AI server-side prediction for latency mitigation and cheating detection: the MPAI-SPG approach

AI Server-Side Prediction for Latency Mitigation and Cheating Detection: The MPAI-SPG Approach

Daniele Spina
Politecnico di Torino
Turin, Italy
daniele.spina@polito.it

Andrea Bottino
Politecnico di Torino
Turin, Italy
andrea.bottino@polito.it

Davide Cavagnino
Università di Torino
Turin, Italy
davide.cavagnino@unito.it

Leonardo Chiariglione
MPAI
leonardo@chiariglione.org

Maurizio Lucenteforte
Università di Torino
Turin, Italy
maurizio.lucenteforte@unito.it

Marco Mazzaglia
Politecnico di Torino
Turin, Italy
marco.mazzaglia@polito.it

Francesco Strada
Politecnico di Torino
Turin, Italy
francesco.strada@polito.it

***Abstract*—This work introduces MPAI-SPG, a novel approach to mitigate latency and detect cheating in online gaming through server-based prediction. The paper reports the implementation of this approach in an online racing game. Using machine learning, we developed a prediction module trained on a custom-built dataset, which is publicly available. Experimental sessions with real players were conducted to assess prediction accuracy and the overall solution's effectiveness in ensuring smooth multiplayer gaming despite data absence. The results demonstrate MPAI-SPG's potential to enhance the gaming experience amidst network challenges. The approach allows for continuous improvement in prediction accuracy, leveraging new training techniques.**

***Index Terms*—online multiplayer gaming; latency; cheating; machine learning**

I. Introduction

The appeal of online multiplayer games has propelled them to the top of the global entertainment industry [1]. Among the prevailing network architectures, "authoritative servers" stand out as cornerstones due to their focus on maintaining consistency between all connected clients (i.e., players) [2]. In this architecture, the server acts as the central arbiter of the game state (GS). By processing data received from all players, the server computes a new GS, which is then distributed to all clients. This model ensures that game progress is consistent across all clients and that the integrity of the gameplay is maintained.

Despite the widespread adoption of authoritative server architectures, they are not immune to the challenges posed by common network issues. One particularly critical concern is latency, i.e., the delay in data transmission between a player's device and the game server. Latency can disrupt the seamless flow of gameplay, introducing inconsistencies that can adversely affect the gaming experience.

Within this context, latency yields two main consequences. First, for the player experiencing latency, it manifests as a disruption in responsiveness, requiring them to wait for server responses to update their local GS, which leads to perceptible delays, disrupting the smooth experience of gameplay. Second, as the server receives delayed client data (CD), the GS on the server becomes inconsistent, compromising the game experience for the clients unaffected by latency.

Various techniques have been developed to overcome these challenges. A widely used approach to solve the first problem is client prediction [3]. With this approach, the client makes informed predictions about future GSs, drawing from historical data, the current context, and the user inputs. This allows the client to display the next GSs without waiting for the server version, resulting in a more responsive gaming experience. When the server GS arrives, the client must then reconcile its state with the one received, resolving any discrepancies to ensure synchronization among all clients and maintain a consistent and fair game progression. Although this method can reduce latency perceived by the player, it also opens up more opportunities for cheating, as the client has greater control over the game state it reports. This requires the server to implement mechanisms to validate or correct the states reported by the client.

Despite the effectiveness of these techniques in improving game responsiveness on clients affected by latency, the unaffected ones will receive an inconsistent GS from the server due to missing data from one client. To tackle this issue, an acknowledged method is Time Delay [4]. The Time Delay technique buffers GS updates to synchronize all clients, fostering a more uniform gaming experience. While Time Delay has demonstrated its efficacy in eliminating state inconsistencies [5], [6], it is also acknowledged that this approach can result in decreased responsiveness [7].

In this study, our objective aligns with the same problem Time Delay seeks to address: providing an optimal gaming experience for clients unaffected by latency while striving to overcome the reported responsiveness issues associated with this technique. We propose a novel methodology that involves implementing prediction techniques at the server level. When latency-affected clients are detected, the server takes over, using a predictive model to accurately forecast player actions based on historical data and the current context. These predic-

tions are then shared with all clients, ensuring a continuous and unified gaming experience, even when real-time data is delayed or absent. Our prediction approach leverages machine learning (ML) algorithms, an area of research only recently being explored in the context of latency mitigation strategies for online multiplayer games [1]. Furthermore, by comparing predictions with CDs, the server could also identify possible cheating attempts, especially when clients have greater control over their local instances (i.e., client prediction). The proposed method takes the name of Server-based Predictive multiplayer Gaming (SPG) and has been conceptualized by the authors within the Moving Picture, Audio and Data Coding by Artificial Intelligence (MPAI) organization, a non-profit group dedicated to developing standards for AI-based data coding. Thus, in the remainder of this work, we will refer to the approach as MPAI-SPG.

To evaluate the feasibility of our method, we implemented MPAI-SPG in an online car racing game. In the implementation, the predictions were managed by a neural network (NN) trained on a dataset curated and published for this study. We conducted a user study involving 12 volunteers to assess the impact of our predictive model on players' experiences during actual gaming sessions. The results suggest that MPAI-SPG has the potential to be a viable alternative solution to help mitigate problems associated with network latency in online multiplayer games.

II. State of the Art

Server-side prediction in online gaming mostly aims to reduce packet exchange [1], and leverages techniques like Dead Reckoning [8] to forecast player movements. While effective in certain scenarios, Dead Reckoning struggles with sudden player movements, limiting its utility in predicting complex behaviors [9], [1], [8], [10]. To address these challenges, recent works [11], [12] have explored ML solutions, demonstrating the potential of ML to enhance prediction accuracy beyond traditional methods. Similarly, [13], [14], [15] have applied NN and deep reinforcement learning to predict player actions in real-time strategy and first-person shooter games, underscoring ML's adaptability to varied gaming contexts.

Despite these advancements, the application of ML in server-side prediction remains scarce, as highlighted by a recent comprehensive survey [1]. Our work builds upon this foundation, proposing a novel approach that utilizes ML not for reducing packet exchange but for mitigating latency by predicting future GSs. To the best of our knowledge, this study represents the first instance of deploying a prediction system on the server to manage players' actions in the absence of CD due to network issues, marking a significant expansion of ML's application in enhancing online multiplayer gaming experiences.

III. Server Prediction

In this section, we describe in detail the architecture of the MPAI-SPG system and its implementation in a car racing game.

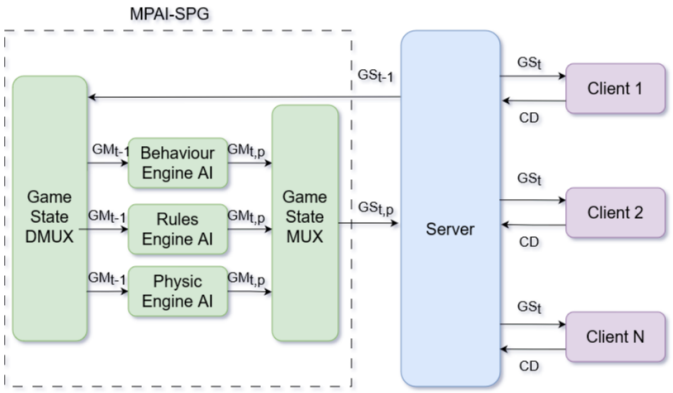

Fig. 1. MPAI-SPG architecture

A. MPAI-SPG Architecture

MPAI-SPG is envisioned as an external software module that directly communicates with the server game logic. Fig. 1 visually represents the MPAI-SPG architecture connected to a server block. As mentioned in the introduction, MPAI-SPG is intended to work in an authoritative server context where, at every time step the server evaluates a new GS based on all the CD received from the remotely connected players. Once computed, the new GS is then sent back to each client instance.

Within the MPAI-SPG framework, the GS is composed of a series of game messages (GM), which are output from three principal engines: (i) the *Behaviour Engine*, orchestrating actions from players and non-player entities; (ii) the *Rules Engine*, ensuring adherence to game mechanics; and (iii) the *Physics Engine*, responsible for physical interactions within the game environment.

The process begins with the current GS (GS_{t-1}) being fed into the MPAI-SPG's *Game State Demultiplexer (GS-DMUX)*, which deconstructs it into discrete GMs (GM_{t-1}). Each GM is then processed by its respective Engine AI, leveraging a trained NN to produce a predicted GM ($GM_{t,p}$). These predictions are aggregated by the *Game State Multiplexer (GS-MUX)*, forming the predicted GS ($GS_{t,p}$), which is then communicated back to the server for the next iteration of GS evaluation.

During the MPAI-SPG module operations, the server independently computes its updated GS (GS_t), which is solely derived from the available CDs. The server utilizes the $GS_{t,p}$ from MPAI-SPG in two scenarios. Firstly, if any CD is missing (e.g., if a client is experiencing latency), the server uses the predicted state to provide all other clients with a continuous gaming experience, compensating for the data shortfall from one or more clients. Secondly, if $GS_{t,p}$ and GS_t are significantly different, the server could detect a cheating attempt deriving from manipulated CDs.

In conclusion, MPAI-SPG functions as an auxiliary arbiter in tandem with the server. It mirrors the server's structure and logic, yet diverges by employing NNs to predict GS outcomes, relying solely on historical GS data rather than processing the

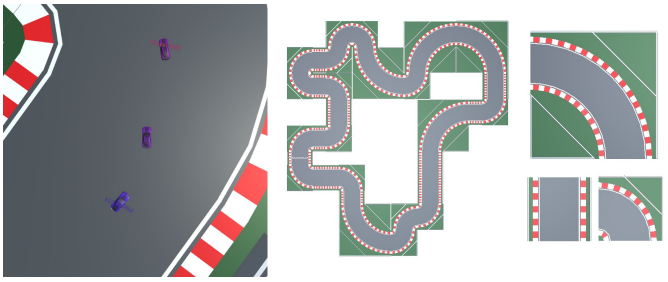

Fig. 2. The Multiplayer 3D Racing Game (left), the racing track (center) and the 3D modular tiles (right) that can be rotated in 90 degrees steps

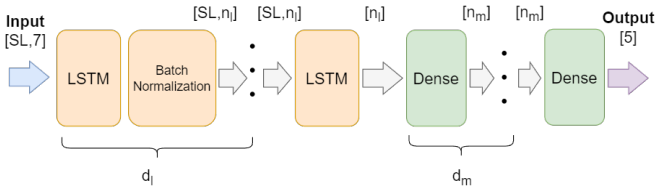

Fig. 3. Neural network architecture

current CDs.

B. The Car Racing Game

To assess the effectiveness of the MPAI-SPG architecture, we developed a proof-of-concept in the form of a 3D multiplayer car racing video game, where players can race on a series of tracks, which are modularly constructed from a series of predefined 3D tiles (Fig. 2). The player's input is captured from a keyboard and controls the vehicle's acceleration, brakes, and steering, ultimately updating its *Spatial Attitude* (SA), a term we use to encompass the vehicle's position, rotation, and velocity.

Each client runs a local game instance, which sends as CD to the server, the SA of the controlled car. The server then processes the CDs from all connected clients and computes GS_t, essentially an aggregate of the SAs of all vehicles. To implement MPAI-SPG in our game, we needed to develop the *Behaviour Engine AI*, as the focus was on predicting the behaviour data of the cars, particularly their SA. In this context, GM_{t-1}, one for each car, is sent to the Engine AI and comprises two layers of information: the environment surrounding the car and its SA. The environmental data provided includes: (i) the type of tile the car is on, (ii) the tile's ranking relative to the other tiles of the track (e.g. first, fourth or last tile of the track), and (iii) the position of the car relative to the center of the tile. The engine outputs a series of $GM_{t,p}$ for each car containing the predicted SA. These data are then combined to create $GS_{t,p}$. Further details on the NN implementation for the *Behaviour Engine AI* are discussed in the following section.

The game was developed using the Unity game engine, and the networking features were implemented through the open-source game networking library Mirror.

IV. Engine AI Model & Training

MPAI-SPG's *Behavioral Engine AI* is designed to evaluate the consequences of player actions, thus outputting the car's SA. The predictions are based on a temporal series of previous GS. After a review of state-of-the-art techniques for timeseries prediction [16], we decided to use a deep Long Short-Term Memory (LSTM) network. The architecture of this network is shown in Fig. 3.

The model consists of a deep LSTM network connected to a multilayer perceptron (MLP). In the network, each LSTM is followed by a Batch Normalization layer, except the last one. The total number of LSTMs is $d_l + 1$. The hidden state of the last LSTM block (with a size of n_l) is sent to the MLP, which consists of $d_m + 1$ Fully Connected (FC) layers. Each FC has an input and output size of n_m, except for the last layer whose output is 5.

The input for the network is a matrix $R^{SL \times 7}$, where SL stands for the length of the time sequence. The seven features include the four environmental data previously described, the car's velocity, and the car's rotation. Position and velocity data are given as pairs of (x, z) values, as the y-axis does not vary for these properties, and car's rotation is only around the y-axis. The output is the predicted global car position, its rotation and velocity expressed as an array in R^5. Therefore, the output does not include environmental data.

When implemented in the game, due to computational constraints, the prediction time of the AI model exceeded the duration of the server time step. Therefore, we defined a parameter called Discard (*D*), which controls the working interval of SPG. Every *D* seconds, SPG evaluates the prediction ($GS_{t+D,p}$), and updates the input matrix, resulting in a sampling interval of the input time series equal to D.

A. Dataset

Due to the lack of available dataset for the training, we generated a synthetic one by simulating game sessions with autonomous agents using the ML-Agents toolkit [17], a ML framework for Unity. This approach overcame the impracticality of collecting data through extensive gameplay sessions.

We trained autonomous agents using Curriculum Learning on tracks of increasing complexity, applying penalties for collisions with track walls or other players and for incorrect checkpoint passages. A checkpoint refers to a marked location on the track, arranged in an ordered sequence, which cars must pass through to ensure they are following the correct path. Rewards were granted for completing laps and correctly navigating through checkpoints. To emulate a variety of real-world driving styles, we trained four distinct agent types, each governed by a unique set of rewards and punishments. These specific rewards encouraged acceleration and optimal alignment to the next checkpoint, while punishments were applied for braking.

To collect driving data, we let the agents race for multiple games, over the same track. Each game is composed of three laps, during which we sampled the car's SA information and surroundings every 0.02 seconds. At the end of this process,

TABLE I
TRAINED MODELS

ID	D_l	N_l	D_m	N_m	SL	Val MAE	Epochs
1	3	64	3	64	20	0.579	100
2	3	256	3	64	20	0.544	35
3	1	256	0	0	40	0.569	87
4	3	256	3	64	40	0.453	100

we collected 2 million records[1]. We divided the dataset into three: train, validation and test set. The train set contains half of the initial dataset, while test and validation sets 25% each.

B. Implementation details

In the following, we describe the parameters used in the training of each model. The loss was evaluated by computing the Mean Square Error between the real car's SA, and the predicted one. The LSTM and MLP layers were initialized with the default values from the TensorFlow library. We used the Adam algorithm for the optimizer, with an initial learning rate (lr) of 0.001, which we halved every time we reached a plateau on the validation loss. We set the batch size to 512 samples. We trained the models for 100 epochs with early stopping. After completing the training, the model parameters that showed the lowest validation loss were saved.

Based on the number of layers, units and the sequence length, the time needed for completing a single epoch varies, but the mean time is about 20 minutes. The training was done on a machine with a CPU AMD Ryzen™ 7 5800H, 16 GB of RAM and the integrated GPU AMD Radeon™ Graphics.

C. Results

In this section, we outline the results yielded by the best networks during the training process and how they performed when deployed in the game scenario.

1) Training Results: We trained various networks while changing the d_l, n_l, d_m, n_m, SL and D parameters. We tested values ranging from shallow networks (i.e. 0 depth) to 5 for the d_l and d_m. For n_l and n_m, we started from 32 units and reached 512. For SL the lowest value implemented was 10, while the highest was 100. Lastly, we tried 0.1 and 0.2 seconds for the D parameter. Table I reports the configurations and results of the four best models trained. The Table assigns to each model a unique ID, and throughout the article, we refer to them by their respective IDs. Since the models with D equal to 0.1 were not among the four best, in Table I, all the configurations share the same D value of 0.2 seconds. By comparing the minimum Mean Absolute Error (MAE) achieved by the models on the validation set, the ones with higher sequence length generally demonstrated better performance. Additionally, Models 4 and 2 share the same configuration apart from the SL, but Model 2 overfitted, stopping the training earlier. Model 4 was able to achieve the lowest error between all models.

[1]The dataset is publicly available at the following repository: https://github.com/CGVGroup/mpai-spg-ai-training

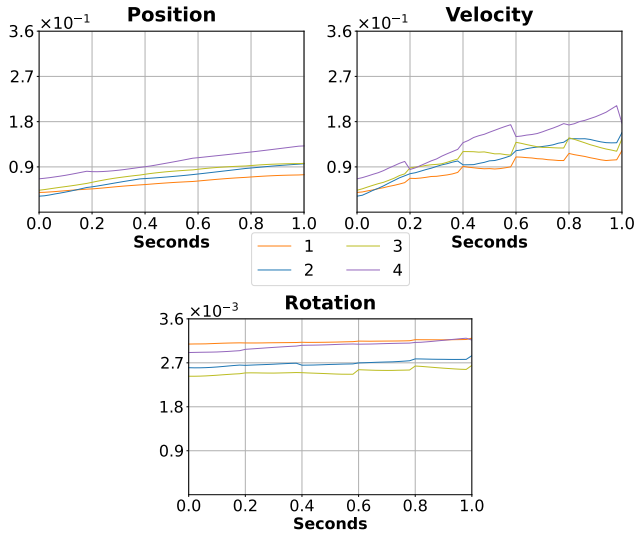

Fig. 4. Comparison Normalized MAE for Velocity, Rotation and Position

2) Game testing: In a real-case scenario, the server may need to apply MPAI-SPG predictions for several consecutive times. Whenever predictions are applied, they serve as input for the subsequent MPAI-SPG predictions, thus propagating the error. We conducted a new experiment with the previously trained models to evaluate how well they perform in those cases. We used the AI car drivers to play new games, during which we apply SPG predictions for one consecutive second on a copy of the AI car driver. By comparing the two cars, the real and the predicted one, we evaluated the MAE for position, velocity and rotation in correlation with the time of SPG activation. Fig. 4 displays the results of this experiment. We normalized the MAEs reported by dividing them by their highest possible prediction error, which correspond to the maximum change in the position, velocity and rotation obtainable by the car in 0.2 seconds.

Model 4, the model with the lowest validation MAE during training, achieved the worst overall prediction quality between all models. Contrary to our anticipation, models with a sequence length of 20 emerged as the top performers. Between models 2 and 1, the former demonstrated higher quality in the initial prediction (0 seconds), whereas Model 1, starting from the second prediction (0.2 seconds), showed a lower error in velocity predictions. Despite Model 1 having the highest error in rotation, the impact is mitigated by the lower magnitude of the rotation error. In fact, Model 1 attains the lowest error on the position evaluation.

Based on these considerations, we decided to implement Model 1 for the SPG's prediction system used in the user tests described in the following section.

V. USER TESTING

We conducted a user testing session with two main objectives: (i) assess the accuracy of MPAI-SPG in predicting human player behaviours and (ii) determine the extent to which participants could discern the influence of MPAI-SPG during

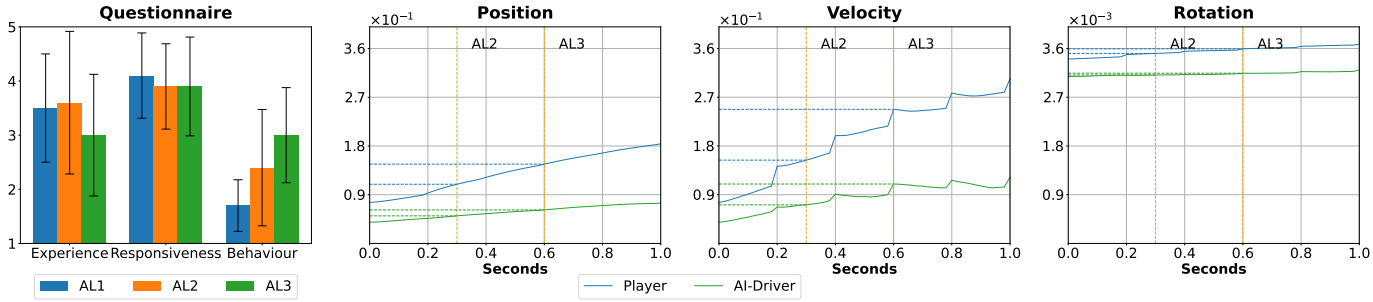

Fig. 5. Questionnaire responses (standard deviations are reported through error bars) and normalized MEA achieved on Human Players

TABLE II
ACTIVATION LEVELS

Level	Frequency (s)	Length (s)
AL1	None	None
AL2	10 ± 2	0.3
AL3	8 ± 2	0.6

gameplay. This section outlines the experimental protocol, the data collected, and an analysis of the results.

A. Protocol & Collected Data

The experimental setup involved a single workstation running a server and four online car racing game clients, three controlled by AI drivers and one by the participant. To simulate the effects of MPAI-SPG (i.e., we did not test under real latency conditions), we developed an activation module in charge of dispatching consecutive $GS_{t,p}$ to participants' instances. This module is controlled by two parameters: *length*, the number of consecutive seconds where $GS_{t,p}$ is sent, and *frequency*, the distance in seconds between activations. Different values of these parameters establish three Activation Levels (ALs), summarized in Table II.

We recruited 12 participants who were initially given the opportunity to race solo to familiarize themselves with the game's controls and mechanics. Then, they completed two race consisting of two laps around the track depicted in Fig. 2 for each AL. The sequence began with AL1, followed by the other ALs in a random order, ensuring experimental balance. After each race, participants filled out a questionnaire consisting of: (i) one item for the overall gaming *Experience*, (ii) one item for the perceived *Responsiveness* and (iii) three items to investigate how often the players perceived an odd *Behaviour* on the others' cars in relation to position, velocity and direction. Responses were recorded on a 5-point Likert Scale, where a score of 1 reflected a negative experience for the first two items and indicated the frequency of perceived anomalies for the remaining. Statistical differences in questionnaire responses between ALs were computed via a one-way ANOVA followed by Tukey post hoc test. Additionally, to evaluate the prediction accuracy of real players, during the initial solo race, we secretly conducted predictions on the player's car for periods of 1 second while simultaneously recording the actual and predicted car data.

B. Results

In our assessment of the prediction model's accuracy, we measured the normalized MAE in position, velocity, and rotation between the human-driven car and its predicted counterpart. The normalization was evaluated as for the previous experiment. These metrics, presented in Fig. 5, are also compared with those achieved previously on the AI-driven car. It's evident from the results that the prediction errors for the human player are higher than for the AI driver, particularly regarding velocity, which is inherently the most challenging aspect to predict due to its rapid variability in a racing game context. Despite these disparities, the outcomes were within our expectations since the predictive NN was trained exclusively on data from AI-controlled cars. The relatively small differences, especially in the position and rotation parameters, suggest that employing synthetic data for training is a viable strategy, though it is not without its imperfections. This insight is crucial, as it underscores our approach's potential while highlighting areas for improvement. The graphs indicate a need for the MAE to be reduced across all parameters and for human and AI-driving prediction curves to align more closely.

Fig. 5 shows the aggregate results from the questionnaires. In the metrics of *Experience* and *Responsiveness*, values are moderately high, above 3, and the differences across ALs are minimal, indicating that the integration of predictions (AL2 and AL3) did not significantly impair the players' gaming perception. However, the ANOVA test did not reveal statistical significance between ALs for both scales ($p = 0.26$ in *Experience* and $p = 0.68$ in *Responsiveness*). A more noticeable discrepancy emerged in the ratings for odd *Behavior* (higher values indicate worst results), where players observed more unusual movements from the AI-driven cars under the prediction conditions, likely due to occasional prediction inaccuracies. This was most evident where predictions were longer and more frequent as the AL3 condition averaged a 3 out of 5 score, significantly differing from AL1 ($p < 0.05$ in Tuckey's post hoc test). However, the maximum average rating was 3, categorized as 'Sometimes', indicating that despite some inaccuracies, the overall performance of the MPAI-SPG model retains a level of robustness. This suggests its potential effectiveness in scenarios with increased latency, where it can compensate for missing player data. Moreover, for the

AL1 condition, where predictions were not employed, we encountered some unanticipated findings. The overall gaming experience received a marginally lower rating then AL2, which might be attributed to players still acclimatizing to the game during their first session despite a trial race. Furthermore, in this same condition, participants reported occasional odd behaviour from the AI drivers, which could reflect limitations in the AI's driving logic, resulting in unanticipated maneuvers. These insights highlight the crucial role of advanced and well-trained driving agents, which in future iterations of the study will help improve the compiling of a more precise dataset and user testing involving AI agents, to overcome shortages in human players.

VI. Limitation And Future Works

The limitations of our study primarily revolve around the imperfect prediction accuracy of the MPAI-SPG module, both when applied to AI drivers and to human players. Both issues could be addressed in future iterations by improving the dataset with data from more sophisticated AI-drivers, which better mimic human behaviour, or with data collected from real players gaming sessions. Moreover, alternative approaches to LSTM, like diffusion models, should be explored to verify their possible better efficacy in predicting the nuanced dynamics of human gameplay. Additionally, the simulated testing environment did not incorporate actual latency, rather, it simulated the MPAI-SPG countermeasures. Future research will aim to test the approach under real latency conditions, further validating its applicability in live gaming scenarios. Furthermore, despite the implemented architecture also aimed at detecting cheating attempts, due to the imperfect prediction accuracy, this feature could not be tested during the user study and will be of utter relevance in future tests. Finally, due to the constrained user sample size, we acknowledge the limitation in statistical power to detect significant differences. In future iterations of this research, a larger sample size will be considered to enhance the robustness of our findings and potentially reveal more significant insights.

VII. Conclusions

In this work, we introduced MPAI-SPG, a novel approach that has the potential to mitigate latency effects and detect cheating in online gaming through server-based predictive modeling. To verify the feasibility of the solution, we implemented it within an online car racing game and developed the prediction modules based on NN training, performed on a dataset custom-built for this study and open-sourced to the research community. We conducted an experimental session with 12 real players, to verify the accuracy of the predictions and the overall efficacy of the solution in providing a smooth multiplayer gaming experience. Despite the predictions' accuracy could be improved, the results hint at the possibilities of this approach to enhance the gaming experience by providing seamless gameplay despite network challenges.

VIII. Acknowledgements

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE00000001 - program "RESTART").

References

[1] S. Liu, X. Xu, and M. Claypool, "A survey and taxonomy of latency compensation techniques for network computer games," *ACM Computing Surveys (CSUR)*, vol. 54, no. 11s, pp. 1–34, 2022.

[2] G. Gambetta, "Fast-paced multiplayer," 2001, https://gabrielgambetta.com/client-server-game-architecture.html.

[3] S. Aggarwal, H. Banavar, A. Khandelwal, S. Mukherjee, and S. Rangarajan, "Accuracy in dead-reckoning based distributed multi-player games," in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games*, 2004, pp. 161–165.

[4] M. Mauve, "Consistency in replicated continuous interactive media," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, 2000, pp. 181–190.

[5] C. Savery, T. N. Graham, and C. Gutwin, "The human factors of consistency maintenance in multiplayer computer games," in *Proceedings of the 2010 ACM International Conference on Supporting Group Work*, 2010, pp. 187–196.

[6] J. Xu and B. W. Wah, "Concealing network delays in delay-sensitive online interactive games based on just-noticeable differences," in *2013 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2013, pp. 1–6.

[7] D. Liang and P. Boustead, "Using local lag and timewarp to improve performance for real life multi-player online games," in *Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games*, 2006, pp. 37–es.

[8] Y. Chen and E. S. Liu, "Comparing dead reckoning algorithms for distributed car simulations," in *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2018, pp. 105–111.

[9] V. Y. Kharitonov, "Motion-aware adaptive dead reckoning algorithm for collaborative virtual environments," in *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and its Applications in Industry*, 2012, pp. 255–261.

[10] L. F. K. de Almeida and A. S. Felinto, "Evaluation of the motion-aware adaptive dead reckoning technique under different network latencies applied in multiplayer games," in *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 2018, pp. 137–13 709.

[11] E. P. Duarte Jr, A. T. Pozo, and P. Beltrani, "Smart reckoning: Reducing the traffic of online multiplayer games using machine learning for movement prediction," *Entertainment Computing*, vol. 33, p. 100336, 2020.

[12] S. Akama, T. Motoo, T. Ishioka, T. Fujihashi, S. Saruwatari, and T. Watanabe, "Deep reinforcement learning model design and transmission for network delay compensation in 3d online shooting game," in *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*. IEEE, 2023, pp. 200–205.

[13] D. Halbhuber, N. Henze, and V. Schwind, "Increasing player performance and game experience in high latency systems," *Proceedings of the ACM on Human-Computer Interaction*, vol. 5, no. CHI PLAY, pp. 1–20, 2021.

[14] D. Halbhuber, M. Seewald, F. Schiller, M. Götz, J. Fehle, and N. Henze, "Using artificial neural networks to compensate negative effects of latency in commercial real-time strategy games," in *Proceedings of Mensch und Computer 2022*, 2022, pp. 182–191.

[15] T. Motoo, J. Kawasaki, T. Fujihashi, S. Saruwatari, and T. Watanabe, "Client-side network delay compensation for online shooting games," *IEEE Access*, vol. 9, pp. 125 678–125 690, 2021.

[16] Z. Han, J. Zhao, H. Leung, K. F. Ma, and W. Wang, "A review of deep learning models for time series prediction," *IEEE Sensors Journal*, vol. 21, no. 6, pp. 7833–7848, 2019.

[17] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *arXiv preprint arXiv:1809.02627*, 2020. [Online]. Available: https://arxiv.org/pdf/1809.02627.pdf