# Overcooked Plus: A Comprehensive Cooking Scenario TestBed for Enhancing the Evaluation of Autonomous Planning Algorithms

Jinyu Cai
*Waseda University*
Tokyo, Japan
bluelink@toki.waseda.jp

Jialong Li
*Waseda University*
Tokyo, Japan
lijialong@fuji.waseda.jp

Nianyu Li
*ZGC Laboratory*
Beijing, China
li_nianyu@pku.edu.cn

Mingyue Zhang
*Southwest University*
Chongqing, China
myzhangswu@swu.edu.cn

Ruijia Yang
*Waseda University*
Tokyo, Japan
zuika@fuji.waseda.jp

Kenji Tei
*Tokyo Institute of Technology*
Tokyo, Japan
tei@c.titech.ac.jp

***Abstract*—This paper introduces Overcooked Plus, a testbed designed to evaluate the performance of autonomous systems in complex and dynamic cooking scenarios. In these scenarios, single or multiple-agent systems must complete various cooking tasks such as fetching ingredients, handling kitchen utensils, chopping, cooking, and dishwashing. The testbed aims to offer a customizable platform that supports comprehensive evaluation of planning algorithms and adaptive behaviors within autonomous systems. Key features of Overcooked Plus include controllable task difficulty, complex task constraints, multi-task planning, dynamic environments, a graphical interface for human-on-the-loop, and a communication interface for multiagent systems. The paper details the system architecture and implementation of Overcooked Plus, and presents experimental setups to demonstrate its usage.**
**GitHub: github.com/545659928/Overcooked-Plus**

***Index Terms*—Cooking Scenario, Planning Algorithms, Multi-agent System, Benchmark, Testbed**

## I. Introduction

Autonomous and self-adaptive systems, that can operate and make decisions under uncertain conditions, have been widely studied driven by the need for automation. The application scenarios for autonomous systems cover various sectors, including industrial applications like drone control [1], warehouse automation [2], traffic management [3], autonomous driving [4], and home automation such as smart kitchens [5] and cleaning robots [6].

To evaluate the performance and capabilities of these systems and their algorithms, various testing platforms and benchmarks have been developed to simulate complex and dynamic environments. One major category includes platforms like OpenAI's Gym [7], which offers a variety of standardized environments such as the Atari game suite for assessing agents' planning capabilities and robotic manipulation tasks for assessing control algorithms. Similarly, MuJoCo [8] provides physics-based environments for evaluating control algorithms in simulated robotics and locomotion tasks. Unity's ML-Agents toolkit [9] also stands out, providing a flexible platform for creating complex, 3D environments that support a wide range of reinforcement learning tasks. Additionally, robotic simulation environments like the Robot Operating System (ROS) combined with Gazebo [10], allow for realistic robotic simulations in physics-based settings.

For agent-based systems, grid-based platforms such as Flatland [11] and Gridworld [12] provide structured settings for agents to navigate and interact, often used to evaluate path-finding [13], [14] and planning algorithms. Additionally, platforms like Multi-Agent Particle Environments (MPE) [15] offer simple yet effective scenarios to study interactions between multiple agents. Another significant category includes traffic flow simulation environments, with a representative tool SUMO (Simulation of Urban MObility) [16], which is widely used for the studies of vehicles and their interactions in urban settings.

One promising domain for evaluating autonomous systems is the cooking scenario, in which single or multi-agent systems (along with human players) interact with the environment to acquire ingredients at specific locations and complete cooking tasks in a specific procedure. This cooking scenario is not only popular in video gaming (e.g., Overcooked, Cooking Mama) but also has practical applications in industrial (e.g., restaurants or food processing factories) and home settings [17]. Given its popularity and practicality, several studies, particularly in the field of Reinforcement Learning (RL) [18], [19], have explored cooking scenarios to evaluate planning algorithms of autonomous systems. However, this exploration remains in the preliminary stages, often involving simple recipes (e.g., salad) and static environments. This simplification

Corresponding Author: Jialong Li

hinders the comprehensive evaluation of the decision-making capabilities within autonomous systems, as it lacks the varied challenges introduced by dynamic environments and complex constraints.

To address these limitations, this paper introduces Overcooked Plus [1], a cooking scenario testbed designed to provide a customizable and comprehensive evaluation platform for planning algorithms within autonomous systems. Specifically, we redevelop the open-source implementations [21] to incorporate a variety of new features and improvements, facilitating a more comprehensive assessment of autonomous systems. The specific enhancements and contributions are as follows:

- Controllable Task Difficulty and New Cooking Methods: Overcooked Plus offers three different difficulty levels, ranging from simple single-ingredient salads to complex, long-horizon multi-ingredient recipes. It introduces cooking methods like frying pans that do not require continuous interaction, enhancing the assessment of multitasking and task scheduling.
- Task Constraints: Overcooked Plus introduces several complex additional constraints, such as "steak must not be cooked for more than 20 timesteps or it will burn, causing the task to fail," "burnt food must be disposed of," and "dishes must be washed before they can be used to serve food." These constraints enhance the evaluation of forward planning and fault handling in autonomous systems.
- Multi-task Planning: Unlike existing naive testbeds, Overcooked Plus allows multiple concurrent tasks, requiring adaptive systems to navigate and prioritize tasks efficiently for maximum performance.
- Dynamic Environments: Overcooked Plus introduces dynamic kitchen environments where layouts change over time and certain areas and equipment may become temporarily inaccessible, thereby enhancing the evaluation of adaptability to environmental changes.
- Human-Player Integration: Overcooked Plus strengthens the graphical user interface (GUI) and supports human-player integration, allowing for effective validation of adaptive systems in human-machine collaboration (cooperation) or robustness (adversarial) scenarios.
- Communication interface: Overcooked Plus provides a basic communication interface, facilitating customization under various communication conditions, such as communication range and failure rate, potentially promoting the evaluation of multi-agent systems.

The rest of the paper is organized as follows: Section II provides an overview of the cooking scenarios of Overcooked Plus. Section III briefly introduces the system architecture of Overcooked Plus, explaining the high-level structure and interactions of its components. Section IV delves into the technical implementation of each component within the system, including data structures and algorithms used. Section V presents our experimental setup, describing a specific scenario to demonstrate how Overcooked Plus can be configured and run. Finally, Section VI concludes the paper and discusses future work.

[1]The term "Overcooked" is derived from the well-known cooking video game [20]

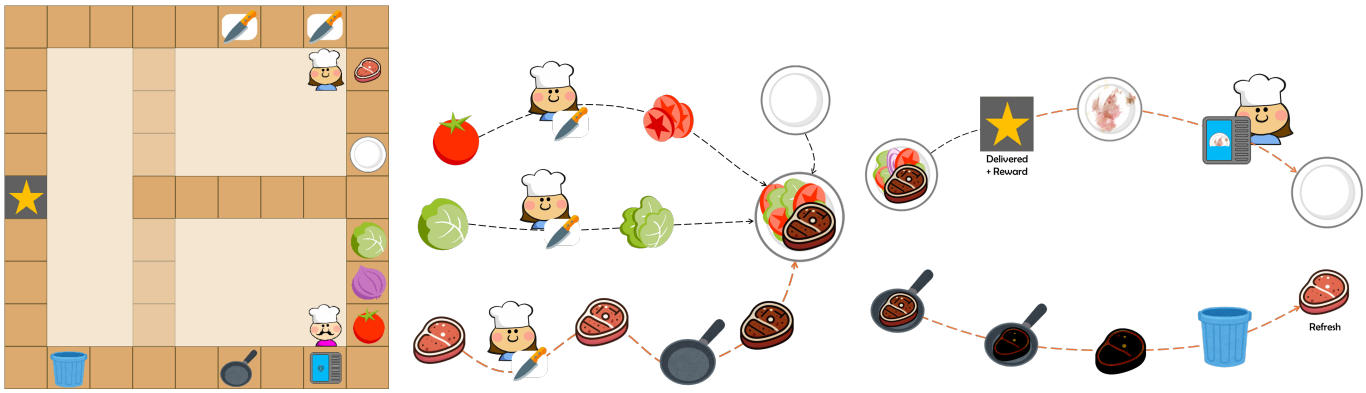
## II. Cooking Scenario Overview

Figure 1 summarizes a cooking scenario, featuring a restaurant kitchen with one or more agents, ingredients, kitchenware, and beige-colored tables. In this scenario, tables act as obstacles that agents cannot pass through but can use to place or pick up items. Existing study [21] has involved simple tasks like making vegetable salads, where agents chopped tomatoes and onions on cutting boards and then delivered these ingredients using plates.

In this study, we introduce new types of ingredients, corresponding kitchen tools, and rules. Both vegetables and the newly added meat can be chopped on cutting boards through repeated interactions. However, only meat can be cooked on frying pans. Unlike cutting boards, cooking on a frying pan progresses automatically over time without agent interaction. As shown in Fig. 1c, if meat is left on the pan for too long, it will burn and become inedible, requiring it to be discarded in the trash bin to reset its state. Additionally, plates become dirty after a dish is delivered. Dirty plates cannot hold food and must be cleaned in the sink before reuse.

Furthermore, the translucent grids in the figure represent new dynamic map elements that block movement in the upper or lower parts of the kitchen as time progresses. The upper area contains cutting boards, plates, and steaks, while the lower area contains vegetables, frying pans, and sinks. Both areas contain essential items needed to complete tasks. In this dynamic environment, autonomous systems must plan ahead, coordinating tasks such as passing ingredients between agents or altering strategies when certain items become temporarily inaccessible to maintain overall efficiency.

The evaluation process is customizable, but the default setting is as follows: The testbed initially provides a list of tasks, each corresponding to a specific dish. Agents must process various ingredients to complete the selected tasks. For example, in the task shown in 1b, agents might need to prepare a dish requiring a chopped tomato, a chopped lettuce, and a cooked steak on a plate. The agents must cook the steak on the frying pan, chop the tomato and lettuce on the cutting boards, and clean dirty plates in the sink as needed. Once these steps are completed, the ingredients can be assembled on a plate and delivered. Upon task completion, the system refreshes resources and updates the task list. The evaluation ends when a specified time step limit is reached.




(a) Game Image (b) Task Example (c) Rules for interactions

Fig. 1: Overview of Overcooked Plus. Black lines represent existing features from the open-source project, while orange lines indicate new features introduced in this study. In (a), blocks marked with a star represent submission stations. The semi-transparent blocks indicate dynamic obstacle areas, which alternately block the upper and lower regions as the environment progresses.

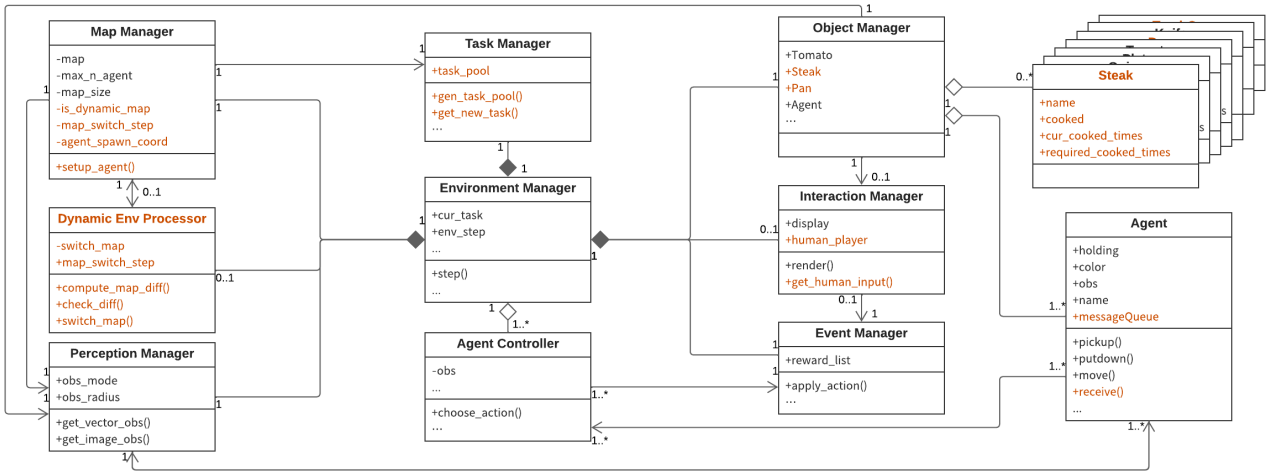



Fig. 2: Implementation Overview of Overcooked Plus. Orange text represents newly added elements in this study.

## III. Implementation Overview

### A. Key Components

Overcooked Plus comprises following key components:

- Environment Manager: The central controller responsible for initializing and managing the entire environment. It updates the environment's state, drives the main loop, and ensures smooth interaction among various components.
- Object Manager: Manages all objects within the environment, including agents, ingredients, and kitchenware. It handles their internal states and available action functions.
- Perception Manager: Generates observation information for each agent and returns it in different forms based on the environment settings.
- Interaction Manager: Outputs visualizations and supports human player inputs.
- Agent Controller: Defined by researchers, responsible for outputting corresponding actions based on observations of the environment.
- Event Manager: Calculates the environmental changes resulting from actions taken by the agents, determines triggered events within the environment, and assesses rewards.
- Dynamic Environment Processor: Manages dynamic aspects of the environment, such as layout changes.
- Map Manager: Manages and stores predefined maps.
- Task Manager: Randomly generates task recipes based on the environment's content.

### B. Component Interactions

We introduce the interactions between components that are summarized in Fig. 2. At the start, the Environment Manager initializes the environment by requesting

the corresponding map information from the Map Manager based on the environment configuration. The Object Manager then creates all objects within the environment according to the map information and requests the current environment tasks from the Task Manager. Finally, the Environment Manager establishes the modules to be used in the main loop according to the environment parameters, completing the initialization process.

After initialization, the main loop begins. The Perception Manager generates observations for each agent based on their field of view settings and all objects present on the map. If enabled, the Interaction Manager renders visualizations and continuously captures user inputs as actions for specific agents. The Agent Controller generates actions for each agent based on their observations. Subsequently, the Event Manager calculates the environmental changes resulting from the agents' actions, determines the occurrence of events within the environment, and returns the corresponding rewards. If the dynamic environment is enabled, the Dynamic Environment Processor assesses whether the map configuration needs to be switched at the current time step, and updates the layout accordingly. The Object Manager updates the state changes for all objects for the current time step. The Environment Manager then proceeds to the next iteration of the main loop.

## IV. Implementation Details

Referencing Fig. 2, this section introduces the implementation details of each component.

**Environment Manager** serves as the central controller of the system, responsible for coordinating all modules and driving the environment's progression. During initialization, it requests map information from the Map Manager and synchronizes the environment. It creates objects using the Object Manager and determines the modules needed for the main loop (e.g., Interaction Manager, Dynamic Environment Processor) based on the environment settings. The main loop ensures the environment operates correctly.

**Object Manager** handles the internal states of all objects within the scene. During initialization, it creates objects for each item on the map and records their coordinates and functional attributes. Ingredients have specific properties such as state and required processing steps. Kitchenware items have unique functions like *hold*(), *chop*(), and *cook*(). Agents have attributes like coordinates and observation history, and actions including *pickup*(), *putdown*(), *move*(), and *receive_message*(). Researchers can customize items via the items file. This paper adds various new items with unique features. For example, the new frying pan can cook the newly introduced meat, which may become unusable if overcooked. The kitchen layouts may change over time and certain areas and equipment may become temporarily inaccessible. These new features increase the task's complexity and environment's dynamics, challenging the systems' adaptability in complex settings.

**Perception Manager** processes agent observations, supporting both vector and RGB image formats. In vector mode, it captures the map within the agent's field of view, records object positions and states, and converts them into vectors with one-hot encoding. In RGB mode, it generates an RGB matrix of the environment scene. If agent communication is enabled, observations include communication content from other agents.

**Interaction Manager** is implemented using Python's Pygame library and facilitates human interaction and visualization. It creates the environment display window, renders the scene, and maps user inputs to agent actions. In Overcooked Plus, introducing human players adds more uncertainty. Human players can be cooperative or adversarial. In a cooperative setting, the adaptive system must dynamically adjust its planning to optimize collaboration efficiency. In an adversarial setting, the system must demonstrate robustness and adaptability to counteract obstructive actions by human players. Rendering is based on the current state and PNG files from the graphics folder. Researchers can customize the visual output by modifying files in the render folder.

**Agent Controller**, primarily defined by researchers, supports custom planning algorithms to validate adaptive systems. It determines agent actions based on observations and should be capable of generating planned actions from observations in vector or RGB mode.Overcooked Plus provides a default Reinfocement Learning(RL)-based planner.

**Event Manager** calculates the effects of agent actions, interpreting inputs as specific actions such as movement or interaction. It applies these actions, resulting in environmental changes, updates object states, and returns reward values based on predefined functions.

**Dynamic Environment Processor** manages map switching. If dynamic environments are enabled, it switches maps at specified time steps from preset configurations. It calculates the differences between maps and ensures the switch does not conflict with current objects in the environment, delaying the switch if necessary.

**Map and Task Managers** Maps are stored in YAML format in the maps folder, allowing researchers to easily design and apply maps. As shown in Listing 1, map information consists of integer arrays, each representing a specific object. YAML files also include environment settings strongly associated with the map, such as map size, dynamic settings, and switching frequency. The Task Manager updates the task pool based on available ingredients on different maps, combining them with environment rule settings. Tasks are randomly selected from the pool for each request.

```
# ITEMIDX values represent the following:
#   space: 0, counter: 1, agent: 2, tomato: 3, lettuce: 4,
#   plate: 5, knife: 6, delivery: 7, onion: 8, pan: 9,
#   steak: 10, sink: 11, trash_can: 12
name: MapC
dimensions: [9, 9]
game_map:
```

```
  - [1, 1, 1, 1, 1, 6, 1, 6, 1]
  - [1, 0, 0, 1, 0, 0, 0, 0, 10]
  - [1, 0, 0, 1, 0, 0, 0, 0, 1]
  - [1, 0, 0, 1, 0, 0, 0, 0, 5]
  - [7, 0, 0, 1, 1, 1, 1, 1, 1]
  - [1, 0, 0, 0, 0, 0, 0, 0, 4]
  - [1, 0, 0, 0, 0, 0, 0, 0, 8]
  - [1, 0, 0, 0, 0, 0, 0, 0, 3]
  - [1, 12, 1, 1, 1, 9, 1, 11, 1]
max_n_agnet: 3
is_dynamic_map: True
switch_map:
  - [1, 1, 1, 1, 1, 6, 1, 6, 1]
  - [1, 0, 0, 0, 0, 0, 0, 0, 10]
  - [1, 0, 0, 0, 0, 0, 0, 0, 1]
  - [1, 0, 0, 0, 0, 0, 0, 0, 5]
  - [7, 0, 0, 1, 1, 1, 1, 1, 1]
  - [1, 0, 0, 1, 0, 0, 0, 0, 4]
  - [1, 0, 0, 1, 0, 0, 0, 0, 8]
  - [1, 0, 0, 1, 0, 0, 0, 0, 3]
  - [1, 12, 1, 1, 1, 9, 1, 11, 1]
map_switch_step: 7
agent_spawn_points: [[1, 7], [7, 7], [1, 4]]
```

Listing 1: Example of map config

## V. Example of Using Overcooked Plus

### A. Built-in Scenarios

Although Overcooked Plus supports customizing scenarios. it also includes three built-in scenarios, that vary in difficulty levels, to facilitate out-of-the-box usability.

- Easy Scenario: Designed to assess the sequential planning capability in simple tasks, this scenario includes tasks with recipes requiring only one type of vegetable. Kitchenware is limited to cutting boards, and plates do not need to be cleaned after task submission.
- Medium Scenario: To verify the temporal planning and fault-handling abilities, this preset includes meat and related kitchenware. Each task contains at least one type of meat and one type of vegetable, with a total of 2-3 required ingredients. The adaptive system must correctly sequence ingredient processing to maximize efficiency and incorporate fault-handling decisions, such as dealing with burned meat.
- Hard Scenario: This highly challenging preset requires tasks with at least three ingredients. Plates become dirty after each submission and must be washed in the sink before reuse. The map configuration is dynamic; after a certain number of time steps, obstacles change, temporarily blocking access to certain areas and kitchenware.

Each preset can be configured for multi-agent mode or include human players, enabling researchers to evaluate adaptive systems from different perspectives.

### B. Customizable Configurations for Scenario Design

```python
import gym

rewardList = {"subtask finished": 10,
"correct delivery": 200, "wrong delivery": -5,
"step penalty": -0.1, "burned penalty": -2}
```

```python
env = gym.make("Overcooked-Plus", map_name='mapC',
               dynamic_map=True,
               rewardList=rewardList, n_agent=2,
               agent_communication=False, obs_radius=2,
               mode='vector', GUI=False)
# Quickly create the environment using presets
# env = gym.make('Overcooked Plus', preset="medium",
    n_agent=2, human_player=False)

obs = env.reset()
done = False

# Researcher-specified adaptive system planners to be
    tested
agent_planner1 = RLplanner(env, config)
agent_planner2 = RLplanner(env, config)
planners = [agent_planner1, agent_planner2]

# Main simulation loop
while not done:
    actions = []
    for i in range(n_agent):
        actions.append(planners[i].get_action(obs[i]))
    new_obs, rewards, done, info = env.step(actions)
    for i in range(n_agent):
        planners[i].update(obs[i], rewards[i], done)
    obs = new_obs
```

Listing 2: Example of the evaluation settings

Overcooked Plus is primarily implemented using Python's gym library. Researchers can create a testbed by using the *gym.make*() function. Besides the presets introduced in the previous section, the testbed is highly customizable, allowing researchers to tailor their evaluation. Listing 2 provides a simple example of creating an experimental environment using Overcooked Plus.

The code begins by importing the gym library and defining the reward structure and the number of agents. The *rewardList* dictionary specifies the points awarded or penalized for various events, such as completing subtasks, correct and incorrect deliveries, burning food, and step penalties.

Next, researchers can choose to initialize Overcooked Plus using presets or specific settings. By calling the *gym.make*() function, parameters such as map type, dynamic map settings, reward structure, number of agents, agent communication, observation radius, observation mode (vector mode in this case), GUI settings, human player settings, and debug mode are passed to configure the environment.

After initializing the environment, the state is reset using *env.reset*(), which also retrieves the initial observations. The *done* flag is set to False, indicating that the simulation loop should continue running.

Researchers then initialize the planning algorithm to be evaluated. The planning algorithm must be capable of taking observations as input and outputting the action index to be executed. The number of planners should match the number of agents. In this example, RL planners are used for the agents, and these planners are stored in a list.

Within the loop, actions for each agent are determined based on their current observations using the *get_action* method of the RL planners. These actions are executed in the environment using *env.step*(*actions*), which returns the new observations, rewards, *done* flag, and additional information, which are then used to update the planners. The loop continues until the environment ends.

### C. Experiment Result Analysis

Overcooked Plus supports recording various types of data, enabling researchers to effectively evaluate the performance of autonomous agents. The following are key types of data analysis supported by the environment.

*1) Performance Metrics:* Overcooked Plus provides several performance metrics to assess the effectiveness of agent behaviors. Key metrics include:

- Task Completion Rate: The percentage of tasks successfully completed by the agents within a given time frame.
- Cumulative Reward: The progression of total rewards accumulated by the agents during the experiment.
- Average Reward Trend: The average reward earned by the agents over time.
- Time Efficiency: The time steps taken by the agents to complete tasks, used to evaluate changes in efficiency.

*2) Agent Behavior Analysis:* Researchers can analyze the behaviors of agents to understand their decision-making processes and interaction patterns. These include:

- Action Distribution: The frequency of various actions taken by the agents (e.g., moving, picking up ingredients, chopping).
- Heatmap: Visualization of the most frequently visited areas on the map by the agents, highlighting their movement patterns and focus areas.

## VI. Conclusion and Future Work

In this paper, we introduced Overcooked Plus, a testbed designed to evaluate autonomous agents in complex and dynamic cooking scenarios. Our testbed provides a comprehensive, flexible, and scalable tool supporting diverse tasks and agent interactions. We detailed the system architecture, component details, and provided an example scenario demonstrating Overcooked Plus's usability and customization.

Future work will focus on enhancing human-agent interaction models to better evaluate collaboration in mixed-agent environments. Improving communication interfaces will support more nuanced agent interactions, leading to better cooperation and efficiency. Additionally, integrating natural language-based task descriptions to further support the evaluation of large language model (LLM)-based methods [22].

## References


[1] E. Frazzoli, J. Enright, M. Pavone, and K. Savla, "Uav routing and coordination in stochastic, dynamic environments," pp. 2079–2109, 2015.

[2] A. Dömel, S. Kriegel, M. Kaßecker *et al.*, "Toward fully autonomous mobile manipulation for industrial environments," *International Journal of Advanced Robotic Systems*, vol. 14, no. 4, p. 1729881417718588, 2017.

[3] I. Rubin, A. Baiocchi, Y. Sunyoto, and I. Turcanu, "Traffic management and networking for autonomous vehicular highway systems," *Ad Hoc Networks*, vol. 83, pp. 125–148, 2019.

[4] L. Liu, S. Lu, R. Zhong, B. Wu, Y. Yao, Q. Zhang, and W. Shi, "Computing systems for autonomous driving: State of the art and challenges," *IEEE Internet of Things Journal*, vol. 8, no. 8, pp. 6469–6486, 2020.

[5] L. Chen, C. D. Nugent, and H. Wang, "A knowledge-driven approach to activity recognition in smart homes," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 6, pp. 961–974, 2012.

[6] D. M. Martínez, G. Alenyà, and C. Torras, "Planning robot manipulation to clean planar surfaces," *Eng. Appl. Artif. Intell.*, vol. 39, pp. 23–32, 2015.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider *et al.*, "Openai gym," 2016.

[8] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[9] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2020.

[10] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[11] J. Li, Z. Chen, Y. Zheng, S.-H. Chan *et al.*, "Scalable rail planning and replanning: Winning the 2020 flatland challenge," in *Proceedings of the international conference on automated planning and scheduling*, vol. 31, 2021, pp. 477–485.

[12] J. Cai, J. Li, M. Zhang, and K. Tei, "Value iteration networks with gated summarization module," *IEEE Access*, vol. 11, pp. 60 407–60 420, 2023.

[13] J. Li, M. Zhang, Z. Mao, H. Zhao *et al.*, "Goal-oriented knowledge reuse via curriculum evolution for reinforcement learning-based adaptation," in *2022 29th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2022, pp. 189–198.

[14] M. Zhang, J. Li, H. Zhao *et al.*, "A meta reinforcement learning-based approach for self-adaptive system," in *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 2021, pp. 1–10.

[15] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *arXiv preprint arXiv:1703.04908*, 2017.

[16] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann *et al.*, "Microscopic traffic simulation using sumo," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2575–2582.

[17] B. Sharath, R. Srisha, K. Shashidhar, and S. S. Bharadwaj, "Intelligent and smart cloud based autonomous robotic kitchen system," *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pp. 601–606, 2018.

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

[19] W. Tan, W. Zhang, S. Liu, L. Zheng, X. Wang, and B. An, "True knowledge comes from practice: Aligning large language models with embodied environments via reinforcement learning," *arXiv preprint arXiv:2401.14151*, 2024.

[20] G. T. Games, "Overcooked," 2016. [Online]. Available: https://store.steampowered.com/app/448510/Overcooked/

[21] S. A. Wu, R. E. Wang, J. A. Evans, J. B. Tenenbaum *et al.*, "Too many cooks: Coordinating multi-agent collaboration through inverse planning," *Topics in Cognitive Science*, 2021.

[22] J. Li, M. Zhang, N. Li, D. Weyns, Z. Jin, and K. Tei, "Exploring the potential of large language models in self-adaptive systems," in *Proceedings of the 19th Conference on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, April 2024.