ML-MAS: A Hybrid AI Framework for Self-Driving Vehicles

Hilal Al Shukairi
University of Aberdeen
Aberdeen, United Kingdom
hilalss@gmail.com

Rafael C. Cardoso
University of Aberdeen
Aberdeen, United Kingdom
rafael.cardoso@abdn.ac.uk

ABSTRACT

Machine Learning (ML) techniques have been shown to be widely successful in environments that require processing a large amount of perception data, such as in fully autonomous self-driving vehicles. Nevertheless, in such a complex domain, ML-only approaches have several limitations. In this paper, we propose a hybrid Artificial Intelligence (AI) framework for fully autonomous self-driving vehicles that uses rule-based agents from symbolic AI to supplement the ML models in their decision-making. Our framework is evaluated using routes from the CARLA simulation environment, and has been shown to improve the driving score of the ML models.

KEYWORDS

Hybrid AI; BDI agents; deep learning; self-driving vehicles; CARLA

ACM Reference Format:
Hilal Al Shukairi and Rafael C. Cardoso. 2023. ML-MAS: A Hybrid AI Framework for Self-Driving Vehicles. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023*, IFAAMAS, 9 pages.

1 INTRODUCTION

Machine Learning (ML) techniques are a critical part of a fully autonomous system that requires a large amount of sensing. Deep learning in particular has been shown to be very effective in image processing in the context of autonomous vehicles [13, 22]. The considerable effort and time spent to develop a suitable ML model may never provide a perfect solution that manages to avoid all critical scenarios (e.g., collisions, accidents) and to follow all road rules. There will always be some situations for which the model has not trained enough or does not have enough accuracy in its prediction that can lead even the best model to fail [16]. Re-training the model may not be a valid option due to the amount of time and difficulty in preparing the training environment for those critical cases, especially if such situations happen during real world deployments where it can cause harm to humans. For such complex systems, verification and validation is usually a requirement. However, due to the black-box nature of most deep learning techniques, formal verification is either impractical or unfeasible [17]. Even if critical parts of the system can be verified, formal verification will not help in improving the autonomous behaviour. Nevertheless, verification and validation should still be a part of the pipeline of the complete system, but for the remainder of this paper we focus on improving the autonomous behaviour of the vehicle.

An alternative option to improve autonomy in self-driving vehicles is to combine rule-based techniques from symbolic Artificial Intelligence (AI) with ML. This allows the system to provide more deliberate and rational decision-making in difficult scenarios where ML underperforms. This is known as hybrid AI, which has been researched in the past [7, 12, 26] and has recently seen a resurgence in Neuro-Symbolic AI [25]. We view Neuro-Symbolic AI as a subcategory of Hybrid AI, when neural components are intrinsically connected to rule-based reasoning (e.g., when a rule-based system feeds into the learning procedure). Therefore, for this paper we prefer to use the more general term of hybrid AI to describe the combination of learning and cognitive reasoning in decision-making.

The notion of combining deliberative reasoning with learning has been extensively studied in human decision-making and cognitive theories, as described by the psychologist Daniel Kahneman in his book "Thinking Fast and Slow" [18]. According to Kahneman's theory, human decisions are made by a cooperation between "System 1: Thinking fast" and "System2: Thinking slow". System 1 is used for intuitive, imprecise, fast and unconscious decisions. In contrast, System 2 is used in more complex situations that require logical and rational thinking. Kahneman estimates that about 95% of our thinking is using System 1 to make decisions. When applying this concept to Hybrid AI we have that System 1 represents the ML component (unconscious decisions learned by experience) and System 2 represents the symbolic AI component (cognitive decisions). Humans learn how to drive through training and practice. The more we drive in a particular route the more we rely on System 1, but any uncommon scenario we encounter (e.g., taking a new route for the first time, or an incoming vehicle going the wrong way) will require System 2 to take over. For the remainder of this paper, we refer to System 1 as the learning system, and System 2 as the cognitive system.

In this paper, our aims are to explore decision-making using both a ML system and a rule-based system, and to determine how to switch control between the two in order to make better decisions. We use pre-trained ML models as our System 1 and introduce rational rule-based agents as our System 2. When there is a critical situation (e.g., incoming collisions or traffic jams and congestion) that cannot be left for the ML model to decide (due to under-fitting, over-fitting, or accuracy issues), then we should rely on the cognitive reasoning of the agents. We use CARLA [14] as our open-source simulation environment, which was developed to support research in the development, training and testing of autonomous urban driving systems. System 1 is a pre-trained ML model and System 2 is a cognitive agent based on the Belief-Desire-Intention (BDI) model [6, 23] programmed in the Jason Multi-Agent System (MAS) language [5].

At a more theoretical level, we investigate four research questions from the "Thinking Fast and Slow in AI" [3], which discussed the theory from "Thinking Fast and Slow" [18] when applied to AI:

Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), A. Ricci, W. Yeoh, N. Agmon, B. An (eds.), May 29 – June 2, 2023, London, United Kingdom.

RQ1 How do we model the governance of System 1 and System 2 in an AI?
RQ2 Which factors trigger the switch between the two systems?
RQ3 How should System 2 act once the switch is triggered?
RQ4 When should problems be handed back from System 2 to System 1?

2 RELATED WORK

The work in [1] proposes a verification technique for decision-making of self-driving vehicles that combines design phase and runtime verification. The decision-making uses a rational agent based on the BDI model, which perceives the environment, sensor data and feedback to follow a self-planning path. The decision is verified by two well-known model checkers: Model Checker for Multi-Agent Systems (MCMAS) [21], used during the design phase to check the consistency and stability of the BDI agent logic; and PRISM [19], a probabilistic model checker used at runtime to verify the success probability of the decisions in order to help the rational agent select the best choice. The BDI agent is rather simple, driving in a parking-lot scenario, not in a complete self-driving environment such as the ones provided in CARLA. Furthermore, deep learning is used only as a fusion-sensor input, and not as a hybrid AI decision-making mechanism.

Hierarchical Adaptable and Transferable Networks (HATN) [27] is an approach used to generate driving behaviours by mimicking the human's cognitive level during driving. This approach proved to be efficient in challenging scenarios such as roundabouts and unusual intersections. The hierarchical component of the framework comes from breaking complex scenarios into smaller tasks that make the learning more efficient. The model architecture is based on a set of Graph Recurrent Unit (GRU) networks and dense fully connected neural networks. This approach is useful for the specific scenarios it targets, but cannot be applied to more general driving (e.g., CARLA routes).

Despite the recent boon in hybrid and neuro-symbolic AI [2, 24, 28], there has not been many recent works combining ML with BDI-based agents. Some instances of work done in the past include the work in [11], which uses a weightless neural network and a BDI agent to extract landmark information from a map in order to determine the location of a robot in the map. The virtual neural sensor partly analyses the image, feeding pre-processed data to the agent (which is a more simplistic rule-based system rather than a proper BDI agent) for calculating the estimated location. An extension of that work can be found in [15], this time with a more traditional BDI agent containing beliefs and plans. The BDI agent is combined with Artificial Neural Networks to perform active video surveillance in two application domains, railway tunnels and outdoor storage areas. The main architecture remains similar, the neural network is used to generate pre-processed data from image detection and then the agent is used to further interpret that data.

3 THE ML-MAS FRAMEWORK

The ML-MAS framework combines the decision-making of pre-trained ML models (System 1) and BDI agents (System 2) in an effort to integrate learning and rational reasoning behaviours. Our framework is applied to the self-driving vehicles domain. In order to be able to evaluate and test our approach in practical scenarios, we use the CARLA simulation environment [14]. CARLA provides realistic 3D simulation of urban driving with sensors such as RGB Cameras, LiDAR, obstacle detectors, etc. Furthermore, it includes a Python API to interact with the simulation which facilitates its integration with the ML models and the BDI agents. In addition, CARLA offers dedicated route evaluation metrics via the *Leaderboard* library [20]. The library also offers debug functionalities such as route replays from recorded execution logs. In the traditional challenge, the goal is to provide full autonomous control of a vehicle over a series of routes containing other vehicles and pedestrians while trying to minimise accidents, collisions, and traffic violations at the same time. The other entities in the simulation are controlled by a centralised internal traffic manager.

Figure 1 gives an overview of the ML-MAS main components. CARLA, the pre-trained ML model, and the Leaderboard library are all used as external black-box resources, i.e., they are not altered in any way. The pre-trained ML models and the Leaderboard routes we use are explained in our evaluation section (Section 4). The main internal components of ML-MAS are the BDI bridge, the orchestrator, and the BDI agent.

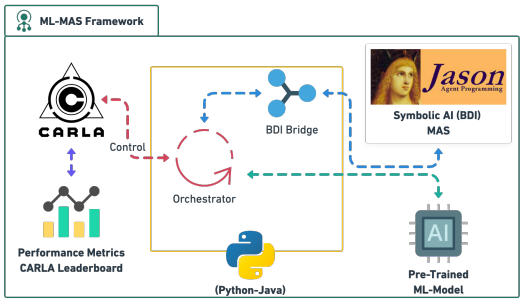


Figure 1: ML-MAS framework overview.

Traditional if-else rules and other symbolic decision techniques can result in higher complexity, making the system unmanageable and hard to maintain over time. BDI agents give our framework the capability of accomplishing goals in a rational way using the agent's know-how in the form of plans (e.g., to avoid front, back, crossing collisions). Moreover, BDI plans are scalable, extendable, and concurrent, which is important in our application domain because a rule can be activated while another one is still executing. While this initial set of rules could be represented with a simpler rule-based system, it would make future extensions more difficult and costly (e.g., convoy of autonomous cars). According to recent literature reviews [4, 8], Jason [5] is one of the most well-know BDI agent programming languages that is still being actively maintained.

3.1 BDI Bridge

The BDI bridge implements a seamless communication bridge between the BDI agent and the CARLA API. The architecture of the BDI bridge is shown in Figure 2. The bridge is based on socket communication and consists of a client part (in Java-Jason) and a server (in Python). The former is integrated into the BDI agent component, and the latter is integrated into the orchestrator that communicates with CARLA using its API.

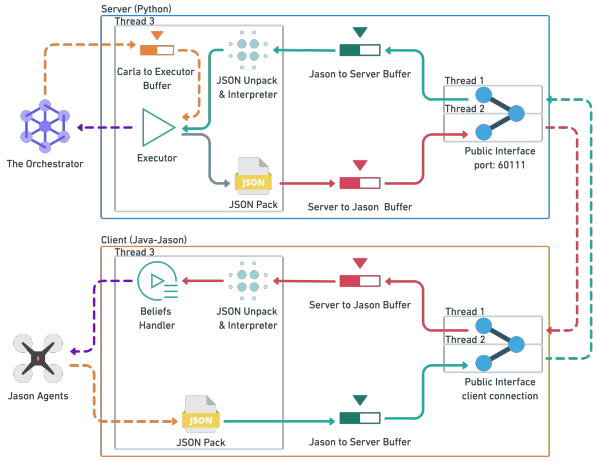


Figure 2: Bridge connecting the orchestrator and agents.

To improve efficiency, both client and server run in three main threads: a thread to send the messages, a thread to receive the messages, and a thread to handle the messages. Each thread works independently and passes information using a set of buffers. This design ensures no thread conflicts or reaches a deadlock with other threads, as only one thread can write to the specific buffer while the others can only read from it. Moreover, the communication between the client and the server is done automatically and handles any disconnecting and reconnecting required, transparent to the parties integrating it, providing a reliable communication service.

The messages are specified in the well-known JSON format. Both client and server use a JSON unpacker and interpreter to identify the message type and decide how to handle each message. Listing 1 shows a control (action) message in JSON format as an example of a message being sent from the agent to the orchestrator.

```
"type": {"id": 1, "name": "control"},
"data": {
    "metricsType": 1,
    "throttle": 0.0,
    "steer": 0.0,
    "brake": 0.0,
    "reverse": false,
    "repeat": 1
        }
```

Listing 1: Example of a control message in JSON format.

The message handler thread behaves differently in the server and in the client. The server has an *executor* which processes the action requested by the message received from Jason. The client has a dedicated *Beliefs Handler* that manages the agents' beliefs based on the information received from the orchestrator. Note that the Jason BDI client can be replaced with another BDI client to make it compatible with other BDI-based languages.

3.2 Orchestrator

The orchestrator implements the required functions to interface with CARLA and with the Leaderboard (in particular, the Autonomous Class). A configuration file for the orchestrator can be easily changed to allow it to integrate with any ML model that inherits the same Autonomous Class. The functionality of the orchestrator includes loading the ML model, its configuration and pre-trained weights, and forwarding the data from the required sensors that are requested by the model. It also ensures the availability of the sensor data needed for the BDI agent, as well as pre-processing the data and sending only what is required by the agent. This acts as a filter which solves the common problem of flooding the belief base of the agent with too much information that is not being used. Communication with the BDI agent is done through the BDI bridge. The orchestrator then waits for an action from either the ML model or from the BDI agent. For efficiency, if in a simulation frame there are no data to send to the agent, then it relies on the action from the ML model only.

Algorithm 1 gives an overview of the function that loads the configured routes and scenarios using the *getRoutesAndScenarios*() function (line 1). Frames Per Second (FPS) is locked into 20 frames per gaming time second (line 2). Then, it loops through each route ensuring that the simulation environment (loading the town map, the position of actors, the weather condition, etc.) and the required systems (loading up the weights in the ML model, connecting with the BDI agent via the BDI bridge, etc.) are ready and properly configured (lines 5–10). Note the use of game time as well as the system time. The system time is not used in the evaluation, it is merely there for informational purposes. The evaluation is designed to not be affected by computation speed and response time of the solution or the hardware specification being used to run it. This is achieved by using *Game_time*, a game second runs 20 simulation frames (this is the default value which is used in the competitions) where with each frame every actor in the simulation is allowed to act (vehicles, pedestrians, traffic lights, etc.).

For every game time frame, the orchestrator collects the sensor data (coming from the simulation environment) and call the function *runStep*, which will interface with the ML model and the BDI agent, returning a control message from one of them that is then sent to the environment for execution (using the *eval*() function). The evaluation metrics are stored every time a route concludes.

Algorithm 2 describes the *runStep* function. Lines 5–7 check if there are any repeat actions to perform. Repeat is a special property that the BDI agent can specify when sending a message with an action. It is used to inform the orchestrator to perform the same action for a specified number of frames without asking the BDI agent or the ML model until the repeat is over. This feature is important because some plans are designed to stop the vehicle from any speed that they may have, so a brake action has to be repeated a number of times proportionally to the vehicle speed in order to achieve the intended goal of the plan.

If there is no BDI action being repeated, then the orchestrator sends the sensor data to the ML model and receives its control message back. The sensor data is pre-processed in preparation to be sent to the BDI agent. If the pre-processed data does not trigger any BDI plan, then the orchestrator uses the ML control. Otherwise, if a BDI plan is triggered, then the pre-conditions of the triggered plan are tested against the set of pre-processed data (and the ML control as well in some cases). If the pre-conditions are not satisfied, then the BDI agent returns a *noaction*. Otherwise, the orchestrator receives a control message from the BDI agent, which will then take priority over the ML model control.

Algorithm 1: Main function to run a scenario.

```
Function main()
    Route_list ← getRoutesAndScenarios()
    FPS ← 20
    foreach Route ∈ Route_list do
        Sensors_list ← getMLSensors() · getBDISensors()
        loadWorldMap(Route, Sensors_list)
        ML_Model ← prepareMLWeights()
        BDI_Agent ← connectBDIBridge()
        Game_time ← 0
        System_start_time ← getCurrentTimeInSec()
        while Route_status ≠ finished do
            Game_time ← Game_time + 1
            for k ← 1 to FPS do
                Sensors_data ← collectData(Sensors_list)
                Control ← runStep(Sensors_data)
                Metrics ← Metrics · eval(Route, Control)
                Route_status ← status(Route, Metrics)
                if Route_status = finished then
                    break
        System_time ←
          getCurrentTimeInSec() − System_start_time
        store(Metrics, Game_time, System_time)
```

Algorithm 2: Orchestrator function to run a step.

```
Function runStep (Sensors_data)
    Last_control ← getLastControl()
    Repeat_counter ← getRepeatCounter()
    Frame_number ← Frame_number + 1
    if Repeat_counter ≥ 1 then
        Repeat_counter ← Repeat_counter − 1
        return Last_control        // Previous BDI control
    ML_control ← getMLControl(Sensors_data)
    Preprocessed_data ← preprocess(Sensors_data)
    if Preprocessed_data ∉ BDI_triggers() then
        Repeat_counter ← 0
        return ML_control
    Last_control, Repeat_counter ←
      getBDIControl(Preprocessed_data, ML_control)
    if Last_control = noaction then
        Repeat_counter ← 0
        return ML_control
    addBDIMetrics(Last_control, Repeat_counter)
    return Last_control                    // BDI control
```

3.3 Data Pre-Processing

The simulation environment in CARLA comes with a set of sensors that researchers can use. There are two challenge tracks in the CARLA Leaderboard. "Sensors Track" allows the use of 6 sensors: GPS, IMU, LiDAR, RADAR, RGB camera, and a speedometer. "Map Track" adds an additional pseudosensor with information about the map. Our BDI agent currently only uses information from the LiDAR and the speedometer. We also require traffic light information, which we obtain directly from the CARLA API.

The LiDAR is a rotating ray-casting sensor which provides a cloud of 3D coordinates (x, y, z) of the points and their intensity around the vehicle. This sensor's rotation behaviour gets most of the object's points surrounding the vehicle, but it does not guarantee getting all of them. For example, depending on the FPS and the configured rotation rate, in some cases only the obstacle points in the right side are collected but not some of the left-most points, and vice-versa. Thus, the aforementioned Beliefs Handler is designed to keep a history of these data to be able to get the knowledge about the current right and left obstacles at the same time. In contrast, the speedometer is a simple sensor that provides the vehicle's current speed and does not require any further pre-processing.

There is a vast amount of cloud points coming from the LiDAR data, containing more than 50,000 points at a time. Therefore, analysing and pre-processing the data is necessary to generate useful symbolic data to send to the agent. We apply three pre-processing steps to the LiDAR cloud point: (i) filter the points by limiting the distance (X between -5m to 8m, Y between -4m to 4m, and Z between 0.7m to 2m); (ii) group the remaining points into six directions front (F), back (B), straight front (SF), straight back (SB), left (L) and right (R), as shown in Figure 3; and (iii) get the two closest points (X,Y) in each of the six directions. These three pre-processing steps compress the 50,000+ LiDAR points into only 12 valuable points.

Furthermore, by observing data from the execution of ML models in CARLA, we have determined that many of the critical failure cases occur in traffic light intersections. These situations require "System 2" to take control in order to devise some rational solution. The traffic light information is collected from the CARLA API, and serve as an example of infrastructure-to-car communication.

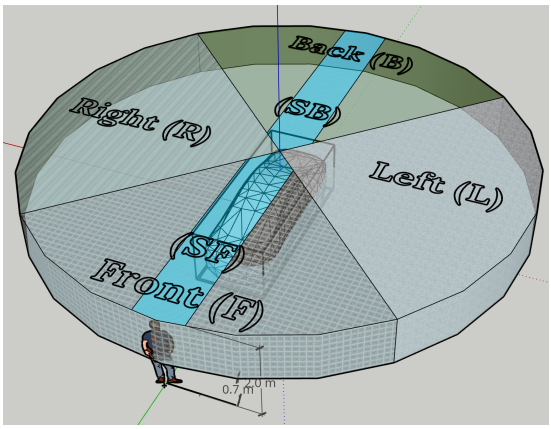


Figure 3: LiDAR cloud points grouped by direction.

The information about the traffic lights is much more simple, but some pre-processing steps are still required to maintain useful symbolic data while at the same not overloading the agent with too

much information. We perform the following three pre-processing steps: (i) get the distance to any traffic light that is up to 15 meters in front of the vehicle (including coordinates of other traffic lights in the same intersection); (ii) calculate if the vehicle is inside the traffic light intersection, as shown in Figure 4; and (iii) stop tracking the traffic light information if the vehicle has exited the intersection.

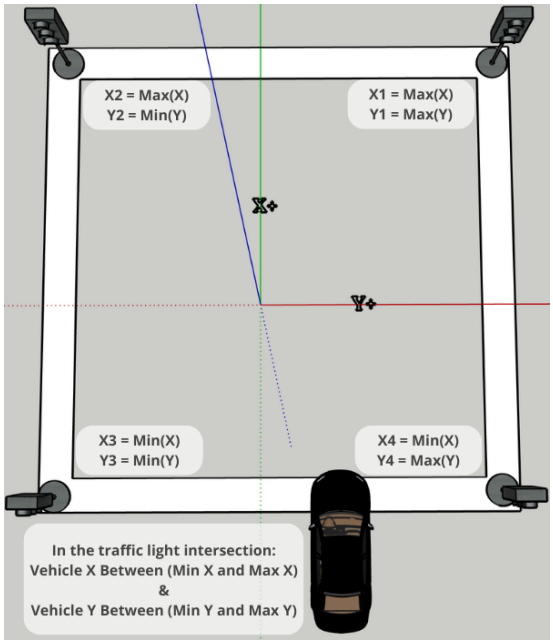


Figure 4: Calculating the traffic light intersection flag. Note the notation for the axes, CARLA considers the front of the vehicle as $X+$ axis, on the right of it is the $Y+$ axis, and the Z axis is the vertical axis.

Besides the pre-processed data from the LiDAR and the traffic lights, as well as the data from the speedometer, basic information about the current frame and step number are also sent to the agent in order to help predict the motion of the obstacles. For example, whether it is moving towards the vehicle or opposite from it, by comparing the distance difference between the information from the previous frame and the current one. This is possible because of the history provided by the Beliefs Handler.

3.4 BDI Agent

Based on the pre-processed data received by the BDI agent, the Beliefs Handler can add the beliefs listed in Table 1, which are then used to trigger the relevant plan. Each belief has the *Frame* number value that is used to keep track of the history (up to four previous frames) of the data and also to be able to predict the obstacles motion by comparing the previous frame to the current frame.

We have identified seven critical situations where System 2 input is required, based on infractions and collisions of the top-ranking ML models in the Leaderboard training routes. For each situation, we implement a plan in the Jason BDI agent. The seven BDI plans (rules) were obtained via experimentation by observing the training routes using various ML models. We identified the scenarios where

Table 1: BDI agent beliefs.

Belief	Description
info(Frame,Speed)	Step number and vehicle speed
f(Frame,X,Y,MinX,MinY)	Front obstacle detected
b(Frame,X,Y,MinX,MinY)	Back obstacle detected
sf(Frame,X,Y,MinX,MinY)	Straight front obstacle detected
sb(Frame,X,Y,MinX,MinY)	Straight back obstacle detected
r(Frame,X,Y,MinX,MinY)	Right obstacle detected
l(Frame,X,Y,MinX,MinY)	Left obstacle detected
traffic_light(Frame,Type, Colour, DifX,DifY, Distance,IsInInt)	Traffic light in front, or inside a traffic light intersection
ml_control(Frame,Throttle,Steer, Brake, Hand_brake,Reverse)	Current ML model action

these models were having problems and then created the rules that would first detect the problem and then attempt to solve it. The seven plans are as follows.[1]

Close crossing collision avoidance. The plan triggers when there are close obstacles on the front, right or left of the vehicle with less than 2 meters distance in the Y axis. Once triggered the plan contains an action to break the vehicle to try to avoid the collision. The Jason plan for this behaviour is shown in Listing 2. The +! sign represents the addition of a goal, in this case the triggering event is the addition of the goal frame(F) with F the Leaderboard step/frame. The commands between : and <– form the context of the plan (i.e., the precondition of the plan). Finally, the commands after <– are a sequence of actions to be executed when the plan is triggered and the context is valid. Figure 5a shows an example of a nearby cyclist about to cross the road.

```
+!frame(F): f(F,X,_,_,MinY) & MinY < 2.0 & X < 4.5 &
    info(F,Sp) & Sp > 0.5
        <- control(2,0.0,0.0,1.0,false,false,(Sp*3)).
              // hard break
```

Listing 2: Jason plan for close crossing collision.

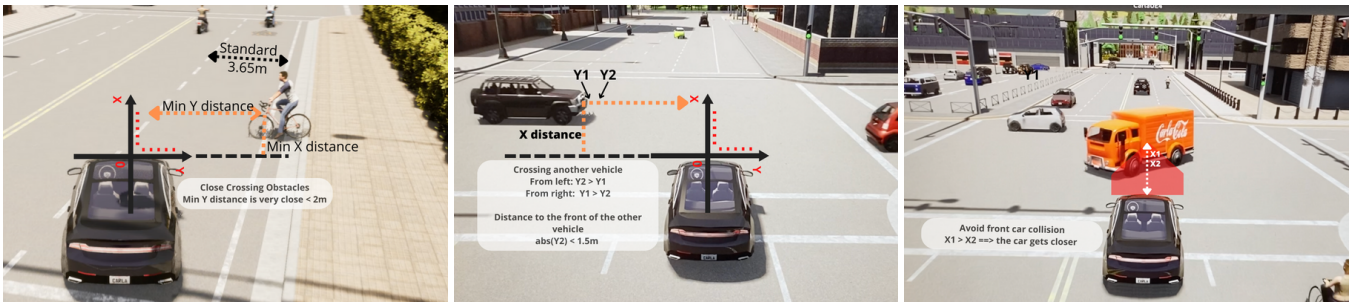
Far crossing collision avoidance. This plan is designed to detect obstacles in the Y axis distance that are expected to come in front of the vehicle. This is calculated based on obstacle motion by comparing the distance differences between the frames and the current vehicle speed. Listing 3 shows parts of this plan. Figure 5b has an example of a fast-moving vehicle in a traffic light intersection that is successfully detected by this plan.

```
+!frame(F): f(F,X,Y,_,MinY) & f(F2,_,Y2,_,MinY2) &
    (F-F2) <= 2 & Y > 0 & Y2 > 0 &  MinY > 2.0 &
    X < 6  & Y < 4 & (MinY2-MinY) > 0.15  &
    ml_control(F,_,St,_,_,_) & St < 0.1 & info(F,Sp)
        <- control(1,0.0,0.0,1.0,false,false,(Sp*3)).
              // hard break
```

Listing 3: Jason plan for far crossing collision.

Front collision avoidance. This plan is used to predict the front obstacles that are getting closer to the vehicle. It observes the front for obstacles from a distance of 7.9 meters, which is enough that if an obstacle is detected then it manages to brake at the proper

[1]Demonstration videos and the code for the experiments are available in https://github.com/alshukairi/MLMAS-Framework-AAMAS23.



(a) Close crossing avoidance. (b) Far crossing avoidance. (c) Front avoidance.

(d) Back avoidance. (e) Traffic light green. (f) Exiting intersection.

Figure 5: Visual representations of the BDI agent plans.

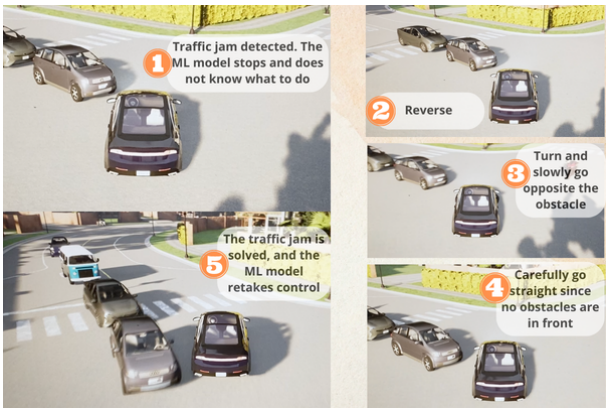


Figure 6: Visual representation of the plan for traffic jams.

time to avoid the incoming vehicle. Figure 5c shows a truck moving towards the vehicle, trying to pass to the other lane.

Back collision avoidance. This plan attempts to avoid such collision by predicting when a fast-moving vehicle is coming from the rear side or when there is another vehicle too close to the rear. Figure 5d has an example of a vehicle coming from the back and trying to pass to another lane. Thus, our plan moves the vehicle forward if possible (considering any front obstacles), providing a safe space for the other vehicle to manoeuvre.

Traffic light green. Sometimes the ML model would take too long to move the vehicle when the traffic light was green and there were no obstacles. This is a simple plan that acts as a wake-up for the ML model to start moving the vehicle because the traffic light turns to green, as shown in Figure 5e.

Exiting intersection. This plan is to ensure that the vehicle exits the traffic light intersection at a reasonable slow speed to allow it to avoid any sudden obstacles. An example is shown in Figure 5f.

Traffic jam navigation. The plan detects that the ML model is stuck in a particular place without moving the vehicle for an extended period of time (60 frames or 3 gaming seconds). This detection mechanism also considers other information that distinguishes it from stopping normally. Once the traffic jam detection is triggered, there are various sub-plans that attempt to move the vehicle out of it. Some options include: reverse, reverse and turn, go forward, go forward and yaw. After some attempts, usually there is a small waiting period to check if the ML model is able to retake control or not. Figure 5f demonstrates a sample of a traffic jam that was solved by using this plan in five steps.

4 EVALUATION

The CARLA Leaderboard evaluation consists of six towns (maps) with a total of 100 secret routes and 76 public routes. The public routes are used for training and testing, while the secret routes are kept hidden and used for the official Leaderboard challenge. Submitting a solution to the Leaderboard is a lengthy process, and even just running the 76 public routes locally can take a very long time. As a shorter alternative, the "Longest6 Benchmark" [10] extends the CARLA Leaderboard with a new set of 36 routes. These routes are based on the six towns from the Leaderboard, therefore any training

done on the original 76 public routes will also be useful for this new set of 36 routes. We use this benchmark for our experiments.

We select the LAV [9] as our pre-trained ML model. Integrating a new ML model with ML-MAS is a seamless process which can be done via a configuration file. The LAV model is a deep learning approach that uses a range of different techniques (e.g., recurrent neural networks, convolutional neural networks) to build a representation of the driving behaviour of not only the vehicle being driven, but also of other vehicles, all while remaining invariant to the viewpoint of the controlling vehicle. Note that the LAV model comes pre-configured with experimentally trained weights. This is not directly comparable with the final weights that achieved a (winning) score of 61% in the Leaderboard challenge of 2020; those weights were not made public at the time of writing this paper. We first run the LAV pre-trained model by itself in the Longest6 benchmark, and then we run ML-MAS using LAV as the ML model.

The Leaderboard evaluator is designed to be independent of either the computer specification or the model processing speed. This is done by using the CARLA game time instead of real system time. The Leaderboard evaluator is designed to run with a fixed 20 frames per gaming time second. This is synchronised with every actor in the environment (vehicles, pedestrians, etc.). Our goal in this paper is to improve the driving score obtained by ML models in the CARLA Leadearboard challenge, nevertheless, we report the configuration of the computer used in the experiments for compatibility purposes: Ubuntu 18.04 operating system, Intel Core i7-6820HK CPU @ 2.70 GHz processor, GeForce GTX 1070 Mobile graphics card, Nvidia-driver-465, and Cuda 11.3.

4.1 Metrics

The CARLA Leaderboard provides metrics designed to measure different aspects of driving. The final *driving score* metric is a multiplication between two aggregation metrics [20]: *route completion* and *infraction penalty*. Route completion is a percentage of the completed distance in a route compared to the route's length. There are five sub-metrics that negatively impact route completion: off-road driving, route deviation, agent blocked (the vehicle is stuck for a period of time), simulation timeout and route timeout. The first is a percentage reduction over the route completion score, and the other four cause the route evaluation to stop once their conditions are met (more details about each condition can be found in [20]). Infraction penalty starts with a base value of 1.0 (higher values are better, i.e., less infractions have been committed). This value is reduced by multiplying it by the coefficient of each of the five road infractions: collisions with pedestrians (0.5), collisions with other vehicles (0.6), collisions with static elements / layout (0.65), running a red light (0.7), and running a stop sign (0.8).

The final driving score is calculated as follows:

$$dirivingScore = routeCompletionPercentage * infractionPenalty$$

We propose an additional seven ML-MAS specific metrics, with each metric representing the contribution percentage of each of the primary BDI agent plans reported in Section 3.4. The orchestrator keeps a record of all of these metrics by counting the number of frames each plan contributes when sending an action (this value includes the repeat action too). At the end of each route evaluation, the results are stored using the following formula to calculate the contribution of each plan:

$$contributionPercentage = \frac{totalPlanContributionFrames}{totalFrames} * 100$$

4.2 Results

Figure 7a contains the main Leaderboard scores, driving score, route completion and infraction penalty. The ML-MAS Framework improved all of the scores, demonstrating that it successfully enhanced decision-making of the LAV model. Substantial improvements were obtained to the driving score (increase of 18.1%) and route completion (increase of 18.7%), as well as a smaller improvement in the infraction penalty (increase of 8.2%), meaning that ML-MAS committed less infractions than the LAV-only solution.

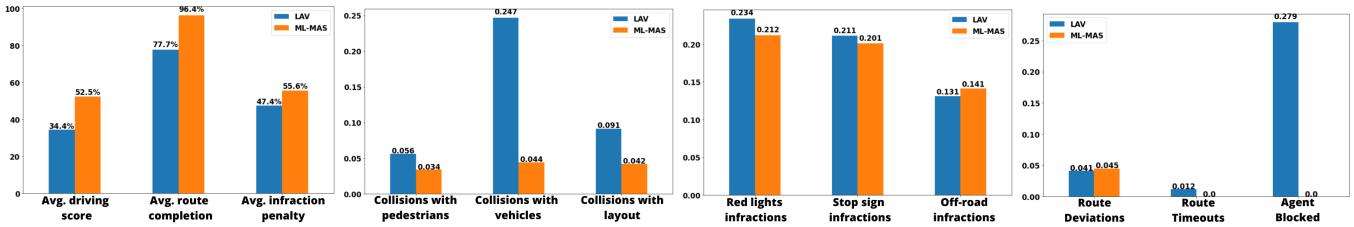
The remaining nine Leaderboard metrics results are displayed in three groups. Group one, in Figure 7b, shows the three collision penalty results, collision with a pedestrian, with a vehicle, and with the layout. ML-MAS reduced the collisions per kilometre by more than half. The most significant reduction gap was in the vehicle collisions, which decreased from 0.247/km to only 0.044/km. The total number of collisions per km for the LAV model was 0.394/km compared to only 0.12/km for ML-MAS.

Group two of the metrics, in Figure 7c, shows a slight improvement in two infractions, the red light and stop sign. There are no main plans for red lights and/or stop signs in the BDI agent, but they are considered in some of the subplans, which may have caused this minor improvement in the metrics. ML-MAS performed more off-road infractions than the LAV model, 0.01/km more to be precise. This occurs because the BDI model considers more complex actions such as reverse and turning opposite to a obstacle, while the ML models are often using only throttle, steer and brake actions. By performing the more complex actions, the BDI agent may choose to commit a minor infraction to avoid a larger one.

In Figure 7d, we have the third group with the route deviation, route timeout and agent blocked metrics. In the route deviation, we have a minor increase (0.004/km) in ML-MAS. This follows the same reasoning from the off-road infractions, the agent will decide to commit them to avoid worse infractions. ML-MAS successfully reduces the route timeout and the agent blocked metrics to zero. The agent blocked score of LAV had a significant impact in their driving score, which was completely solved by ML-MAS. These improvements to route timeout and agent blocked metrics had a very positive effect in the route completion metric for ML-MAS, and were one of the main reasons for having such a large improvement in the driving score over the LAV-only solution.

The ML-MAS metrics results are shown in Figure 8. The plan that was most used to send actions was the Close Crossing Collision Avoidance, with an average of 2.4% to the total number of actions. Despite causing the best improvement in the score, the Traffic Jam Navigation plan is used the least, with 0.3%. The total interference from the rational agent in the evaluated routes is 8.1%, with the remaining 91.9% of the actions being sent by the ML model. This corroborates the expectation that System 1 (LAV) should be acting for most of the time, and System 2 (BDI agent) should interfere only a small percentage of the time to provide rational decisions.

To test the generality of our framework, we have also conducted experiments with a second ML model, the TransFuser model [10].



(a) Main metrics, higher is better.

(b) Group one metrics, normalised per kilometre, lower is better.

(c) Group two metrics, normalised per kilometre, lower is better.

(d) Group three metrics, normalised per kilometre, lower is better.

Figure 7: Leaderboard results for LAV and ML-MAS (with LAV).

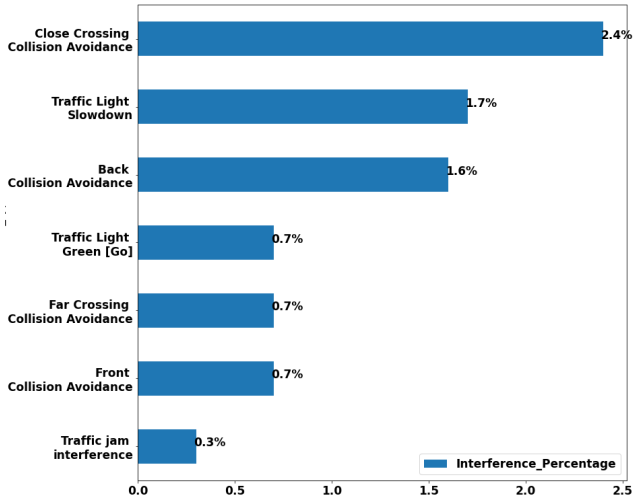


Figure 8: Results of the ML-MAS (with LAV) metrics.

ML-MAS managed to improve the driving score by 3.6% from 44.8% (TransFuser) to 48.4% (ML-MAS). The route completion was again improved by a good margin, 85.9% for TransFuser and 98.2% for ML-MAS. Unfortunately, ML-MAS committed more infractions, obtaining a worse infraction penalty score, 48.9% compared to 51.0% of TransFuser. Given that most plans were made based on observing executions of the LAV model, this minor improvement using TransFuser can be further improved by adding new plans for scenarios that we did not consider before (such as avoiding traffic infractions).

5 DISCUSSION

In this section we discuss the research questions posed at the beginning of the paper and how we solved them in the context of self-driving vehicles. *RQ1 How do we model the governance of System 1 and 2 in an AI?*

Our orchestrator component provides the governance of System 1 and System 2. Its main function is to orchestrate the decisions that are being sent from both systems. It also ensures that both systems receive the required information from the environment when needed. Governance of the actions being sent is done by prioritising which action is passed to the environment for execution. *RQ2 Which factors trigger the switch between the two systems?*

If one of the seven main plans is triggered and its context is valid, then control is switched (prioritised) to the rational agent (System 2). Upon completion of the plan, control returns to System 1.

RQ3 How should System 2 act once the switch is triggered?

Whenever System 2 is triggered, the agent evaluates the latest beliefs received along with the memorised history of previous frames, and then it either activates one of plans which will result in an action being sent or it decides to not perform any action.

RQ4 When should problems be handed back from System 2 to 1?

Control should be handed back once no plans from System 2 are being executed. This includes the repeat action available in the BDI agent. For example, in a collision scenario where the vehicle is in high speed, the agent can send a brake action to be repeated many times proportional to the vehicle speed, and only after the action has been repeated the control will be handed back to System 1.

6 CONCLUSION

ML-MAS combines ML and symbolic decision-making. Our results successfully demonstrate that this combination between a pre-trained ML model and a BDI agent can have a significant impact in the driving score of autonomous vehicles. Without altering the ML model in any way, we added seven plans that our rational agent used to avoid infractions and improve the driving score of the original ML solution. The results for using different ML models will depend on the generality and effectiveness of the BDI plans and the effectiveness of the original model.

For future work, in terms of implementation, we would like to improve the interfaces of the framework (BDI bridge and orchestrator) to allow other researchers to easily plug their favourite BDI language and ML model. From a more theoretical perspective, we wish to explore communication between the ML model and the BDI agent, so that the process of selecting an action is a result of a direct deliberation between both components. Our methodology for identifying critical situations (i.e., when we need to devise a BDI plan) was ad-hoc and based on experimentation, to make it more generalisable we plan to investigate the literature in self-driving cars to identify more general situations. To make this technique realisable in the real world, further experiments considering computation speed and response time need to be performed. Finally, in terms of applications, we firmly believe that similar ideas can be applied to other domains that can be represented as System 1 and System 2, such as in the case of robotics (more specifically, search and rescue and other emergency use of robots).

REFERENCES

[1] Mohammed Al-Nuaimi, Sapto Wibowo, Hongyang Qu, Jonathan Aitken, and Sandor Veres. 2021. Hybrid verification technique for decision-making of self-driving vehicles. *Journal of Sensor and Actuator Networks* 10, 3 (2021), 42.

[2] Tarek R. Besold, Artur S. d'Avila Garcez, Sebastian Bader, Howard Bowman, Pedro M. Domingos, Pascal Hitzler, Kai-Uwe Kühnberger, Luís C. Lamb, Priscila Machado Vieira Lima, Leo de Penning, Gadi Pinkas, Hoifung Poon, and Gerson Zaverucha. 2021. Neural-Symbolic Learning and Reasoning: A Survey and Interpretation. In *Neuro-Symbolic Artificial Intelligence: The State of the Art*, Pascal Hitzler and Md. Kamruzzaman Sarker (Eds.). Frontiers in Artificial Intelligence and Applications, Vol. 342. IOS Press, 1–51. https://doi.org/10.3233/FAIA210348

[3] Grady Booch, Francesco Fabiano, Lior Horesh, Kiran Kate, Jonathan Lenchner, Nick Linck, Andreas Loreggia, Keerthiram Murgesan, Nicholas Mattei, Francesca Rossi, and Biplav Srivastava. 2021. Thinking Fast and Slow in AI. *Proceedings of the AAAI Conference on Artificial Intelligence* 35, 17 (May 2021), 15042–15046. https://doi.org/10.1609/aaai.v35i17.17765

[4] Rafael H. Bordini, Amal El Fallah Seghrouchni, Koen V. Hindriks, Brian Logan, and Alessandro Ricci. 2020. Agent programming in the cognitive era. *Auton. Agents Multi Agent Syst.* 34, 2 (2020), 37. https://doi.org/10.1007/s10458-020-09453-y

[5] Rafael H. Bordini, Michael Wooldridge, and Jomi Fred Hübner. 2007. *Programming Multi-Agent Systems in AgentSpeak using Jason.* John Wiley & Sons, Chichester, UK.

[6] M. E. Bratman. 1987. *Intentions, Plans, and Practical Reason.* Harvard University Press.

[7] Rodney A. Brooks. 1991. Intelligence without representation. *Artificial Intelligence* 47, 1 (1991), 139–159. https://doi.org/10.1016/0004-3702(91)90053-M

[8] Rafael C. Cardoso and Angelo Ferrando. 2021. A Review of Agent-Based Programming for Multi-Agent Systems. *Computers* 10, 2 (Jan 2021), 16. https://doi.org/10.3390/computers10020016

[9] Dian Chen and Philipp Krähenbühl. 2022. Learning from All Vehicles. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* IEEE, 17201–17210. https://doi.org/10.1109/CVPR52688.2022.01671

[10] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. 2022. TransFuser: Imitation with Transformer-Based Sensor Fusion for Autonomous Driving. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2022), 1–18. https://doi.org/10.1109/TPAMI.2022.3200245

[11] Paolo Coraggio and Massimo De Gregorio. 2007. A Neurosymbolic Hybrid Approach for Landmark Recognition and Robot Localization. In *Proceedings of the 2nd International Conference on Advances in Brain, Vision and Artificial Intelligence* (Naples, Italy) *(BVAI'07).* Springer-Verlag, Berlin, Heidelberg, 566–575.

[12] J.M. Corchado and J. Aiken. 2002. Hybrid artificial intelligence methods in oceanographic forecast models. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 32, 4 (2002), 307–313. https://doi.org/10.1109/TSMCC.2002.806072

[13] Mike Daily, Swarup Medasani, Reinhold Behringer, and Mohan Trivedi. 2017. Self-Driving Cars. *Computer* 50, 12 (2017), 18–23. https://doi.org/10.1109/MC.2017.4451204

[14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning.* 1–16.

[15] Massimo DE GREGORIO. 2008. An Intelligent Active Video Surveillance System Based on the Integration of Virtual Neural Sensors and BDI Agents. *IEICE Transactions on Information and Systems* E91.D, 7 (2008), 1914–1921. https://doi.org/10.1093/ietisy/e91-d.7.1914

[16] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. 2020. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics* 37, 3 (2020), 362–386. https://doi.org/10.1002/rob.21918 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21918

[17] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A Survey of Safety and Trustworthiness of Deep Neural Networks: Verification, Testing, Adversarial Attack and Defence, and Interpretability. *Computer Science Review* 37 (2020), 100270. http://www.sciencedirect.com/science/article/pii/S1574013719302527

[18] Daniel Kahneman. 2012. *Thinking, Fast and Slow.* Penguin Books, Harlow, England.

[19] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification*, Ganesh Gopalakrishnan and Shaz Qadeer (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 585–591.

[20] CARLA Leaderboard. 2022. CARLA Autonomous Driving Leaderboard. https://leaderboard.carla.org/. [Online; accessed 25-June-2022].

[21] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. 2009. MCMAS: A Model Checker for the Verification of Multi-Agent Systems, In MCMAS: A Model Checker for the Verification of Multi-Agent Systems. *International Journal on Software Tools for Technology Transfer* 5643, 682–688. https://doi.org/10.1007/978-3-642-02658-4_55

[22] Niccolò Piazzesi, Massimo Hong, and Andrea Ceccarelli. 2021. Attack and Fault Injection in Self-driving Agents on the Carla Simulator – Experience Report. In *Computer Safety, Reliability, and Security*, Ibrahim Habli, Mark Sujan, and Friedemann Bitsch (Eds.). Springer International Publishing, Cham, 210–225.

[23] A. S. Rao and M. Georgeff. 1995. BDI Agents: From Theory to Practice. In *Proceedings of the first International Conference on Multi-Agent Systems.* San Francisco, USA, 312–319.

[24] Md. Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. 2021. Neuro-symbolic artificial intelligence. *AI Commun.* 34, 3 (2021), 197–209. https://doi.org/10.3233/AIC-210084

[25] Zachary Susskind, Bryce Arden, Lizy K. John, Patrick Stockton, and Eugene B. John. 2021. Neuro-Symbolic AI: An Emerging Class of AI Workloads and their Characterization. https://doi.org/10.48550/ARXIV.2109.06133

[26] Ray Tsaih, Yenshan Hsu, and Charles C. Lai. 1998. Forecasting S&P 500 stock index futures with a hybrid AI system. *Decision Support Systems* 23, 2 (1998), 161–174. https://doi.org/10.1016/S0167-9236(98)00028-1

[27] Letian Wang, Yeping Hu, Liting Sun, Wei Zhan, Masayoshi Tomizuka, and Changliu Liu. 2021. Hierarchical adaptable and transferable networks (HATN) for driving behavior prediction. In *NeurIPS 2021, Machine Learning for Autonomous Driving Workshop.*

[28] Nanning Zheng, Shaoyi Du, Jianji Wang, He Zhang, Wenting Cui, Zijian Kang, Tao Yang, Bin Lou, Yuting Chi, Hong Long, Mei Ma, Qi Yuan, Shupei Zhang, Dong Zhang, Feng Ye, and Jingmin Xin. 2020. Predicting COVID-19 in China Using Hybrid AI Model. *IEEE Transactions on Cybernetics* 50, 7 (2020), 2891–2904. https://doi.org/10.1109/TCYB.2020.2990162