Moving Picture, Audio and Data Coding
by Artificial Intelligence
www.mpai.community

MPAI Technical Report

Server-based Predictive Multiplayer Gaming (MPAI-SPG) - Mitigation of Data Loss effects (SPG-MDL)

V1.0

Technical Report: Server-based Predictive Multiplayer Gaming (MPAI-SPG) - Mitigation of Data Loss effects (SPG-MDL) V1.0

Foreword

The international, unaffiliated, non-profit ***Moving Picture, Audio, and Data Coding by Artificial Intelligence (MPAI)*** organisation was established in September 2020 in the context of:

1. **Increasing** use of Artificial Intelligence (AI) technologies applied to a broad range of domains affecting millions of people
2. **Marginal** reliance on standards in the development of those AI applications
3. **Unprecedented** impact exerted by standards on the digital media industry affecting billions of people

believing that AI-based data coding standards will have a similar positive impact on the Information and Communication Technology industry.

The design principles of the MPAI organisation as established by the MPAI Statutes are the development of AI-based Data Coding standards in pursuit of the following policies:

1. Publish upfront clear Intellectual Property Rights licensing frameworks.
2. Adhere to a rigorous standard development process.
3. Be friendly to the AI context but, to the extent possible, remain agnostic to the technology thus allowing developers freedom in the selection of the more appropriate – AI or Data Processing – technologies for their needs.
4. Be attractive to different industries, end users, and regulators.
5. Address five standardisation areas:
    1. *Data Type*, a particular type of Data, e.g., Audio, Visual, Object, Scenes, and Descriptors with as clear semantics as possible.
    2. *Qualifier*, specialised Metadata conveying information on Sub-Types, Formats, and Attributes of a Data Type.
    3. *AI Module* (AIM), processing elements with identified functions and input/output Data Types.
    4. *AI Workflow* (AIW), MPAI-specified configurations of AIMs with identified functions and input/output Data Types.
    5. *AI Framework* (AIF), an environment enabling dynamic configuration, initialisation, execution, and control of AIWs.
6. Provide appropriate Governance of the ecosystem created by MPAI Technical Specifications enabling users to:
    1. *Operate* Reference Software Implementations of MPAI Technical Specifications provided together with Reference Software Specifications
    2. *Test* the conformance of an implementation with a Technical Specification using the Conformance Testing Specification.
    3. *Assess* the performance of an implementation of a Technical Specification using the Performance Assessment Specification.
    4. *Obtain* conforming implementations possibly with a performance assessment report from a trusted source through the MPAI Store.

Today, the MPAI organisation operated on four solid pillars:

1. The MPAI Patent Policy specifies the MPAI standard development process and the Framework Licence development guidelines.
2. *Technical Specification: Artificial Intelligence Framework (MPAI-AIF)* V2.1 specifies an environment enabling initialisation, dynamic configuration, and control of AIWs in the standard AI Framework environment depicted in Figure 1. An AI Framework can execute AI applications called AI Workflows (AIW) typically including interconnected AI Modules (AIM). MPAI-AIF supports small- and large-scale high-performance components and promotes solutions with improved explainability.

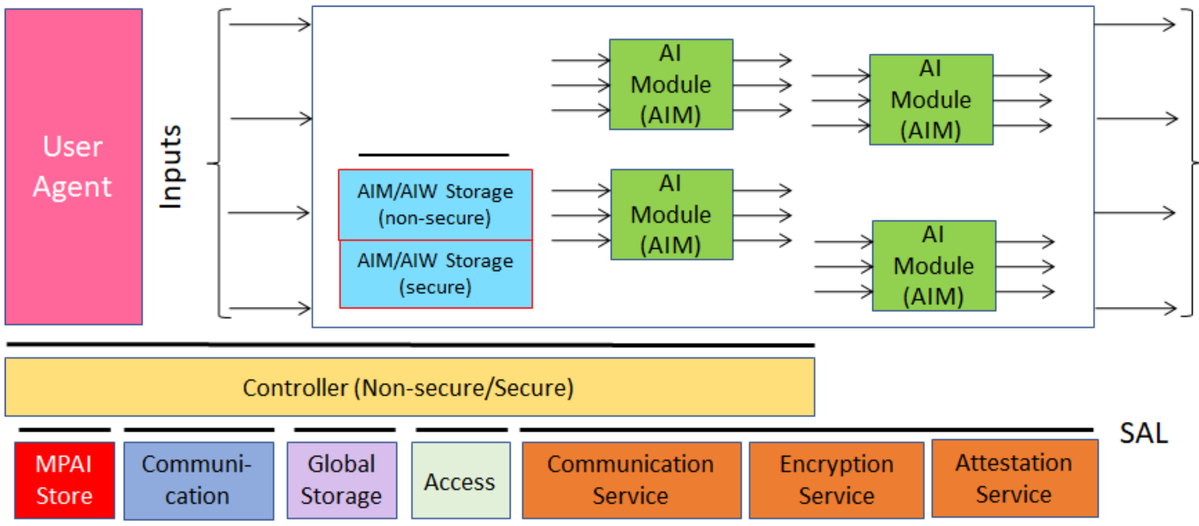

Figure 1 - The AI Framework (MPAI-AIF) V2.1 Reference Model

3. ***Technical Specification: Data Types, Formats, and Attributes (MPAI-TFA) V1.2*** specifies Qualifiers, a type of metadata supporting the operation of AIMs receiving data from other AIMs. Qualifiers convey information on Sub-Types (e.g., the type of colour), Formats (e.g., the type of compression and transport), and Attributes (e.g., semantic information in the Content). Although Qualifiers are human-readable, they are only intended to be used by AIMs. Therefore, Text, Speech, Audio, Visual, and other Data exchanged by AIWs and AIMs should be interpreted as being composed of Content (Text, Speech, Audio, and Visual as appropriate) and associated Qualifiers. Therefore a Text Object is composed of Text Data and Text Qualifier. The specification of most MPAI Data Types reflects this point.
4. ***Technical Specification: Governance of the MPAI Ecosystem (MPAI-GME) V1.1*** defines the following elements:
    1. Standards, i.e., the ensemble of Technical Specifications, Reference Software, Conformance Testing, and Performance Assessment.
    2. Developers of MPAI-specified AIMs and Integrators of MPAI-specified AIWS (Implementers).
    3. MPAI Store in charge of making AIMs and AIWs submitted by Implementers available to Integrators and End Users.
    4. Performance Assessors, independent entities assessing the performance of implementations in terms of Reliability, Replicability, Robustness, and Fairness.
    5. End Users.

The interaction between and among actors of the MPAI Ecosystem are depicted in Figure 2.

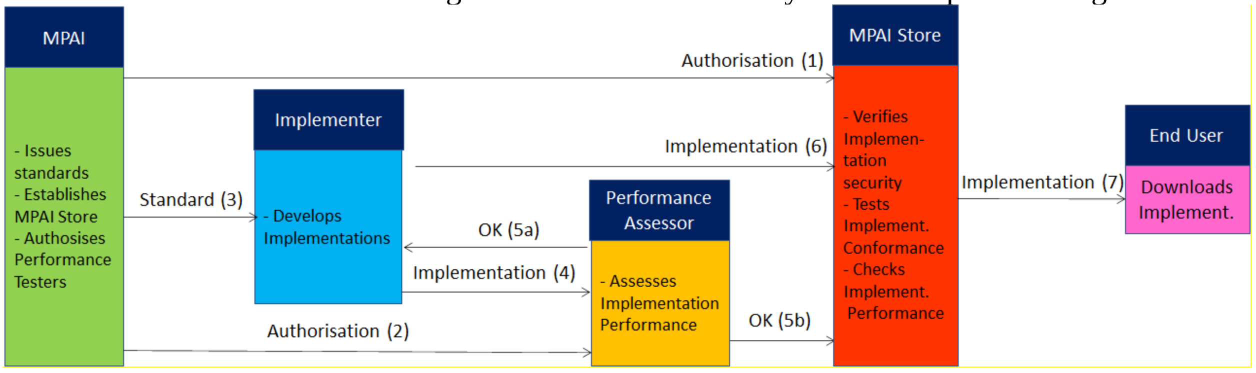

Figure 2 - The MPAI Ecosystem

1 Introduction

In an online Authoritative Multiplayer Game, each player uses a client to send control data to a server. The server updates the current Game State with the data from all clients and then broadcasts it to all clients. The data originating from a client may be properly or maliciously generated and properly received or not received at all. In both cases, the Game State received from the server does not describe a correct and consistent situation.

Among the most widespread game network architectures, *Authoritative Servers* maintain consistency among all connected clients (i.e., the game instances executed locally by players) acting as the central arbiter and controller of the Game State.

Authoritative server architectures are not immune to the challenges posed by network problems, especially Latency, i.e., the delay incurred by data transmitted by a player instance to the game server, because it can disrupt the seamless flow of gameplay.

There are several techniques [1] currently used to cure this situation. In *Client Prediction* [2], client game state is updated locally using predicted or interpolated data while waiting for the server data; in *Time Delay* [3] [4], [5], [6], the server buffers the game state updates to synchronise all clients; and in *Time Warp* [7] the server rolls back the game state to when controller data was sent by a client and acts as if the action was taken then, reconciling this new game state with the current game state. These methods have shortcomings. Client Prediction causes perceptible delay, Time Delay affects responsiveness, and Time Warp disadvantages other players because the new game state likely differs from the previous one.

Technical Report: Server-based Predictive Multiplayer Gaming (MPAI-SPG) – Mitigation of Data Loss Effects (SPG-MDL) V1.0 describes the steps and a methodology that involves server-level prediction techniques. Specifically, the server uses a predictive model to forecast player actions based on historical data and the current Game State when Latency-affected clients are detected. These predictions are then shared with all clients, ensuring a continuous and unified gaming experience. The approach leverages Machine Learning (ML) algorithms, an area of research that is only recently being explored in the context of Latency mitigation strategies for online multiplayer games. Furthermore, the server could also identify possible cheating attempts by comparing predictions with the current Game State, especially when clients have greater control over their local instances (i.e., client prediction).

2 Scope

Technical Report: Server-based Predictive Multiplayer Gaming (MPAI-SPG) – Mitigation of Data Loss Effects (SPG-MDL) V1.0 – in the following also called SPG-MDL V1.0 or SPG-MDL – provides guidelines on the design and use of Neural Networks that produce reliable and accurate predictions that may be used to compensate the absence of players' control data in multiplayer gaming contexts based on authoritative server architectures.

SPG-MDL V1.0 does not consider:

1. Data corrupted by the network.
2. Client cheating.

3 Terms and Definitions

Capitalised Terms have the meaning defined in *Table 1* or by other MPAI Technical Specifications whose definition is accessible online. Non-capitalised terms have the meaning commonly defined for the context in which they are used.

A dash "-" preceding a Term in Table 1 means the following:

1. If the font is normal, the Term in Table 1 without a dash and preceding the one with a dash should be placed before that Term. The notation is used to concentrate in one place all the Terms that are composed of, e.g., the word Client followed by the word Cheating, Data or Prediction.
2. If the font is *italic*, the Term in Table 1 without a dash and preceding the one with a dash should be placed after that Term. The notation is used to concentrate in one place all the Terms that are composed of, e.g., the word Engine preceded by one of the words Behaviour, Game State, Physics, and Rules.

All MPAI terms are accessible online here.

Table 1 - MPAI-SPG specific terms and acronyms

Term	Acronym	Definition
Checkpoint		An invisible gate which a car passes through during a race. A car must go through all the checkpoints in order to complete a lap.
Client		
- Cheating		An operation of a client deliberately sending wrong data to the game server.
- Data	CD	Any data sent by the client to the server.
- Prediction		A technique used on the client in an authoritative server architecture, to predict a Game State.
Curriculum Learning		A training strategy which starts with simpler examples and gradually increasing complexity.
Engine		
- *Behaviour*		The Engine that computes the evolution of the Game State by translating the inputs of the players into corresponding actions.
- *Game State*		A process managing the Game State of the server.
- *Physics*		The Engine that computes the evolution of the Game State using the rules underpinning the physics of the Environment.
- *Rules*		The Engine that computes the evolution of the Game State using the rules underpinning the game logic.
Entity		Any object or character within the game world that can interact with other objects or characters.
Environment		The virtual space where the game takes place and the objects populating it.
Environmental Data		Data regarding the Environment: Tile type and Tile ranking
Game		
- Clock Period		The time required by the game engine to update the game state.

- Message	GM	Messages exchanged between the Game State Engine and the Physics, Rules and Behaviour Engines containing information proper of the relevant Engine.
- Message*	GM*	Messages that each of the Physics, Rules, and Behaviour Engines-AIs receive from the Game State DMUX and send to the Game State Assembler.
- Server		
- State	GS	A data type representing all the information about the game at a given instant.
- State DMUX	GS-DMUX	AIM that demultiplexes the GS received from the GS Engine into discrete Game Messages* (GM_t).
Gaming		
- *Authoritative multiplayer*		A type of gaming where a central system maintains the definitive game state, enforces rules, and resolves player actions to ensure consistency, fairness, and synchronization across participants.
- *Multiplayer*		A form of video gaming that allows multiple players to interact and compete against each other or cooperate as teammates over a network, either locally or online.
- *Predictive Multiplayer*		A technique that anticipates player actions and decisions, enhancing the gaming experience by reducing the impact of latency in multiplayer gaming.
Long Short-Term Memory	LSTM	A type of recurrent neural network architecture, designed to recognize patterns in sequences of data.
Machine Learning	ML	A Process using training data to create a Model able to perform specific tasks such as classification and regression.
Model		A Data Type representing an Artificial Neural Network.
Multilayer Perceptron	MLP	A class of feedforward artificial neural network that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer.
Non-Player Characters	NPC	Game entities not controlled by the clients' inputs.
Neural Network	NN	A set of interconnected data processing nodes whose connections are affected by weights. Also referred to as Artificial Neural Network.
Prediction		
Predicted Game State Assembler	GS-ASS	These predictions are assembled by the Predicted Game State Assembler (GS-ASS), forming the predicted Game State (pGS_{t+1}), which is then communicated back to the server for the next iteration of Game State evaluation. Note that, the formats of both, the computed and predicted GSs are the same.
Ranking		A position in a hierarchy or scale.
Server Prediction		A technique used on the server, to predict the next Game State without the Client Data.
Spatial Attitude	SA	A Data Type representing an object's Position, Orientation and their Velocities and Accelerations.
Tile		A portion of the racetrack.
- Ranking		The position of the Tile from the start of the racetrack.

- Type		The shape of the Tile: straight, narrow curve and wide curve.
Time		
- Delay		A Latency effect mitigation technique used to keep the game synchronised in all clients.
- Warp		A latency effect mitigation technique used on the server to evaluate the consequences of a client's action based on their latency delay.

4 References

[1] S. X. X. a. M. C. Liu, «A survey and taxonomy of latency compensation techniques for network computer games,» *ACM Computing Surveys (CSUR),* pp. 1-34, 2022.

[2] S. a. B. H. a. K. A. a. M. S. a. R. S. Aggarwal, «Accuracy in dead-reckoning based distributed multi-player games,» in *Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games,* 2004.

[3] M. Mauve, "Consistency in replicated continuous interactive media," in *Proceedings of the 2000 ACM Conference on Computer Supported Cooperative Work*, Philadelphia, Pennsylvania, USA, 2000.

[4] C. a. G. T. N. a. G. C. Savery, "The human factors of consistency maintenance in multiplayer computer games," in *Proceedings of the 2010 ACM International Conference on Supporting Group Work,* Sanibel Island, Florida, USA, 2010.

[5] J. a. W. B. W. Xu, «Concealing network delays in delay-sensitive online interactive games based on just-noticeable differences,» in *2013 IEEE International Conference on Multimedia and Expo (ICME),* Sanibel Island, Florida, USA, 2013.

[6] D. a. B. P. Liang, «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,* Singapore, 2006.

[7] J. a. M. C. Sun, «Evaluating streaming and latency compensation in a cloud-based game,» in *Proceedings of the 15th IARIA Advanced International Conference on Telecommunications (AICT),* 2019.

[8] J. B. Diederik P. Kingma, «Adam: A Method for Stochastic Optimization,» in *3rd International Conference for Learning Representations*, San Diego, 2015.

[9] *Technical Specification: AI Framework (MPAI-AIF) V2.0.*

[10] *The MPAI Statutes.*

[11] *The MPAI Patent Policy.*

[12] *Technical Specification; Governance of the MPAI Ecosystem (MPAI-GME) V1.1.*

5 AI Workflow

5.1 Functions

The model considered by SPG-MDL exploits Neural Networks to make accurate Game States predictions. Whenever a client packet is lost, the server evaluates the new Game State thanks to the SPG-MDL prediction: if it is accurate, the Latency of a player is invisible to the unaffected clients without decreasing the game responsiveness or fairness. The Prediction can also be used to detect cheating attempts by comparing the prediction with the server Game State. This use, however, is not considered by SPG-MDL V1.0.

SPG-MDL focuses on keeping the game synchronised between server and clients. Whenever the server needs to evaluate a new Game State, the server requests the SPG-MDL module (Figure 4) to compute the next state through AI Modules before updating the game. If there is any missing data or if the evaluated Game State is too different from the SPG-MDL prediction, the server uses the latter to update the Game State.

In this way, even if a client is experiencing Latency, it is perceived by the other clients as having a continuous playing behaviour. The advantage of this method is that, if predictions are accurate, the effect of the synchronisation process on the lagging client is not noticeable to the player when the network resumes normal operations. Additionally, SPG-MDL helps with reducing the server-client delay. A client with high latency will still receive the results of its actions with a delay, but the server will send the new Game State before receiving the action from the client, effectively halving the waiting time. Therefore, all clients can achieve an optimal user experience. Since MPAI-SPG-MDL is installed only on the server, it is possible to implement additional techniques at Client side to mitigate this problem, for example Client Prediction.

5.2 Reference Model

As SPG-MDL works in an authoritative server context, the server evaluates a new Game State at every time step using all CDs received from remote players (Figure 3).

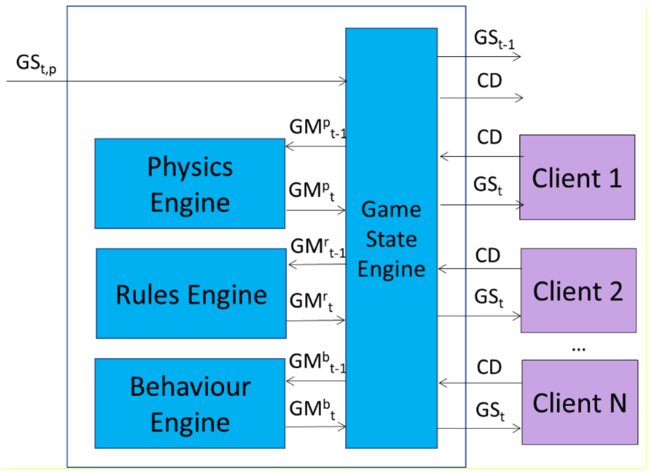

Figure 3 - Operation of Authoritative server

Once computed, the new GS is distributed to each client instance. Within the SPG-MDL framework, the Game State is composed of a series of Game Messages (GM), which are output from three principal engines:

1. Behaviour Engine, orchestrating actions from players and non-player entities.
2. Rules Engine, ensuring adherence to game mechanics.
3. Physics Engine, responsible for physical interactions within the game environment.

SPG-MDL is a “twin” Game Server that is called into action when some CDs are not received (Figure 4). The following steps explain the procedure:

1. The server feeds the current Game State (GS_t) to the SPG-MDL’s Game State Demultiplexer.
2. The Demultiplexer deconstructs the Game State into discrete Game Messages* (GM_t) where the added ‘*’ symbol differentiates these Game Messages from the server’s.

3. Each Game Message* is processed by its respective Engine AI, a trained Neural Network Model that produces a predicted Game Message* (pGM_{t+1}).
4. The Predicted Game State Multiplexer assembles all predictions and forms the predicted Game State (pGS_{t+1}).
5. The predicted Game State is communicated back to the server for the next iteration of Game State evaluation.
6. The server uses the predicted state to compensate for the data shortfall from one or more clients

Note that:
1. The formats of both the computed and predicted GSs are the same.
2. The server independently computes its updated Game State (GS_{t+1}), which is solely derived from available CDs.
3. The server utilises pGS_{t+1} when any Client Data is missing.
4. The online game server architecture described in this section is a reference model. The three engines are not a requirement for any kind of game for which the MPAI-SPG methodology is applied. For example, some games may not use a Physics Engine if physical-based behaviour is not required.

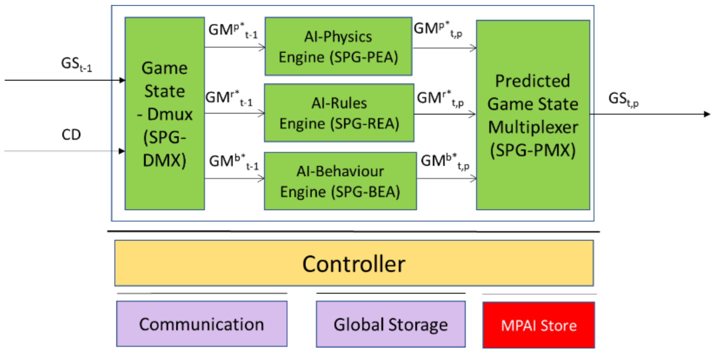

Figure 4 – Reference Model of AI Modules of Mitigation of Data Latency effects (SPG-MDL)

5.3 Input/Output Data

Table 2 provides the input and output data of SPG-MDL:

Table 2 - I/O Data of AI Modules of Mitigation of Data Latency effects (SPG-MDL)

Input	**Description**
Game State	The Game Server Game State at the preceding time instant.
Control Data	Set of players' Control Data at current time.
Output	**Description**
Game State	Predicted Game State at time current time.

5.4 Functions of AI Modules

Table 3 provides the functions of SPG-MDL.

Table 3 - Functions of AI Modules of Mitigation of Data Latency effects (SPG-MDL)

AIM	**Function**
Game State Demultiplexer	Demultiplexes the Game State into its components.
AI-Physics Engine	Predicts the output of the twinned Physics Engine
AI-Rules Engine	Predicts the output of the twinned Rules Engine
AI-Behaviour Engine	Predicts the output of the twinned Behaviour Engine

Predicted Game State Multiplexer	Assembles the components of the predicted Game State

5.5 I/O Data of AI Modules

Table 4 lists the AI Modules and the corresponding input and output data.

Table 4 - I/O Data of AI Modules of Mitigation of Data Latency effects (SPG-MDL)

AIM	Receives	Produces
Game State Demultiplexer	Game State Controller Data	Game Message for AI-Physics Engine Game Message for AI-Rules Engine Game Message for AI-Behaviour Engine
AI-Physics Engine	Game Message for AI-Physics Engine	Game Message from AI-Physics Engine
AI-Rules Engine	Game Message for AI-Rules Engine	Game Message from AI-Rules Engine
AI-Behaviour Engine	Game Message for AI-Behaviour Engine	Game Message from AI-Behaviour Engine
Predicted Game State Multiplexer	Game Message from AI-Physics Engine Game Message from AI-Rules Engine Game Message from AI-Behaviour Engine	Predicted Game State

5.6 Reference Software

5.6.1 Disclaimers

1. This SPG-MDL Reference Software Implementation is released with the BSD-3-Clause licence.
2. The purpose of this Reference Software is to provide a working Implementation of SPG-MDL, not to provide a ready-to-use product.
3. MPAI disclaims the suitability of the Software for any other purposes and does not guarantee that it is secure.
4. Use of this Reference Software may require acceptance of licences from the respective repositories. Users shall verify that they have the right to use any third-party software required by this Reference Software.

5.6.2 Guide to the SPG-MDL code

The game was developed using the Unity game engine, and the networking features implemented through the opensource game networking library Mirror.
The following components are provided:

- The car racing game
- 4 different categories of Agent Players trained using Unity's ML-Agents library
- The dataset used for training, generated simulating game sessions played by the Agent Players
- Jupyter notebooks for training experiments and results
- The trained models used by the AI-Behaviour Engine

Available at https://experts.mpai.community/software/mpai-spg/car-racing-game/

Registration is required to access the repository. Details on how to use all the material is provided in the repository's README page.

6 Process Description and Application

6.1 Process Outline

The process is defined as a series of steps to follow, where the key steps needed to design and implement an MPAI-SPG model are described. The first 4 steps are required to outline the game setup to allow more informed decisions for the implementation of SPG.

1. Select the game.
2. Define the Entities (to enable parameters identification):
   a. Environment
   b. Human-controlled players (HPC) and Non-player characters (NPC)
3. Define the Game State and relevant Entities.
4. Design training dataset.
5. Collect training dataset.
6. Train prediction models:
   a. define viable architectures
   b. define the training parameters
   c. compare training results of different architectures
7. Implement SPG.
8. Evaluate SPG to select the model yielding the best predictions.
9. Implement modules which simulate the disturbances.
10. Evaluate the SPG enabled game experience with human players.

For each step, high level guidelines are provided to outline the actions required. In addition, for each step, an example of how the guidelines should be followed are described using a car racing game as a use case.

6.2 Step 1 - Select a Game

6.2.1 Guidelines

SPG can be applied to any kind of multiplayer online game with the sole requirement that the networking architecture is based on an authoritative server. To function properly, the server must be able to correctly interpret and process the predicted game states received from SPG. Therefore, the server logic must be accessible to the developer as changes to the source code are required.

6.2.2 Example Game

The example game is a 3D multiplayer car racing video game where both the client and the server logic have been developed from scratch. In the game, each client runs a local game instance which sends the Spatial Attitude (SA) of the controlled car to the server by means of Client Data (CD). The server then processes the CD from all connected clients and computes GS_{t+1}, essentially an aggregate of the SAs of all vehicles.

6.3 Step 2 - Define the Entities

6.3.1 Guidelines

Entities are the building blocks of the game's environment and gameplay mechanics. They include any object or character within the game world that can interact with other objects or characters. The objective of this step is to identify how Entities affect the Game State to single out which of them will benefit from predictions. This is done by analysing the selected game and identifying all the Entities that are part of the game design and environment.

6.3.2 Example Game

The racing videogame has two types of game Entities: the immutable ones which populate the environment, and the cars. They can be either Human Playable Characters (HPC) or Non-Playable Characters (NPCs).

Environment

The Environment is the virtual space where the game takes place. For the considered game, it corresponds to the racetracks where the cars drive. Racetracks are built by combining different predefined Tiles; thus, any racetrack can be seen as an array of tiles (Figure 5). Each Tile is characterised by the following properties:

1. *Tile's type*: specifies the shape of the Tile (i.e. turn right, straight)
2. *Tile's rank*: corresponds to the index of the racetrack array associated with the Tile (i.e. the first Tile has rank 0, the fourth Tile has rank 3, etc.).

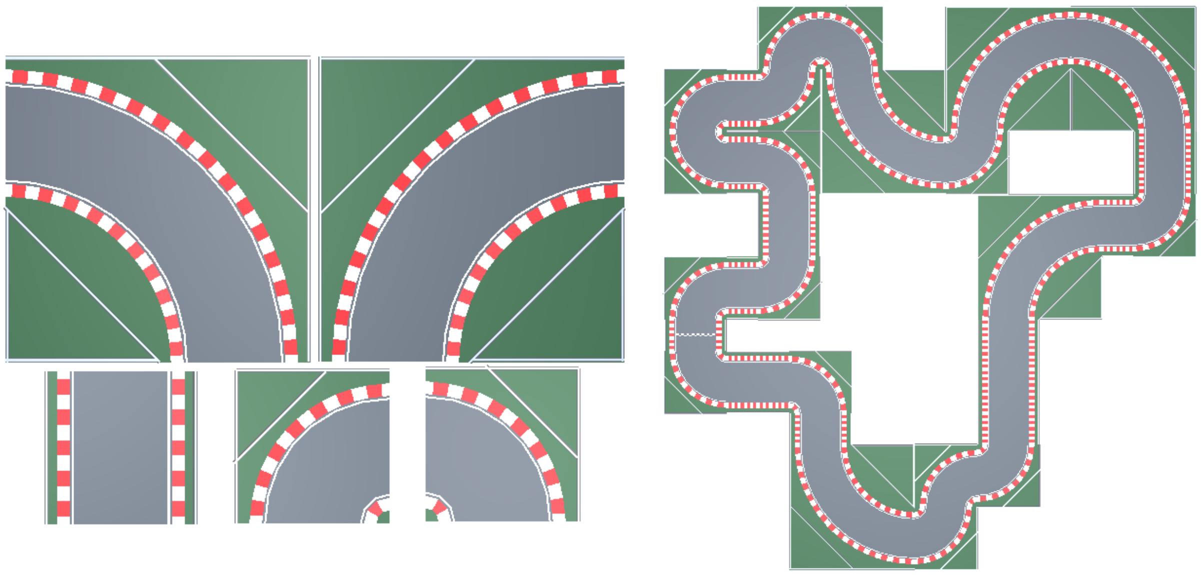

Figure 5 – Modular tiles and an example of a racetrack

Racetracks contain multiple Checkpoints, i.e., a marked location on the track, arranged in an ordered sequence. Each car must pass through a Checkpoint to ensure they are following the correct path. To complete a lap, a car must pass through each Checkpoint in the right order.

HPC/NPC

In the considered game, cars are the only playable Entities. The player's input is captured from a keyboard and controls the vehicle's acceleration, brakes, and steering, ultimately updating its SA. The three inputs are integers values which can have the following values:

- Acceleration: 0 or 1
- Brakes: 0 or 1

- Turning: -1, 0, or 1

Cars can collide with other cars and with the walls of the racetrack. The physical simulation is allocated to the Game Server's Physic Engine.

6.4 Step 3 - Identify the relevant Game State Entities and their properties

6.4.1 Guidelines

To make Predictions, it is required to identify the minimal set of Entities and their properties affecting the Game State. These Entities and properties should be carefully selected to achieve a balance between Prediction accuracy and Model complexity since the Model will be required to operate in real-time. This selection could result from an iterative process evaluating the outcomes validated during the following steps in the procedure.

6.4.2 Example Game

The Game State is affected by:

- The aggregation of the Spatial Attitudes of all cars.
- The next Checkpoint each car must traverse.
- Each car's lap number.

The car's Spatial Attitude was considered as the minimal required data to accurately predict the next Game State. In this context, the predicted data are only processed by the Behaviour Engine.

6.5 Step 4 – Training Dataset Design

6.5.1 Guidelines

In an ideal scenario, the entire Game State can be used as an input vector for predicting the desired properties defined in the previous step. However, the time required to compute a prediction (T_C) can be affected by the number of parameters included in the input vector. Therefore, if T_C exceeds the computational time available in a single game update cycle (16 ms in the best case), a subset of the entire Game State must be selected.

At this stage it is difficult to make an informed decision on the game state elements that should be retained in the subset. Therefore, the selection of game state elements should be extended to cover all those elements which could contribute to the Game State Prediction. Once these elements have been defined, a dataset is needed for training the Neural Network Model. Data may be obtained from already available datasets or should be produced. In the latter case, two approaches are possible: collect data from humans playing the game or rely on AI agents trained to imitate human player behaviour, including the fact that players have different abilities and styles.

The selection of the most effective GS subset must be assessed from the prediction accuracies evaluated in the next steps of the procedure. Therefore, the definition of the final GS subset may require an iterative process.

6.5.2 Example Game

The training data chosen to feed the prediction network includes the car's Spatial Attitude and the surrounding environment. The environment data collected includes:

1. The type of Tile the car is on.
2. The Tile's ranking relative to the other Tiles of the track (e.g. first, fourth or last tile of the track).

3. The car’s position relative to the centre of the Tile.

Since the game was developed from scratch, no datasets were available. Therefore, to suitably train MPAI-SPG, a synthetic dataset was generated by simulating game sessions with autonomous agents using the ML-Agents toolkit, a ML framework for Unity. This approach overcomes the impracticality of collecting data through extensive gameplay sessions.

Autonomous agents, trained with a Curriculum Learning approach, are trained on tracks of increasing complexity, applying penalties for collisions with track walls or other players and for incorrect Checkpoint passages, and rewards for completing laps and correctly navigating through Checkpoints. Also, to emulate a variety of real-world driving styles, four distinct agent types governed by a unique set of rewards and penalties are used. Rewards were applied encouraging acceleration and optimal alignment to the next Checkpoint, and penalties for braking.

6.6 Step 5 – Training Dataset Collection

6.6.1 Guidelines

In the case no dataset is available, one must be produced by running actual gaming sessions, played by either real players or AI agents. When running these sessions, the following points should be kept in mind:

- Player’s abilities should have as wide a variety as possible. This guideline applies both to the case of real players and AI agents.
- To ensure variability, consider different starting conditions when running the gaming sessions.
- Define an adequate sampling frequency for the selected subset of the game state. Frequency selection is influenced by the game type: fast vs slow-paced style game, e.g., real time strategy games vs first person shooters.

Keeping in mind the considerations of Step 4, it is crucial to consider the need to maintain the complexity of the problem at a manageable level. For instance, it is important to find the right balance between having a large variety of game scenarios and finding the most representative ones. The produced dataset must be divided into three sets: train, test and validation.

6.6.2 Example Game

To produce the dataset, the AI agents described in Step 4 raced over a single track in multiple game instances. The choice of collecting data over a single track is an example of balancing complexity over representativeness, as the selected track was the most complex one and included many possible scenarios already appearing in simpler tracks.

Each game was composed of three laps, during which the Game State subset (Step 4) was sampled every 0.02 seconds. At the end of this process, a dataset of 2 million records was collected. The dataset was divided into train, validation and test sets. The train set contains half of the initial dataset, while test and validation sets contain 25% each.

6.7 Step 6 – Train Prediction Models

6.7.1 Guidelines

In most cases, the game state evolution can be modelled as a time series. Therefore, a Neural Network architecture suitable for the task of predicting the Game State should be found. Possible candidates are LSTM, Transformer, and Diffusion Models.

Once a technology has been selected, a Model should be trained using different game state element subsets (see Step 5). Each subset is evaluated for performance against the validation set and a pool of the better subsets (e.g., three or four) is identified for use in the following Steps, where their performance will also be evaluated in the target game.

6.7.2 Example Game

MPAI-SPG's Behaviour Engine AI is designed to predict the car's SA. The Predictions are based on a temporal series of previous Game States using a deep LSTM. 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 model is depicted in Figure 6.

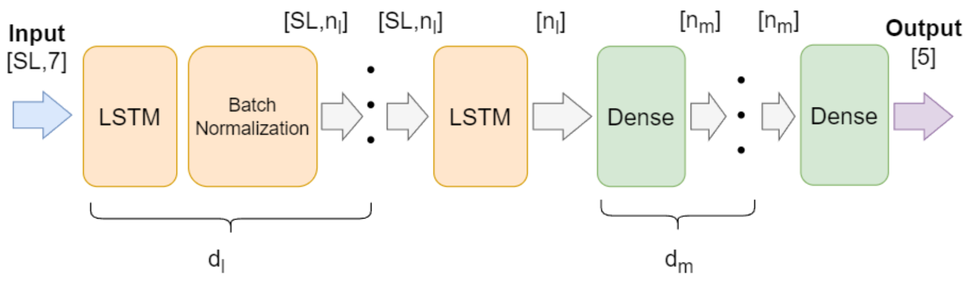

Figure 6 – Neural network architecture.

During training, the loss is measured by the Mean Square Error between the real and predicted car's SA. The LSTM and MLP layers are initialized with the default values from the TensorFlow library. The optimiser leverages the Adam algorithm [8], with an initial learning rate (lr) of 0.001 halved every time the validation loss plateau is reached. The batch size is 512 samples. The Model is trained for 100 epochs with early stopping. After completing the training, the Model parameters that show the lowest validation loss are saved.

The Game State element subset which yielded the best results is composed of 7 elements:

- Tile Type the car is on (1 element),
- Tile Ranking relative to the other Tiles of the track (e.g. first, fourth or last Tile of the track) (1 element)
- Position of the car relative to the centre of the Tile (2 elements)
- Car's velocity (2 elements),
- Car's rotation (1 element).

Therefore, the input for the LSTM network is $R^{SL\times 7}$, a matrix where SL stands for the length of the time sequence.

Table 5 lists the most impactful parameters during training and their suggested values.

Table 5 - Training parameters

Property	Acronym	Min	Max
Numbers of LSTM layers	d_l	0	5
The number of units in each LSTM layer	n_l	32	512

Depth of the MLP	d_m	0	5
Number of hidden units in the MLP	n_m	32	512
Sequence Length	SL	10	50
Time between Predictions	T_p	0.1	0.2

In the context of this game the Mean Absolute Error (MAE) quantifies the difference between the predicted and the real SA. Therefore, a subset of networks which produce the smallest MAE on the validation is selected. By analysing the number of Epochs, it is possible to understand if a network overfits; this information can be used to implement Early Stopping during training.

Table 6 reports the configurations and results of the four best Models trained. Each Model is identified by a unique ID, which will be used to reference them in the following. By comparing the minimum MAE achieved by the Models on the validation set, the ones with higher SL generally demonstrated better performance. Additionally, Models 2 and 4 share the same configuration apart from the SL, but Model 2 overfitted, stopping the training earlier. Among all models, Model 4 achieved the lowest error.

Table 6 - Trained models

ID	**d_l**	**n_l**	**d_m**	**n_m**	**SL**	**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

6.8 Step 7 – Implement SPG

6.8.1 Guidelines

Ideally, TC should be equal or smaller than the clock period, as this ensures that Predictions are computed before a new game state is evaluated. However, this condition is not guaranteed, in case the SPG Engines cannot keep up with the computational load. As shown in Figure 7, two strategies are possible. One is to wait until the server is ready to provide the latest available data, which happens at multiples of the game clock period (Tg). Alternatively, the Engines can start working on the input data as soon as the previous computation is completed. In the first case, the training can be done by keeping the Prediction time constant, T_p. The second option may be more effective but requires that Engine be trained to make Predictions on a variable time frame.

6.8.2 Example Game

In the game, SPG was required to predict only the car's SA. Therefore, only the Behaviour Engine AI was developed. In this context, the Game State DMUX extracts, for each car, a game message GM_t^{B*} containing the environment surrounding the car and its SA, sending it to the Behaviour Engine AI. The engine outputs a series of pGM_{t+1}^{B*} for each car, containing the predicted SA. These data are then combined by the Game State Assembler which outputs a predicted Game State (pGS_{t+1}) which is forwarded to the Game Server. The first solution described above was implemented to account for the problem of T_C exceeding the game clock period. The value of T_p is equal to 0.1 seconds. Therefore, the previous annotation changes in the following: pGM_{t+Tp}^{B*} and pGS_{t+Tp}.

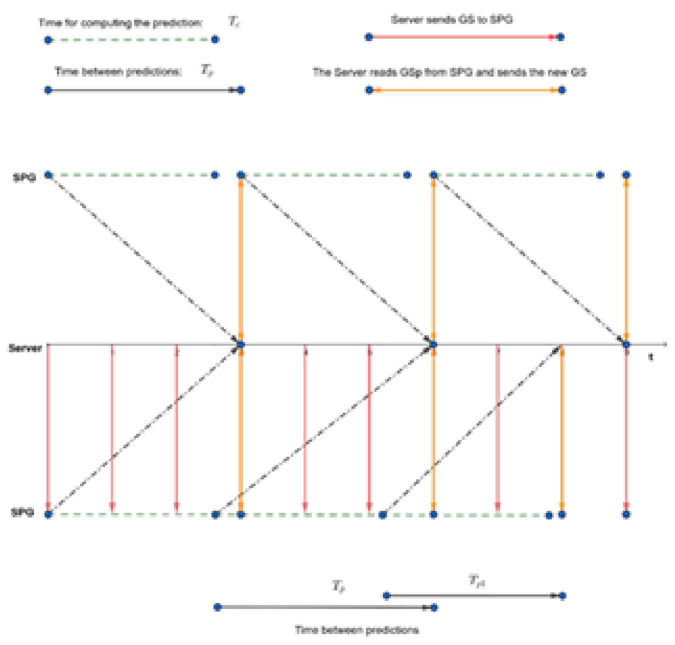
Time for computing the prediction: T_c
Time between predictions: T_p
Server sends GS to SPG
The Server reads GSp from SPG and sends the new GS
SPG
Server
t
SPG
T_p
T_{p1}
Time between predictions

Figure 7- Prediction alternatives when Tc is greater than Tp

6.9 Step 8 – Using and Evaluating SPG Predictions

6.9.1 Guidelines

When using MPAI-SPG Predictions in a game environment, two possible issues may arise:

- The MAEs measured in the training environment may differ when Predictions are applied in the game because the models which give the best results during training (Step 6) may not be the best performing (i.e., prediction accuracy) in the game.
- The server may need to apply MPAI-SPG Predictions for several consecutive times. Whenever Predictions are used, they serve as input for the subsequent MPAI-SPG predictions. This causes error accumulating as time progresses. This behaviour will be referred in the following as Error Accumulation.

Therefore, a pool of Models with the best results obtained in Step 6 (i.e., training environment) should be evaluated in the game scenario and the best performing one selected for use in the game. To perform this evaluation, the outputs of two simulations should be compared for each Model, one where Predictions are not used (i.e., the human or agent player, is in control), and another where SPG is in control. Data should be collected from the simulations to calculate a comparison metric (e.g., MAE of the SA). The comparison metric can be used to decide which Model produces the best Predictions overall and by analysing the metric's evolution over time, the level of Error Accumulation can be assessed. This analysis will verify which of the Models from Step 6 is the best choice for the game environment.

Also, it should be noted that if the Models have been trained with synthetic data (i.e., data collected from agent players), when asked to predict the Game State using human player data a domain shift issue could happen. Therefore, in this scenario, the same evaluation should be performed by comparing SPG Predictions with Entities controlled by human players.

6.9.2 Example Game

To identify which of the four models (trained in Step 6) has the lowest error accumulation, several games were run where a ghost car would invisibly run alongside an agent player's car. The ghost car would normally replicate the player's car SA and, at different intervals, it would be controlled by SPG predictions, applied for 1 consecutive second. During this interval the game saves the SA of the car controlled by the player and the car controlled by SPG predictions. The result of this process, repeated a few tens of times, was used to compute the MAE between position, velocity and orientation of the car controlled by the agent player and the car controlled by SPG predictions.

The MAEs between the SPG predictions and the car driven by the agent player is shown in Figure 8. The MAEs were normalized to the highest possible change in the position, velocity and orientation that a car can make in 0.2 seconds (T_p). The results reveal that Model 4 was the one with the highest error accumulation.

Model 4, the model with the lowest validation MAE during training (Step 6), achieved the worst overall prediction quality between all models. Contrary to our anticipation, the other three models (having 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 orientation, the impact is mitigated by the lower magnitude of the rotation error. In fact, Model 1 attains the lowest error on the position evaluation.

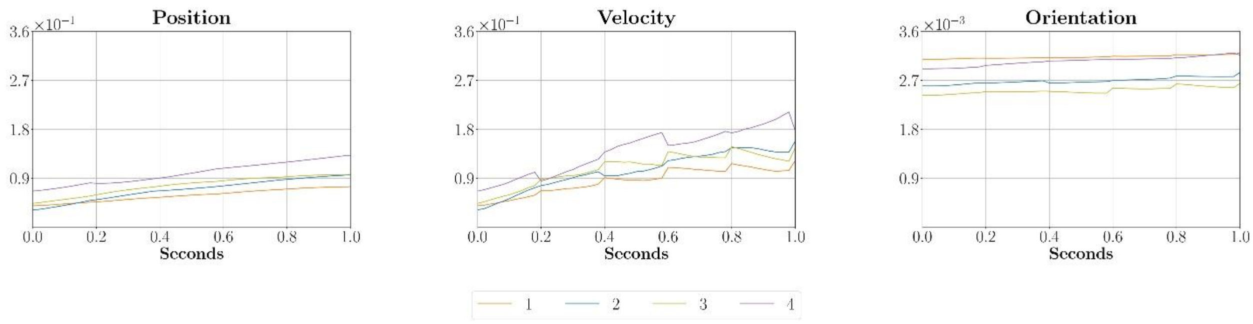

Figure 8 - Normalized Position, Velocity and Orientation MAEs between the agent and the SPG predictions for the four models trained in Step 6.

The second experiment was conducted keeping all conditions but replacing the agent player with human players, 12 of which played 2 laps and using only Model 1 to compute the SPG predictions. The objective was to assess how well SPG was able to predict missing data when the player was a human. Figure 9 shows that the accumulated error of the Position and Velocity after 1 second was about 1.5 times (blue line) the error of the agent player (green line). The performance of the Orientation on the human and agent players was about the same.

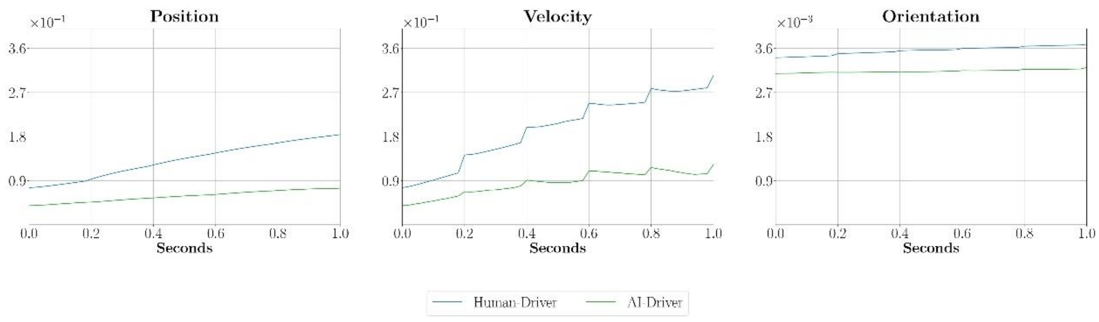

Figure 9 - Normalized Position, Velocity and Orientation MAEs between the human and the SPG predictions using Model 1.

6.10 Step 9 – Simulate Disturbance

6.10.1 Guidelines

The experiments in the previous step aimed at evaluating the prediction accuracy of the trained models to select the better performing one. Once identified, SPG should be evaluated (using the identified model) in an actual online multiplayer scenario. Since SPG activates in case of missing data due to network issues, this condition should be simulated. This simulation can be achieved at two levels:

- Application Level: the application (the game) purposely does not send data (client) or discards it (server).
- Network Level: actual network issues (latency or packet loss) are simulated outside the application environment.

6.10.2 Example Game

In the example game, network disturbance was simulated at the application level. This was achieved by developing a discard module, in charge of discarding client data on the server. This module was controlled by two parameters: *discard length*, the number of consecutive seconds where data is discarded, and *discard interval*, the seconds between discard sequences. Different values of these parameters establish three Discard Levels (DLs), summarized in Table 7.

Table 7 - Discard Level Parameters

DL	Length (s)	Interval (s)
DL1	0	0
DL2	0.3	10 +/- 2
DL3	0.6	8 +/- 2

6.11 Step 10 – SPG Qualitative Assessment

6.11.1 Guidelines

As a final step the gaming experienced by human players must be assessed. Human players should play the game under two conditions: no network disturbances (i.e., no predictions are computed and used by SPG) and simulating disturbances with SPG active to compensate for missing data. Several game runs should be performed under both conditions and at the end of each condition players should be asked to fill out a questionnaire addressing the following key points:

1. Perception of anomalous behaviour from player-controlled and game-controlled entities
2. Perceived game responsiveness to players' inputs
3. Overall gaming experience

These key points apply to all games, but the questionnaire may address other game-dependent key points.

Depending on the availability of human players, the game may be played by some human and some agent players.

To further evaluate the impact of SPG, a third condition can be tested with simulated disturbances but without SPG correction.

6.11.2 Example Game

In the example game the qualitative assessment involved gaming sessions where:

- One human player raced against other 4 agent players.

- Each agent player was executed on a separate process instance and connected as client to the server
- Only the 4 agent players were affected by simulated network issues, using the discard module described in Step 9.

The human player completed several game sessions under two conditions: absence and presence of network issues. In the latter condition two Discard Levels (DL1 and DL2 from Table 7) were used. At the end of each game session the player filled a questionnaire composed of the following questions (aimed at addressing the key points described in the guidelines):

- Q1: How would you rate your gaming experience in this match?
- Q2: How would you rate the responsiveness of the inputs?
- Q3: How many anomalous behaviours did you encounter during the match?

The results summarised in Figure 10 show that between the no network issues (blue bar) case and network issues with DL1 level (orange bar) there is a very small difference for the Game Experience and Responsiveness scales, while for DL2 (green bar) the difference is more evident. On the other hand, the perception of odd behaviours (green bar) increases in DL1 and DL2, however remaining below 3 on a scale of 5.

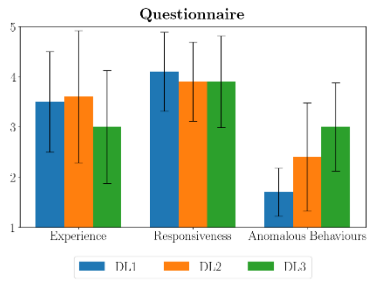

Figure 10 - Questionnaire results.

Annex 1 - General MPAI Terminology

Capitalised Terms not already included Table 1 are defined in Table 8. To concentrate in one place all the Terms that are composed of a common name followed by other words (e.g., the word Data followed by one of the words Format, Type, or Semantics), the definition given to a Terms preceded by a dash "-" applies to a Term composed by that Term without the dash preceded by the Term that precedes it in the column without a dash.

Table 8 - MPAI-wide Terms

Term	Definition
Access	Static or slowly changing data that are required by an application such as domain knowledge data, data models, etc.
AI Framework (AIF)	The environment where AIWs are executed.
AI Model (AIM)	A data processing element receiving AIM-specific Inputs and producing AIM-specific Outputs according to according to its Function. An AIM may be an aggregation of AIMs.
AI Workflow (AIW)	A structured aggregation of AIMs implementing a Use Case receiving AIW-specific inputs and producing AIW-specific outputs according to the AIW Function.
Application Standard	An MPAI Standard designed to enable a particular application domain.
Channel	A connection between an output port of an AIM and an input port of an AIM. The term "connection" is also used as synonymous.
Communication	The infrastructure that implements message passing between AIMs.
Component	One of the 7 AIF elements: Access, Communication, Controller, Internal Storage, Global Storage, Store, and User Agent
Composite AIM	An AIM aggregating more than one AIM.
Component	One of the 7 AIF elements: Access, Communication, Controller, Internal Storage, Global Storage, Store, and User Agent
Conformance	The attribute of an Implementation of being a correct technical Implementation of a Technical Specification.
- Testing	The normative document specifying the Means to Test the Conformance of an Implementation.
- Testing Means	Procedures, tools, data sets and/or data set characteristics to Test the Conformance of an Implementation.
Connection	A channel connecting an output port of an AIM and an input port of an AIM.
Controller	A Component that manages and controls the AIMs in the AIF, so that they execute in the correct order and at the time when they are needed
Data	Information in digital form.
- Format	The standard digital representation of Data.
- Type	An instance of Data with a specific Data Format.
- Semantics	The meaning of Data.
Descriptor	Coded representation of a text, audio, speech, or visual feature.
Digital Representation	Data corresponding to and representing a physical entity.

Ecosystem	The ensemble of actors making it possible for a User to execute an application composed of an AIF, one or more AIWs, each with one or more AIMs potentially sourced from independent implementers.
Explainability	The ability to trace the output of an Implementation back to the inputs that have produced it.
Fairness	The attribute of an Implementation whose extent of applicability can be assessed by making the training set and/or network open to testing for bias and unanticipated results.
Function	The operations effected by an AIW or an AIM on input data.
Global Storage	A Component to store data shared by AIMs.
AIM/AIW Storage	A Component to store data of the individual AIMs.
Identifier	A name that uniquely identifies an Implementation.
Implementation	1. An embodiment of the MPAI-AIF Technical Specification, or 2. An AIW or AIM of a particular Level (1-2-3) conforming with a Use Case of an MPAI Application Standard.
Implementer	A legal entity implementing MPAI Technical Specifications.
ImplementerID (IID)	A unique name assigned by the ImplementerID Registration Authority to an Implementer.
ImplementerID Registration Authority (IIDRA)	The entity appointed by MPAI to assign ImplementerID's to Implementers.
Instance ID	Instance of a class of Objects and the Group of Objects the Instance belongs to.
Interoperability	The ability to functionally replace an AIM with another AIW having the same Interoperability Level
- Level	The attribute of an AIW and its AIMs to be executable in an AIF Implementation and to: 1. Be proprietary (Level 1) 2. Pass the Conformance Testing (Level 2) of an Application Standard 3. Pass the Performance Testing (Level 3) of an Application Standard.
Knowledge Base	Structured and/or unstructured information made accessible to AIMs via MPAI-specified interfaces
Message	A sequence of Records transported by Communication through Channels.
Normativity	The set of attributes of a technology or a set of technologies specified by the applicable parts of an MPAI standard.
Performance	The attribute of an Implementation of being Reliable, Robust, Fair and Replicable.
- Assessment	The normative document specifying the Means to Assess the Grade of Performance of an Implementation.
- Assessment Means	Procedures, tools, data sets and/or data set characteristics to Assess the Performance of an Implementation.
- Assessor	An entity Assessing the Performance of an Implementation.
Profile	A particular subset of the technologies used in MPAI-AIF or an AIW of an Application Standard and, where applicable, the classes, other subsets, options and parameters relevant to that subset.
Record	A data structure with a specified structure

Reference Model	The AIMs and theirs Connections in an AIW.
Reference Software	A technically correct software implementation of a Technical Specification containing source code, or source and compiled code.
Reliability	The attribute of an Implementation that performs as specified by the Application Standard, profile, and version the Implementation refers to, e.g., within the application scope, stated limitations, and for the period of time specified by the Implementer.
Replicability	The attribute of an Implementation whose Performance, as Assessed by a Performance Assessor, can be replicated, within an agreed level, by another Performance Assessor.
Robustness	The attribute of an Implementation that copes with data outside of the stated application scope with an estimated degree of confidence.
Scope	The domain of applicability of an MPAI Application Standard
Service Provider	An entrepreneur who offers an Implementation as a service (e.g., a recommendation service) to Users.
Standard	A set of Technical Specification, Reference Software, Conformance Testing, Performance Assessment, and Technical Report of an MPAI application Standard.
Technical Specification	(Framework) the normative specification of the AIF. (Application) the normative specification of the set of AIWs belonging to an application domain along with the AIMs required to Implement the AIWs that includes: 1. The formats of the Input/Output data of the AIWs implementing the AIWs. 2. The Connections of the AIMs of the AIW. 3. The formats of the Input/Output data of the AIMs belonging to the AIW.
Testing Laboratory	A laboratory accredited to Assess the Grade of Performance of Implementations.
Time Base	The protocol specifying how Components can access timing information
Topology	The set of AIM Connections of an AIW.
Use Case	A particular instance of the Application domain target of an Application Standard.
User	A user of an Implementation.
User Agent	The Component interfacing the user with an AIF through the Controller
Version	A revision or extension of a Standard or of one of its elements.
Zero Trust	A cybersecurity model primarily focused on data and service protection that assumes no implicit trust.