different permutations. To make the computation more efficient, we sample only $K = 11$ equally-distanced points from the trajectory when determining the mapping between players $(0, 0.1, \ldots, 0.9, 1.0)$. In the case where the n-th trajectory of r_0 maps to the n-th trajectory of r_1, we can then define:

$$\text{KADE}(r_0, r_1) = \frac{1}{NK} \sum_{n=0}^{N-1} \sum_{k=0}^{K} \left|\left| x_{0,n}\left(\frac{kT_0}{K-1}\right) - x_{1,n}\left(\frac{kT_1}{K-1}\right) \right|\right|_2 . \tag{2}$$

The KADE is used to find the best mapping between players across two rounds. To do so, we denote the q-th permutation of the trajectories in r_1 by $\text{perm}(r, q)$ and compute the optimal permutation as

$$p(r_0, r_1) = \underset{q}{\text{argmin}}\ \text{KADE}(r_0, \text{perm}(r_1, q)) . \tag{3}$$

The metric above provides an approximate, yet efficient, way of finding the closest permutation. Once the correct permutation has been selected, we return the distance by stretching and comparing the trajectories through Dynamic Time Warping (DTW) [SC78], which finds a monotonic transformation that minimizes the distance between two temporal sequences:

$$DTW(r_0, p(r_0, r_1)) , \tag{4}$$

where $DTW(r_0, r_1)$ is the temporal alignment between r_0 and r_1 stretched accordingly to the DTW rules.

3. Rule-Based Execution Modules

We designed our bot using a rule structure known as a behavior tree (BT) [Isl05]. Our BT has three components. MLMOVE utilizes the aiming and firing logic of the components, as well as converting the learned movement model's last movement command to a keyboard movement action. RULEMOVE uses all of the components' logic.

Team Coordinator The highest-level component of the BT manages communication between members of each team. This component computes a team-wide probability distribution of enemy positions [Mor88; Isl13]. Just like humans, bots know enemies' approximate positions at the start of a round and exact positions when the enemies are visible. Their position probability is assumed to diffuse uniformly through all non-visible parts of the map at the rate of maximum running speed.

This component also handles long-term movement for RULEMOVE. It assigns each player a high-level path for attacking or defending an objective and an aggressiveness. For offense players, aggressiveness governs the order in which multiple bots travel the same path. For defense players, it governs how far the defenders position themselves from the objective, closer is safer but yields more territory to the offense.

Individual Player Planner The mid-level component implements the team coordinator's decisions for each player. First, it selects the aiming target for each player. When no enemies are visible, it uses the enemy probability distribution to select an aiming target where enemies are most likely to appear. When enemies are visible, it selects which one to aim at.

The component also handles non-learned movement for RULEMOVE. Based on teammates positions, it selects the next waypoint on the long-term path. The module plans an A*-path to the waypoint using the map's navigational mesh.

Individual Player Action Generator The low-level component generates mouse and keyboard commands for each player. It aims at a target in a human-like manner with a semi-implicit Euler method [BSK20]. If the player is aiming at a target, it generates distance-appropriate firing commands: shooting single shots at long distance and longer sprays during closer engagements. We account for recoil by exporting a recoil aim offset from the game engine. Finally, the action generator converts the A* path into keyboard commands.

For MLMOVE, this component also translates the learned movement model's movement command (produced once every 125 ms) into a keyboard movement action (produced once every 8 ms). The component uses the speed part of the movement command to determine if the keyboard movement action should include walking/crouching, specified by the control/shift keys. The component converts the absolute angular direction of the movement command to an angular direction relative to the player's current view angle, specified by the W/A/S/D keys. We find empirically that our absolute angular discretization made of 16 directions is sufficient to allow accurate in-game human trajectories, such as walking on thin ledges.

The component uses the jumping part of the movement command to determine if the keyboard movement action should include jumping, specified by the space bar.

4. de_dust2 Bombsites

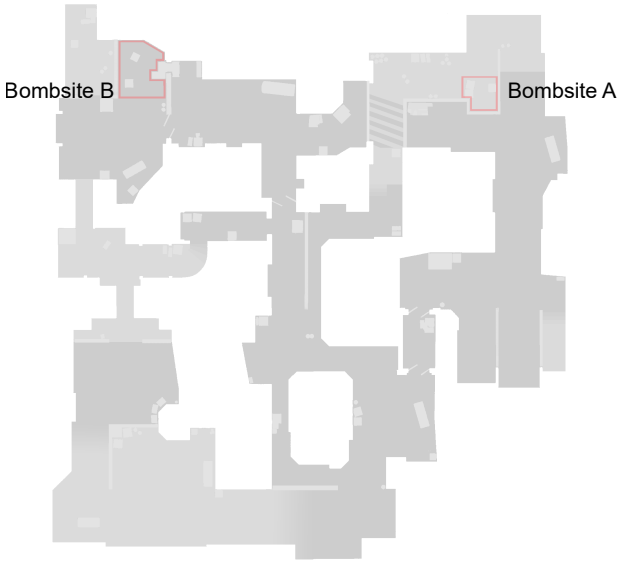

Figure 1: *The locations of Bombsites A and B in de_dust2. The bomb may be planted anywhere in the red outlines.*

Figure 1 shows the locations (in red) of the bombsites. The bomb may be planted anywhere in the red outlined regions. Players may adjust their strategies depending on the plant locations. Some locations require defenders to stand near the bomb, while others allow the defenders to watch the bomb from afar.

5. Additional Results

5.1. User Study

Our user study contains a diverse range of evaluators. Figure 2 shows the distribution, ranging from no experience to Global Elite.

We can use the TrueSkill ratings computed in the study to predict the results of one-on-one comparisons. Figure 3 shows the probability of one player configuration being ranked as more similar to human behavior than another configuration. MLMOVE has an 11% probability of being ranked more similar to human behavior than HUMAN.

5.2. Offense Flanking and Defense Spreading Details

Offense flanking is measured by recording instances when players occupy a configuration of positions simultaneously. For instance, when attacking BombsiteA, there are three attack vectors: from LongA, from CTSpawn, and from ShortStairs. Similarly, when attacking BombsiteB, the attack vectors are from BDoors, from Hole, and from UpperTunnels. We identify instances of flanking by noting the number of rounds with at least one tick where exactly two offensive players are alive and are positioned in two of the three different attack vectors for the target bombsite. This happens frequently in human data, occurring in 47% of the human rounds in the test dataset that start with two offense players alive. Figure 4

Table 1: *The teamwork configurations.*

Configuration	Team	Bombsite	Player 1 Position	Player 2 Position	Player 3 Position
F1	Offense	A	ShortStairs	LongA	N/A
F2	Offense	A	ShortStairs	CTSpawn	N/A
F3	Offense	A	CTSpawn	LongA	N/A
F4	Offense	B	BDoors	UpperTunnel	N/A
F5	Offense	B	Hole	UpperTunnel	N/A
S1	Defense	A	BombsiteA	LongA	ExtendedA
S2	Defense	A	BombsiteA	BombsiteA	LongA
S3	Defense	A	BombsiteA	ARamp	LongA
S4	Defense	B	BombsiteB	BDoors	UpperTunnel
S5	Defense	B	BombsiteB	BombsiteB	BDoors
S6	Defense	B	BombsiteB	BombsiteB	UpperTunnel

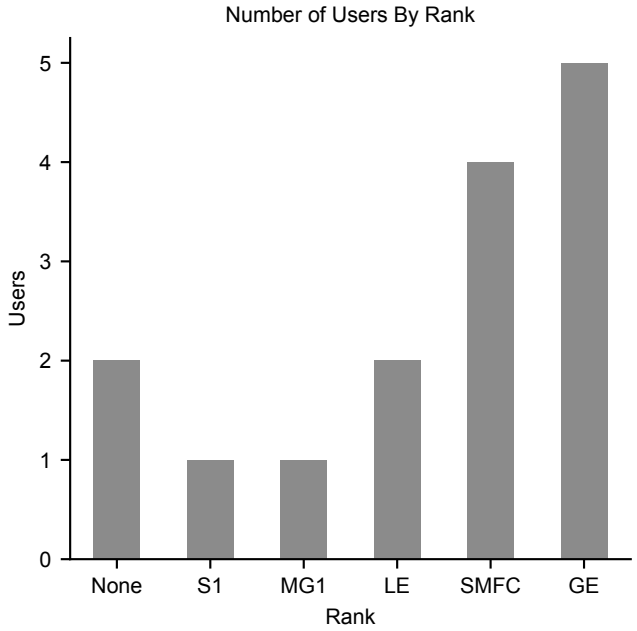

Figure 2: *The user study contains participants with a diverse range of CS:GO experience.*

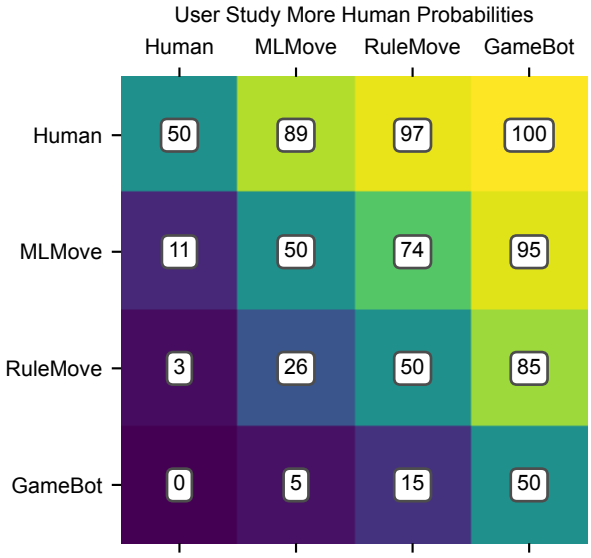

Figure 3: *TrueSkill ratings predict* MLMOVE *will be rated as more human than* HUMAN *11% of the time and than* RULEMOVE *74% of the time.*

shows the error in the frequency of offense flanking configurations relative to humans. For almost all configurations, MLMOVE most closely matches HUMAN frequency.

Defense spreading is measured in a similar manner. We have identified key locations where professional players spread out to counter flanks. When defending BombsiteA, the important defense positions are BombsiteA, LongA, and ExtendedA, which provide coverage of all offense attack vectors. Similarly, when defending BombsiteB, the key positions are BombsiteB, UpperTunnel, and BDoors. We identify instances of spreading by noting the number of rounds with at least one tick where exactly three defense players are alive and are positioned in key combinations of the important defense positions for the target bombsite. This happens frequently in human data, occurring in 45% of the human rounds in the test dataset that start with three defense players alive. Figure 4 shows the error in the frequency of defense spreading configurations relative to humans. For all configurations, MLMOVE most closely matches HUMAN frequency.

Table 1 shows details of which map positions are in each configuration. The de_dust2 file defines the position labels (like LongA).

5.3. EMD Calculation Details

We report the Earth Mover's Distance (EMD) between the distribution induced by the various bots through self-play and the ground truth human distribution. For EMD on two dimensional data, we downsample the map to a 120×120 grid (roughly 1 grid cell per

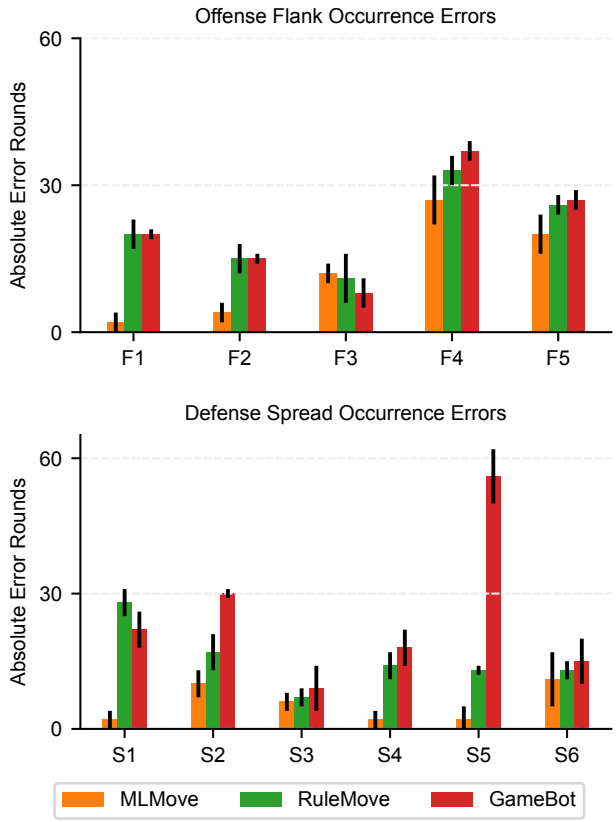

Figure 4: *Median and IQR absolute errors of the number of rounds relative to* HUMAN *where bots on offense assume specified flanking configurations (F1-F5, top) and bots on defense enter specified spreading configurations (S1-S6, bottom).* MLMOVE *reproduces all of the examined flanking and spreading configurations, and does so with a more similar frequency than* RULEMOVE *and* GAMEBOT.

player axis-aligned bounding box). The weights across these clusters are normalized to 1 to allow comparison between distributions with a varying number of points. We measure the EMD separately for offense and defense players, and report the combined values for both scenarios. For EMD on one dimensional data like lifetimes and shots per kill, we compute EMD directly on all of the data points.

5.4. Model Size Ablation

Table 2 shows that larger models run more slowly and moderately improve performance on Map Occupancy. We ablate the number of attention layers (l), the number of heads per layer (h), and the size of the MLPs after each attention layer (d). Transformer heads allow the model to learn multiple, independent representations by splitting the embedding space [VSP*17]. The default values in our paper are $l = 4$, $h = 1$, and $d = 2048$. Our results indicate that small models capture the coarse details, and larger models capture more fine-grained aspects of movement.

5.5. Trajectory Matching in Isolation

We evaluate the ability of MLMOVE to generate human-like trajectories in the absence of confounding factors like aiming, firing, and kills. To accomplish this, we spawn players in a simplified, simulated environment where (unlike CS:GO) we do not take into account the presence of walls, players can instantaneously accelerate to maximum velocity or fall to the floor, and there is no aiming or combat. For agents initially spawned in the same positions, we compare the trajectories generated by the MLMOVE movement module and by other baseline methods with the ground truth trajectories stored in the CSKNOW test set. These trajectories include save behavior and up to five players on a team.

We adopt standard metrics used in the automotive context [ECC*21; NVC*22; WHRK23]. These metrics quantify the capability of a probabilistic model to reproduce human trajectories over a short horizon of time by picking the best rollouts over multiple trials from the same starting positions. More specifically we use minJADE (Minimum Joint Average Displacement Error, i.e., the average L2 distance between the corresponding points of the human and best-predicted trajectory over k trials) and minJFDE (Minimum Joint Final Displacement Error, i.e., the L2 distance between the last point in the human trajectory and the best-predicted trajectory of k trials). We modify the metrics to account for early player deaths in CS:GO by taking each player's final alive position in minJFDE and only including time steps when each player is alive in minJADE.

All the metrics are reported in Hammer units used in the CS:GO engine; as a reference, CS:GO bots are 72 units in height [Com23] and their maximum speed is 250 units/second. We use $k = 6$ and split the trajectories into five seconds segments, matching with [NVC*22]. The resulting interval is long enough to allow players to enter a significant number of keystrokes, yet short enough to limit the effects of aiming, shooting, and killing on the trajectory. We perform auto-regressive rollouts for the five-second trajectories, using predictions at time step t as input at time step t+1. For all other input features, we use the ground truth values stored in the CSKNOW test dataset throughout the entire rollout.

We compare trajectories generated by the MLMOVE movement module (second row in Table 3) with those generated by several baselines. These include one we identify as *Ground Truth Command*, where we measure minJADE and minJFDE while executing the movement commands stored in the CSKNOW dataset in our simplified environment. This measures the error introduced by the assumptions adopted in our simplified environment; errors below this threshold should not be considered significant in this analysis. The other baselines are those typical of the related literature [CLS*19; ECC*21] and are:

- *Stand Still.* The bot remains in its initial position. This policy represents players moving around inside a small region (e.g., in the case of CS:GO, a defender in the bombsite).
- *Starting Command.* Here, the bot continuously executes the first ground truth movement command stored in the 5s sequence in the CSKNOW dataset. This behavior represents players moving consistently towards an objective, or with a high degree of latency/regressiveness.

Table 2: *Model size ablation. Larger models moderately improve Map Occupancy performance while significantly increasing inference latency.*

Metric	L4H1	L1H1	L1H4	L4H4	L16H1	L16H4	L4H1D256	L1H1D256
Map Occupancy EMD	8.2 ± 0.5	8.8	8.5	8.9	7.4	8.0	9.7	9.6
Kill Locations EMD	6.7 ± 0.1	7.3	6.1	6.3	6.3	6.8	6.2	5.8
Lifetimes EMD	4.9 ± 0.4	4.7	4.9	4.3	2.4	3.8	4.6	5.2
Shots Per Kill EMD	2.1 ± 0.1	2.1	2.4	2.5	1.5	1.7	2.1	2.3
Latency (ms)	6.9 ± 0.6	3.1	3.4	7.0	19.1	19.8	3.9	2.6
Model Parameters (M)	5.4	1.5	1.5	5.4	21	21	1.8	0.57
Parameter Size (MB)	21	5.7	5.7	21	81	81	6.7	2.2

Table 3: *Median ± IQR of MinJADE and minJFDE, measured in a simplified de_dust2 map while controlling the bots with several movement policies. The metrics of the* Ground Truth Command *policy indicate the error introduced by our simplifying assumptions.*

Simulation Type	MinJADE	minJFDE
Ground Truth Command	42 ± 22	61 ± 36
MLMove (ours)	117 ± 78	186 ±147
Stand Still	162 ±120	276 ±219
Starting Command	217 ±138	437 ±276
Nearest Neighbor	387 ±265	477 ±343

- *Nearest Neighbor.* We identify the most similar configuration of player positions in the CSKnow test dataset, and replicate the subsequent positions. This policy represents a bot controller that is purely based on a nearest-neighbor approach and lacks the capability of generating new trajectories.

Our simulator running the *Ground Truth Command* is sufficiently accurate to achieve a minJADE of 3.4% of the maximum distance that can be covered during a five-second rollout.

The metrics are reported in Table 3. We observe that MLMove outperforms the remaining baselines, suggesting that it generates trajectories with complexity that go beyond those of agents standing still or moving at constant speed and orientation. The metrics also highlight the nearest neighbor approach to be the worst for trajectory generation in this context.

5.6. Strategy Feature Accuracy

67 of the 323 manually labeled push/save rounds are in the test dataset. During DTWADE-based nearest-neighbor push/save labeling, we only check for similarity with manually labeled rounds in the train dataset. As a result, we can use the 67 test rounds to evaluate the quality of our labels. We label each tick in these rounds using two sources: the manual labels and nearest-neighbor labels. Figure 5 shows the result of this analysis. Nearest neighbor is more accurate at labeling ticks that have a manual save label than a push one. Nonetheless, there are a significant amount of ticks with a nearest neighbor push label that are manually labeled as saves. This may bias the model towards saving behavior manifested as movement predictions that lead to longer bot life times compared to human players.

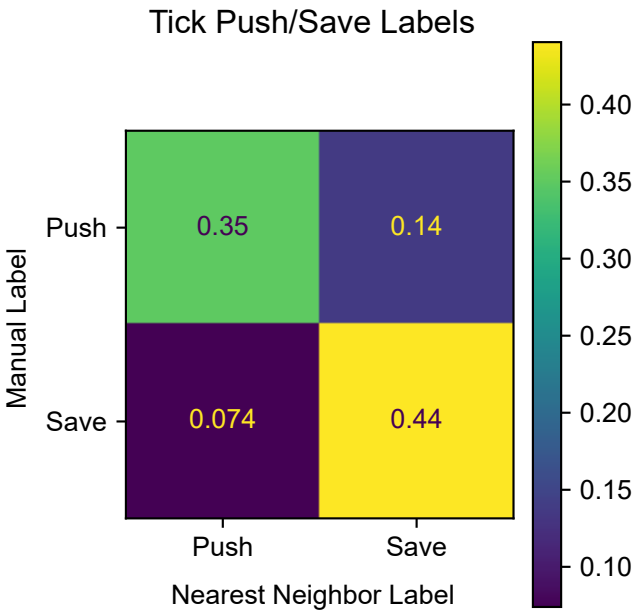

Figure 5: *The nearest-neighbor push/save labels are reasonably accurate.*

References

[AGR*16] Alahi, Alexandre, Goel, Kratarth, Ramanathan, Vignesh, et al. "Social lstm: Human trajectory prediction in crowded spaces". *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 961–971 2.

[BSK20] Bargteil, Adam W, Shinar, Tamar, and Kry, Paul G. "An introduction to physics-based animation". *SIGGRAPH Asia 2020 Courses*. 2020, 1–57 3.

[CLS*19] Chang, Ming-Fang, Lambert, John, Sangkloy, Patsorn, et al. "Argoverse: 3d tracking and forecasting with rich maps". *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, 8748–8757 5.

[Com23] Community, Valve Developer. *Counter-Strike: Global Offensive/Mapper's Reference - Valve Developer Community*. Sept. 2023. URL: https://developer.valvesoftware.com/wiki/Counter-Strike:_Global_Offensive/Mapper%27s_Reference (visited on 11/30/2023) 5.

[ECC*21] Ettinger, Scott, Cheng, Shuyang, Caine, Benjamin, et al. "Large scale interactive motion forecasting for autonomous driv-

ing: The waymo open motion dataset". *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, 9710–9719 5.

[Isl05] ISLA, DAMIAN. *GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI*. Mar. 2005. URL: https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai 3.

[Isl13] ISLA, DAMIÁN. "Third Eye Crime: Building a stealth game around occupancy maps". *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*. Vol. 9. 1. 2013, 206–206 3.

[Mor88] MORAVEC, HANS P. "Sensor fusion in certainty grids for mobile robots". *AI magazine* 9.2 (1988), 61–61 3.

[NVC*22] NGIAM, JIQUAN, VASUDEVAN, VIJAY, CAINE, BENJAMIN, et al. "Scene transformer: A unified architecture for predicting future trajectories of multiple agents". *International Conference on Learning Representations*. 2022 5.

[PESV09] PELLEGRINI, STEFANO, ESS, ANDREAS, SCHINDLER, KONRAD, and VAN GOOL, LUC. "You'll never walk alone: Modeling social behavior for multi-target tracking". *2009 IEEE 12th international conference on computer vision*. IEEE. 2009, 261–268 2.

[SC78] SAKOE, HIROAKI and CHIBA, SEIBI. "Dynamic programming algorithm optimization for spoken word recognition". *IEEE transactions on acoustics, speech, and signal processing* 26.1 (1978), 43–49 2.

[VSP*17] VASWANI, ASHISH, SHAZEER, NOAM, PARMAR, NIKI, et al. "Attention is all you need". *Advances in neural information processing systems* 30 (2017) 5.

[WHRK23] WENG, ERICA, HOSHINO, HANA, RAMANAN, DEVA, and KITANI, KRIS. "Joint Metrics Matter: A Better Standard for Trajectory Forecasting". *arXiv preprint arXiv:2305.06292* (2023) 2, 5.