SoftwareX 28 (2024) 101956

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

MLPro-GT-game theory and dynamic games modeling in Python

Steve Yuwono [*], Detlef Arend, Andreas Schwung

Department of Automation Technology and Learning Systems, South Westphalia University of Applied Sciences, Lübecker Ring 2, Soest, 59494, Germany

ARTICLE INFO

Keywords:
Game theory
Dynamic games
Decision-making
Nash equilibrium
Python
MLPro

ABSTRACT

Game theory, a fundamental aspect of mathematical economics and strategic decision-making, has been increasingly applied to various fields, including economics, biology, computer science, and engineering. Despite its growing importance, there is a significant lack of flexible and user-friendly tools for standardized modeling of them, particularly for real-world applications. Hence, we developed MLPro-GT as part of our open-source MLPro project, which offers modular and standardized yet flexible components, extensive documentation, and a variety of examples. MLPro-GT allows researchers and practitioners to easily incorporate game theory into their applications while lowering the entry barrier for students. This makes individual work more reproducible, shareable, and reusable.

Code metadata

Current code version	MLPro v1.9.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-24-00501
Permanent link to Reproducible Capsule	None
Legal Code License	Apache-2.0 license
Code versioning system used	git
Software code languages, tools, and services used	Python, Github, PyPI, Read the Docs
Compilation requirements, operating environments & dependencies	OS: Linux, Windows, macOS; Packages: numpy, matplotlib, dill, torch, scipy, multiprocess, panda
If available Link to developer documentation/manual	http://mlpro.readthedocs.io/ → Section 7: MLPro-GT - Game Theory
Support email for questions	mlpro@listen.fh-swf.de

1. Motivation and significance

Game theory (GT) is a mathematical concept that examines strategic interactions between rational decision-makers, where each player's outcomes depend on the actions of others [1,2]. By formalizing and analyzing competitive and cooperative scenarios, GT provides valuable insights into optimal decision-making strategies and the behavior of complex systems. In recent years, its applications have expanded beyond economics and mathematics to a wide range of scientific disciplines, which offers a systematic approach to understanding and predicting strategic behavior. In the fields of economics and mathematics, GT has significantly improved the analysis of market dynamics [3], pricing strategies [4], and more. In computer science, it plays a key role in enhancing network security protocols [5]. Additionally, in biology, GT helps with evolutionary strategies and species survival mechanisms [6]. Furthermore, in industrial processes, GT has been applied to model cooperative behavior in self-optimizing modular production systems using machine learning techniques [7,8].

In native GT, traditional models typically assume static environments where strategies and payoffs remain fixed over time [1,2]. However, to account for real-world scenarios where conditions and strategies change dynamically, enhanced concepts like dynamic GT have been developed [9,10]. Dynamic GT builds on native GT by incorporating the element of time and states, which enables the analysis of how players' decisions and strategies evolve as the state of the game changes. This approach is especially useful in highly dynamic environments, where strategies are influenced not only by immediate payoffs but also by the expected future actions and states of other players. By modeling games as a series of decisions over time, dynamic GT offers a deeper understanding of strategic interactions, supporting

* Corresponding author.
E-mail address: yuwono.steve@fh-swf.de (Steve Yuwono).

https://doi.org/10.1016/j.softx.2024.101956
Received 18 September 2024; Received in revised form 16 October 2024; Accepted 23 October 2024
Available online 4 November 2024

the study of complex phenomena such as real-time negotiations, adaptive behavior, and evolving competitive situations. Dynamic GT has found applications in various engineering domains, including dynamic potential games [11] and dynamic Stackelberg games [12] for control problems.

Despite the increasing use of both native and dynamic GT across various disciplines, there is a significant shortage of comprehensive software tools designed to support these analyzes. Most available GT software is designed for specific applications or lacks standardized functionality that can be applied across different types of games. This gap is particularly evident in dynamic GT, where no sufficiently robust framework exists to manage the complexities of time-evolving strategies and interactions. Consequently, researchers and practitioners encounter significant difficulties in effectively implementing dynamic GT models, which limits broader adoption and hinders further progress in the field.

Currently available GT software includes tools such as Nashpy [13], Axelrod [14], Gambit [15], PyNFG [16], Irslib [17], SageMath [18], and GAMUT [19], each offering distinct advantages and limitations:

- **Nashpy** specializes in two-player normal-form games with a user-friendly interface for computing Nash equilibria, but it lacks support for more complex game structures.
- **Axelrod** is tailored for evolutionary GT simulations, which excels in agent-based modeling, though its use is limited outside of evolutionary contexts.
- **Gambit** provides a comprehensive toolkit for analyzing a broad spectrum of games, including extensive-form and non-cooperative games, but it is complex to use and lacks features for dynamic GT.
- **PyNFG** offers a Python-based solution for network form games, providing flexibility for integration, though it has fewer functionalities compared to more mature software.
- **Irslib** and **SageMath** are general-purpose mathematical computing libraries with some GT capabilities, but they are less specialized for GT analysis.
- **GAMUT** provides extensive tools for testing GT algorithms but comes with a steep learning curve and limited support for dynamic games.

Overall, while these tools offer useful functionalities, there remains a significant gap in the availability of comprehensive and standardized software that addresses both native and dynamic GT, as well as integration with other domains.

Therefore, a new framework is needed that provides a standardized structure for models, templates, and processes for both native and dynamic GT. Ideally, this framework should be open-source to ensure broad accessibility and direct usability by the community. Additionally, it should support integration with existing third-party GT packages through wrapper technologies, which allows these tools to be deployed seamlessly within the framework. To address these needs, we have developed MLPro-GT, integrated into MLPro [20,21], an established open-source machine learning framework in Python. By reusing MLPro's existing functionalities (e.g., logging, data handling, data plotting), MLPro-GT benefits from a solid foundation, and the framework's support for wrapper technologies facilitates the extension to third-party packages. Furthermore, MLPro-GT enables GT to be applied across various domains.

Our key contributions are as follows:

1. We present MLPro-GT, a new open-source sub-framework within MLPro designed specifically for GT applications.
2. MLPro-GT includes standardized models, templates, and processes for both native and dynamic GT.
3. The framework is built with a clean, systematic software architecture that accommodates all game structures and learning algorithms/solvers in GT, which offers several ready-to-use games and algorithms.
4. We have validated MLPro-GT as a practical tool for GT practitioners in industry, academia, and students.

This paper is organized into five chapters. Chapter 2 presents an overview of MLPro-GT, including its architecture and functionalities. Chapter 3 provides illustrative examples of both native and dynamic GT applications. Chapter 4 discusses the impact of MLPro-GT in the field. Finally, Chapter 5 concludes the paper and outlines directions for future improvements.

2. Software description

MLPro-GT is an open-source, Python-based GT framework that provides standardized models, templates, and processes for both native and dynamic GT. It reuses several base classes and functionalities of MLPro [20,21], which makes MLPro-GT an embedded sub-framework within MLPro, as shown in Fig. 1. The native game theory module is referred to as MLPro-GT-NA, while the dynamic game theory module is named MLPro-GT-DG. MLPro-GT is designed to be versatile, which allows integration with other domains in GT. For instance, MLPro-GT-DG reuses several functionalities from MLPro-RL due to their close alignment in design, a relationship further detailed in the subchapter, and MLPro-MPPS [22,23] can be repurposed as a board game within MLPro-GT.

As part of MLPro, MLPro-GT is developed using object-oriented design (OOD) and object-oriented programming (OOP) principles. To ensure code quality, we follow the clean code paradigm and employ automated unit testing via GitHub Continuous Integration and Continuous Deployment (CI/CD) pipelines. MLPro under the Apache-2.0 license is available on GitHub[1] and the Python Package Index (PyPI).[2]

2.1. Software architecture

As previously noted, MLPro-GT is comprised of two modules: (1) MLPro-GT-NA for native GT and (2) MLPro-GT-DG for dynamic GT. In developing MLPro-GT, we systematically decompose the components of GT and implement them as distinct classes. Additionally, we reuse existing functionalities from MLPro to enhance the capabilities of MLPro-GT.

Fig. 2 presents the simplified class diagram of MLPro-GT-NA, which comprises 11 classes, ranging from the *GTTraining* class to the *GTSolver* class. Users can configure a game via the *GTGame* class, which can be structured as either a competition (*GTCompetition*), a coalition (*GTCoalition*), or a combination of both. Each game includes a Payoff Matrix (*GTPayoffMatrix*), with the option to define the matrix using a custom function (*GTFunction*). Within a coalition or competition, there is always a set of players defined by the *GTPlayer* class, where each player is assigned its own solver (*GTSolver*), and the combination of these players forms a strategy (*GTStrategy*). Finally, the game can be executed through *GTTraining*, with the outcomes and associated data managed by the *GTTrainingResults* and *GTDataStoring* classes.

Fig. 3 illustrates the simplified class diagram of MLPro-GT-DG, which inherits several classes from MLPro-RL. The core MLPro-GT-DG framework consists of six primary classes, along with one specialized class for dynamic potential games and two for dynamic Stackelberg games. The fundamental classes include *GameBoard* for setting up dynamic environments, *Player* for decision-making agents, and *MultiPlayer* for managing a group of players. The interaction between the players and the game board is defined in the *Game* class, which can be trained and validated using the *GTTraining* and *GTTrainingResults* classes. In MLPro 1.9.0, we offer two templates for dynamic game derivatives, which are state-based potential games via the *PGameBoard* class and hierarchical Stackelberg games through the *GTPlayer_SG* and *GTMultiPlayer_SG* classes. To be noted, MLPro-GT is not restricted to these two dynamic games, but other game types can be extended or customized based on our base classes.

[1] MLPro@GitHub: https://github.com/fhswf/MLPro.

[2] MLPro@PyPI: https://pypi.org/project/mlpro.

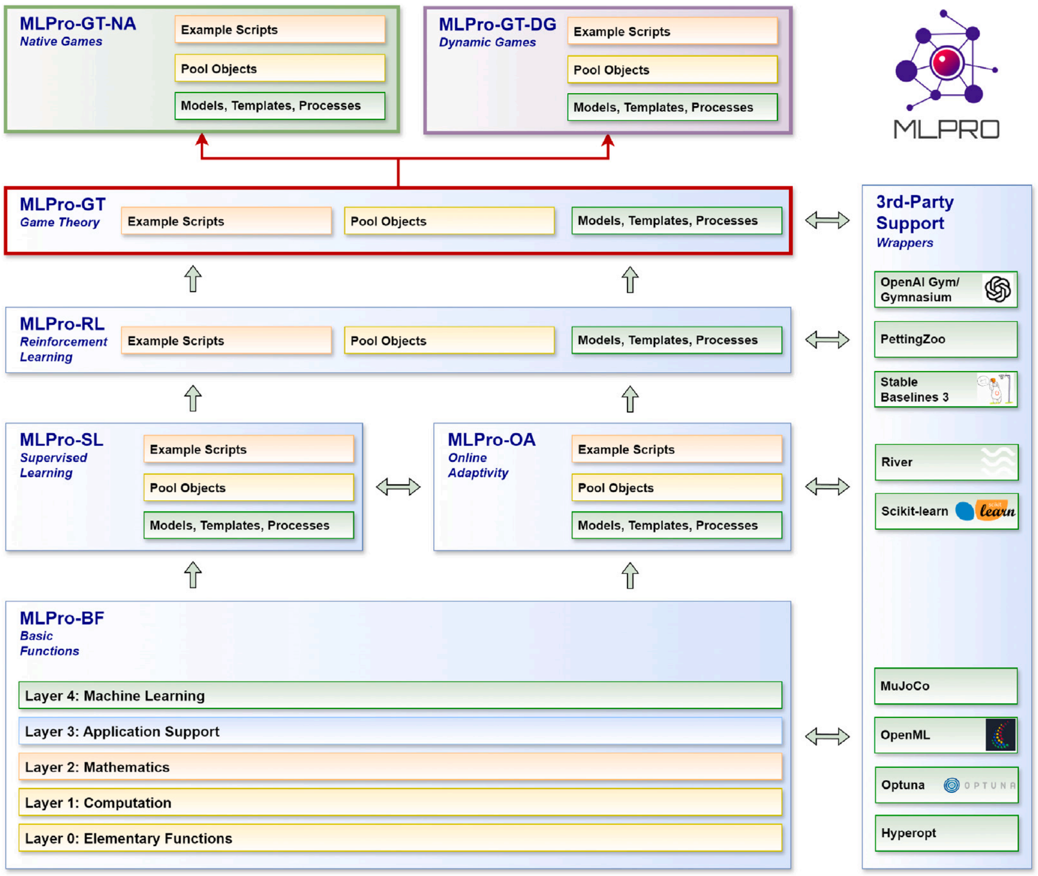

Fig. 1. The integration of MLPro-GT into MLPro.

2.2. Software functionalities

MLPro-GT is a comprehensive GT library that offers a wide array of functionalities designed to facilitate the development and execution of GT models. The library provides standardized foundational components for constructing various GT elements, including key concepts such as games, training, competition, coalitions, payoffs, solvers, and players in MLPro-GT-NA. In the context of MLPro-GT-DG, the library includes other terms, like gameboards and multiplayer setups for dynamic environments. By providing well-defined building blocks, MLPro-GT ensures a modular and structured approach to developing GT frameworks, which makes it easier for researchers and practitioners to design, implement and extend GT models.

One of the standout features of MLPro-GT is its ability to establish a standardized integration between components, which ensures a clean and consistent workflow. This approach not only streamlines the development process but also facilitates the execution of games and training processes in a uniform manner, which produces standardized outputs that allow for comparison across different approaches. Furthermore, MLPro-GT supports integration with other domains, such as reinforcement learning through MLPro-RL, where policies from MLPro-GT-DG can be applied in RL environments or other closed-loop control systems. For example, the MLPro-MPPS module can serve as a gameboard, while native GT can be used for decision-making in various domains.

Moreover, MLPro-GT offers flexibility in setting up diverse types of games and solvers. In MLPro-GT-DG, default configurations for dynamic potential games and Stackelberg games are provided, along with several preconfigured gameboards. Similarly, MLPro-GT-NA includes ready-to-use policies such as greedy and random strategies, which allows for rapid experimentation with different game types. All game and solver components are stored in an accessible pool of objects, which streamlines the process of selecting and customizing them. In addition, MLPro-GT maintains clean, well-documented code and supports third-party package integration via wrapper technologies, which enhances the library's adaptability and ease of use across multiple applications.

3. Illustrative examples

In this chapter, we present several examples and applications of both MLPro-GT-NA and MLPro-GT-DG. For MLPro-GT-NA, we have defined various games with distinct characteristics across different fields, which will be detailed in the subsequent sub-chapter. Similarly, MLPro-GT-DG has been applied in numerous research projects, and these applications will also be discussed in the following sub-chapter.

3.1. Native game theory

MLPro-GT-NA offers a range of ready-to-use games, each with different objectives and types. We also provide two ready-to-use algorithms in MLPro-GT-NA, such as random play for fully random decision-making and the best response for a greedy approach that maximizes immediate payoffs. Other algorithms can be easily customized to satisfy the specific needs of users. Below are some classic examples of games:

1. **The Prisoners' Dilemma** [24] is a classic GT scenario where players independently choose to cooperate or betray, with outcomes based on the combined choices. Each player selects either *Confess* or *Not Confess*, and their payoff depends on both their own and the other player's decisions, as shown in Table 1 for two players and Table 2 for three players. Analyzing the dilemma involves evaluating strategic interactions, incentives, and potential stable strategies or equilibrium.

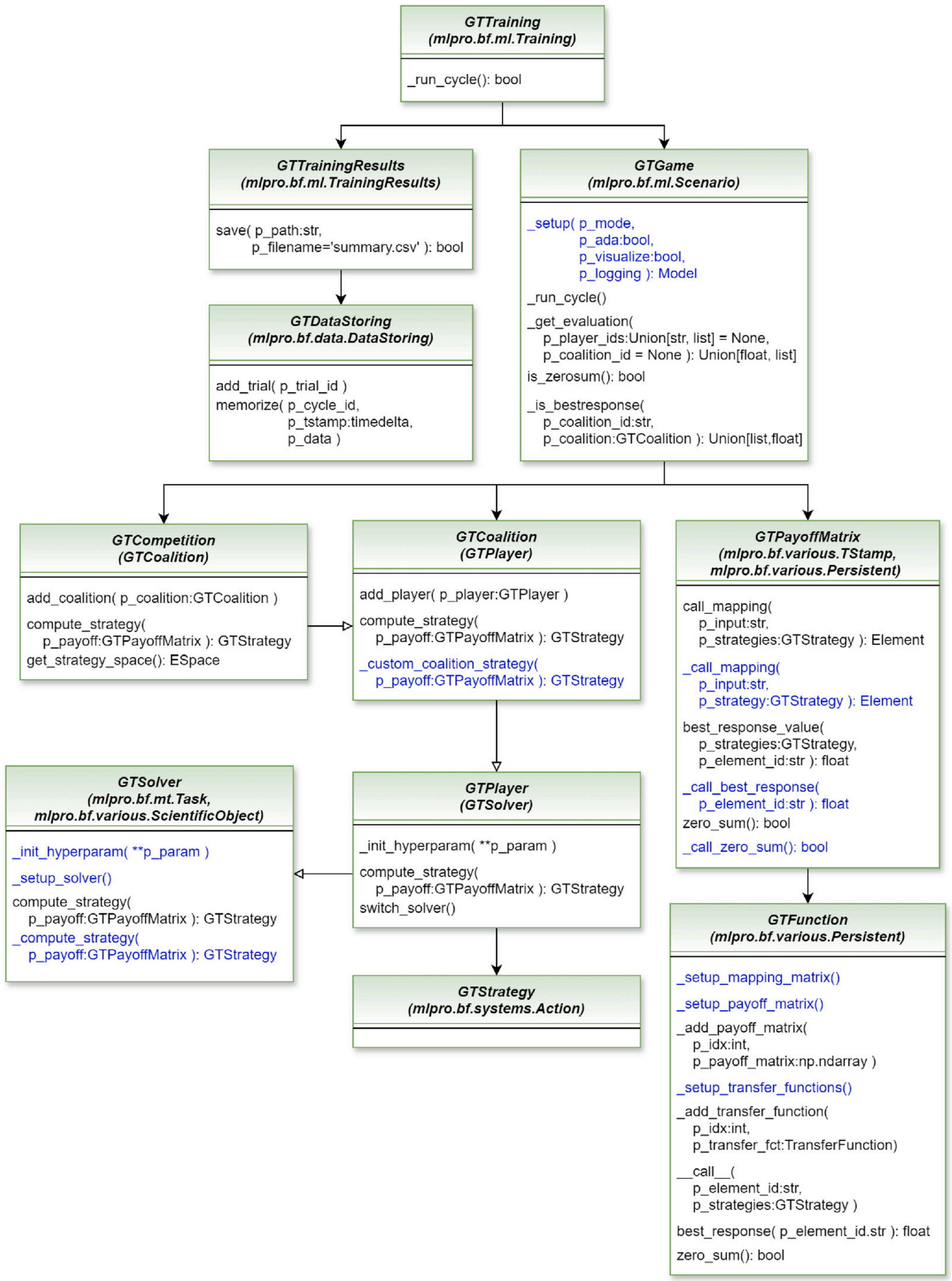
GTTraining (mlpro.bf.ml.Training)
_run_cycle(): bool
GTTrainingResults (mlpro.bf.ml.TrainingResults)
save(p_path:str, p_filename='summary.csv'): bool
GTDataStoring (mlpro.bf.data.DataStoring)
add_trial(p_trial_id)
memorize(p_cycle_id, p_tstamp:timedelta, p_data)
GTGame (mlpro.bf.ml.Scenario)
_setup(p_mode, p_ada:bool, p_visualize:bool, p_logging): Model
_run_cycle()
_get_evaluation(p_player_ids:Union[str, list] = None, p_coalition_id = None): Union[float, list]
is_zerosum(): bool
_is_bestresponse(p_coalition_id:str, p_coalition:GTCoalition): Union[list,float]
GTCompetition (GTCoalition)
add_coalition(p_coalition:GTCoalition)
compute_strategy(p_payoff:GTPayoffMatrix): GTStrategy
get_strategy_space(): ESpace
GTCoalition (GTPlayer)
add_player(p_player:GTPlayer)
compute_strategy(p_payoff:GTPayoffMatrix): GTStrategy
_custom_coalition_strategy(p_payoff:GTPayoffMatrix): GTStrategy
GTPayoffMatrix (mlpro.bf.various.TStamp, mlpro.bf.various.Persistent)
call_mapping(p_input:str, p_strategies:GTStrategy): Element
_call_mapping(p_input:str, p_strategy:GTStrategy): Element
best_response_value(p_strategies:GTStrategy, p_element_id:str): float
_call_best_response(p_element_id:str): float
zero_sum(): bool
_call_zero_sum(): bool
GTSolver (mlpro.bf.mt.Task, mlpro.bf.various.ScientificObject)
_init_hyperparam(**p_param)
_setup_solver()
compute_strategy(p_payoff:GTPayoffMatrix): GTStrategy
_compute_strategy(p_payoff:GTPayoffMatrix): GTStrategy
GTPlayer (GTSolver)
_init_hyperparam(**p_param)
compute_strategy(p_payoff:GTPayoffMatrix): GTStrategy
switch_solver()
GTStrategy (mlpro.bf.systems.Action)
GTFunction (mlpro.bf.various.Persistent)
_setup_mapping_matrix()
_setup_payoff_matrix()
_add_payoff_matrix(p_idx:int, p_payoff_matrix:np.ndarray)
_setup_transfer_functions()
_add_transfer_function(p_idx:int, p_transfer_fct:TransferFunction)
__call__(p_element_id:str, p_strategies:GTStrategy)
best_response(p_element_id.str): float
zero_sum(): bool

Fig. 2. The simplified class diagram of MLPro-GT-NA.

2. **Rock, Paper, Scissors** [25] is a classic non-cooperative, simultaneous-move, zero-sum game in GT. Its cyclic structure ensures no option dominates, and the payoffs sum to zero, as shown in Table 3. Commonly used to teach introductory GT, it illustrates key concepts like mixed strategies, Nash equilibrium, and zero-sum dynamics.
3. **3P Supply and Demand:** [26] Three sellers make strategic decisions on the quantity of goods to supply, while buyers independently determine their demand. Sellers aim to maximize revenue based on quantity and pricing, while buyers seek to maximize utility through purchased goods and prices, as outlined in Table 4. Market dynamics are shaped by the interaction between supply and demand, with equilibrium achieved when supply matches demand, setting a market-clearing price. Sellers compete by adjusting prices and quantities or may cooperate strategically to influence the market. The Nash equilibrium is reached when each player's strategy is optimal given the others' choices.
4. **3P Routing Problems with Congestion:** [27] Three players navigate a network with multiple routes, aiming to minimize their travel time or cost, which is influenced by the collective route choices of all players. As more players choose the same route, congestion increases, which leads to longer travel times or higher costs. Each player makes independent decisions to optimize their utility by weighing the trade-offs between faster, more congested routes and longer, less congested ones. The

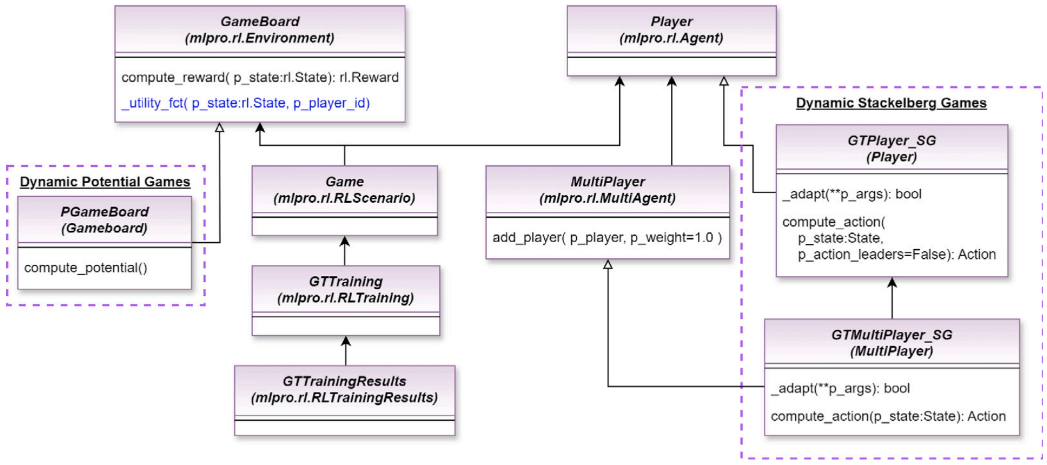
GameBoard (mlpro.rl.Environment)
compute_reward(p_state:rl.State): rl.Reward
_utility_fct(p_state:rl.State, p_player_id)
Player (mlpro.rl.Agent)
Dynamic Stackelberg Games
GTPlayer_SG (Player)
_adapt(**p_args): bool
compute_action(p_state:State, p_action_leaders=False): Action
Dynamic Potential Games
PGameBoard (Gameboard)
compute_potential()
Game (mlpro.rl.RLScenario)
MultiPlayer (mlpro.rl.MultiAgent)
add_player(p_player, p_weight=1.0)
GTTraining (mlpro.rl.RLTraining)
GTTrainingResults (mlpro.rl.RLTrainingResults)
GTMultiPlayer_SG (MultiPlayer)
_adapt(**p_args): bool
compute_action(p_state:State): Action

Fig. 3. The simplified class diagram of MLPro-GT-DG.

Table 1
Payoff matrix of 2P Prisoners' Dilemma.

	Player 2 Confess	Player 2 Not Confess
Player 1 Confess	(2, 2)	(8, 1)
Player 1 Not Confess	(1, 8)	(5, 5)

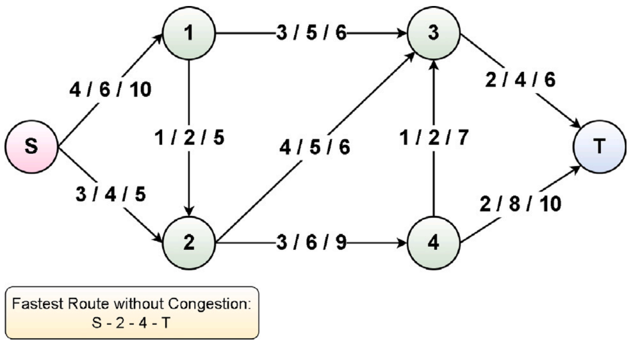

Fig. 4. The travel times in 3P Routing Problems with Congestion.

Table 2
Payoff matrix of 3P Prisoners' Dilemma, *C* for *Confess* and *NC* for *Not Confess*.

(P1, P2, P3)	P2: C, P3: C	P2: C, P3: NC	P2: NC, P3: C	P2: NC, P3: NC
P1: C	(2, 2, 2)	(5, 5, 1)	(5, 1, 5)	(10, 1, 1)
P1: NC	(1, 5, 5)	(1, 10, 1)	(1, 1, 10)	(15, 15, 15)

Table 3
Payoff matrix of Rock, Paper, Scissors.

	Rock	Paper	Scissors
Rock	(0, 0)	(0, 1)	(1, 0)
Paper	(1, 0)	(0, 0)	(0, 1)
Scissors	(0, 1)	(1, 0)	(0, 0)

Table 4
Payoff function of 3P Supply and Demand.

Seller 1		Seller 2		Seller 3	
Price	Quantity	Price	Quantity	Price	Quantity
15	1	10	1	8	1
12	2	8	2	7	2
9	3	6	3	6	3
6	4	4	4	5	4
3	5	2	5	4	5

game reaches a Nash equilibrium when no player can improve their travel time by changing routes, given the choices of others. Fig. 4 illustrates the travel times for different routes based on the number of players using each path, with times (X, Y, Z) representing scenarios for one, two, or three players choosing the same route.

These examples demonstrate that MLPro-GT-NA is capable of constructing and running games with varying numbers of players or coalitions, and can support collaborative, cooperative, or hybrid games. They also highlight that solvers can be customized using standardized methods provided by the package. Consequently, compared to other GT libraries, MLPro-GT-NA stands out for its flexibility and ease of use, which offers the ability to handle various and mixed game configurations.

3.2. Dynamic games

MLPro-GT-DG has been widely applied in our research, which is focused on developing self-optimizing modular production systems using dynamic game theory principles. One of the key test cases is a Bulk Good System [7], as shown in Fig. 5, and its larger-scale counterpart [28].

Additionally, MLPro-GT-DG has been featured in several publications that serve as successful sample applications, including:

1. **Model-based game theory** [28] is a hybrid method between deep supervised learning and game theory that involves using predictive models to simulate and optimize game dynamics within virtual environments, thereby reducing the need for costly and time-consuming interactions with actual systems.
2. **Gradient-based learning in state-based potential games** [29] uses gradients to guide optimization for faster convergence and more efficient exploration compared to traditional ad-hoc random methods in best response learning, thus shortening training times and improving policy effectiveness.
3. **Transfer learning in state-based potential games** [30] is a method that involves reusing knowledge gained by one player to improve the learning and policy optimization of other similar players, thereby enhancing self-optimization and accelerating the learning process in distributed systems.

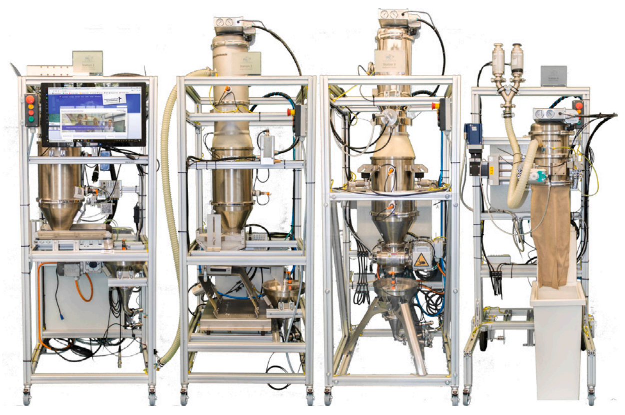

Fig. 5. Bulk good system [8].

4. **Distributed Stackelberg games** [12] is a game structure that combines elements of potential games and Stackelberg games to autonomously optimize decentralized systems by handling multiple objectives and maintaining distributed training processes.

These examples demonstrate that MLPro-GT-DG is compatible with diverse and combined game structures, algorithms such as best response learning [7] and gradient-based learning [29], as well as various configurations while highlighting its potential for integration with industrial control systems. Additionally, compared to other GT libraries, which are unable to handle dynamic games, MLPro-GT-DG can effectively manage the complexities of time-evolving strategies and interactions.

4. Impact

MLPro-GT has the potential for significant societal impact due to its accessibility, flexibility, and the innovative features it offers. As an open-source library, it is freely available to a broad range of users, including students, researchers, and industry professionals. The library is regularly maintained and supported by comprehensive documentation, which ensures ease of use for beginners and experts alike. Its open-source nature also permits commercial use. MLPro-GT is the first framework to provide support for dynamic games, a major advancement in the field.

A key strength of MLPro-GT is its standardized templates for GT components and their integration. This includes a uniform workflow for running and training games, which enables direct comparison of different models. These features are lacking in other GT libraries. Additionally, MLPro-GT supports cross-domain applications. Its broad applicability ensures that not only students but also practitioners in both academia and industry can benefit from its capabilities. For example, as demonstrated in Chapter 3.2, MLPro-GT-DG has been employed in numerous research projects.

5. Conclusions

In conclusion, MLPro-GT offers a comprehensive set of models, templates, and processes for implementing GT in Python. It provides two different GT approaches: (1) MLPro-GT-NA, designed for native GT, and (2) MLPro-GT-DG, which specializes in dynamic games. As a part of the broader MLPro framework, MLPro-GT is open-source and accessible to both students and professionals in research and industry. Notably, MLPro-GT-DG is the first framework dedicated to handling dynamic games, a significant innovation in the field. Beyond its foundational models, MLPro-GT supports third-party integration with other GT packages and features an increasing pool for storing GT algorithms, games, and more.

For future development, we aim to expand MLPro-GT's capabilities by integrating additional third-party packages, like NashPy. We also plan to introduce a wider variety of games and solvers for both native and dynamic GT. Additionally, we seek to apply MLPro-GT in various domains.

CRediT authorship contribution statement

Steve Yuwono: Writing – review & editing, Writing – original draft, Validation, Software, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Detlef Arend:** Writing – review & editing, Project administration, Methodology, Investigation, Formal analysis, Conceptualization. **Andreas Schwung:** Writing – review & editing, Supervision, Methodology, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We would like to extend our gratitude to the other MLPro contributors[3] for helping us in developing the framework.

Data availability

No data was used for the research described in the article.

References

[1] Fudenberg D, Tirole J. Game theory. MIT Press; 1991.
[2] Barron EN. Game theory: an introduction. John Wiley & Sons; 2024.
[3] Zhu C, Fan R, Lin J. The impact of renewable portfolio standard on retail electricity market: A system dynamics model of tripartite evolutionary game. Energy Policy 2020;136:111072.
[4] Mamoudan MM, Mohammadnazari Z, Ostadi A, Esfahbodi A. Food products pricing theory with application of machine learning and game theory approach. Int J Prod Res 2024;62(15):5489–509.
[5] Abdalzaher MS, Muta O. A game-theoretic approach for enhancing security and data trustworthiness in IoT applications. IEEE Internet Things J 2020;7(11):11250–61.
[6] Leimar O, McNamara JM. Game theory in biology: 50 years and onwards. Philos Trans R Soc B 2023;378(1876):20210509.
[7] Schwung D, Schwung A, Ding SX. Distributed self-optimization of modular production units: A state-based potential game approach. IEEE Trans Cybern 2020;52(4):2174–85.
[8] Schwung D, Yuwono S, Schwung A, Ding SX. PLC-informed distributed game theoretic learning of energy-optimal production policies. IEEE Trans Cybern 2022;53(9):5424–35.
[9] Başar T, Olsder GJ. Dynamic noncooperative game theory. SIAM; 1998.
[10] Başar T, Zaccour G. Handbook of dynamic game theory. Springer; 2018.
[11] Zazo S, Macua SV, Sánchez-Fernández M, Zazo J. Dynamic potential games with constraints: Fundamentals and applications in communications. IEEE Trans Signal Process 2016;64(14):3806–21.
[12] Yuwono S, Schwung D, Schwung A. Distributed Stackelberg strategies in state-based potential games for autonomous decentralized learning manufacturing systems. 2024, arXiv preprint arXiv:2408.06397.
[13] Knight V, Campbell J. Nashpy: A Python library for the computation of Nash equilibria. J Open Sour Softw 2018;3(30):904.
[14] Knight V, Campbell O, Harper M, Langner K, Campbell J, Campbell T, et al. An open framework for the reproducible study of the iterated prisoner's dilemma. J Open Res Softw 2016;4(1):e35.
[15] McKelvey RD, McLennan AM, Turocy TL. Gambit: Software tools for game theory. 2006, Version 0.2006. 01.20.
[16] Wolpert D, Lee R, Bono J. PyNFG: A Python package for modeling and solving network form games. 2024, GitHub, URL https://github.com/jwbono/PyNFG.

[3] https://github.com/fhswf/MLPro/graphs/contributors

[17] Avis D. Lrslib. 2024, lrs home page, URL http://cgm.cs.mcgill.ca/~avis/C/lrs.html.
[18] Assumpção GdS, Santos CMd, Campello Dd, de Lima LS, Castro Ad. A proposal of teaching operational research in online contexts: An experience with SageMath in Brazil. Eng Rep 2024;e12863.
[19] Nudelman E, Wortman J, Shoham Y, Leyton-Brown K. Run the GAMUT: A comprehensive approach to evaluating game-theoretic algorithms. In: AAMAS, vol. 4, 2004, p. 880–7.
[20] Arend D, Yuwono S, Diprasetya MR, Schwung A. MLPro 1.0-standardized reinforcement learning and game theory in Python. Mach Learn Appl 2022;9:100341.
[21] Arend D, Diprasetya MR, Yuwono S, Schwung A. MLPro—An integrative middleware framework for standardized machine learning tasks in Python. Softw Impacts 2022;14:100421.
[22] Yuwono S, Löppenberg M, Arend D, Diprasetya MR, Schwung A. MLPro-MPPS—A high-performance simulation framework for customizable production systems. Softw Impacts 2023;16:100509.
[23] Yuwono S, Löppenberg M, Arend D, Diprasetya MR, Schwung A. MLPro-MPPS-A versatile and configurable production systems simulator in Python. In: 2023 IEEE 2nd industrial electronics society annual on-line conference. IEEE; 2023, p. 1–6.
[24] Shubik M. Game theory, behavior, and the paradox of the prisoner's dilemma: Three solutions. J Conflict Resolut 1970;14(2):181–93.
[25] Zhou H-J. The rock–paper–scissors game. Contemp Phys 2016;57(2):151–63.
[26] Esmaeili M, Aryanezhad M-B, Zeephongsekul P. A game theory approach in seller–buyer supply chain. European J Oper Res 2009;195(2):442–8.
[27] Meyers CA, Schulz AS. The complexity of welfare maximization in congestion games. Networks 2012;59(2):252–60.
[28] Yuwono S, Schwung A. Model-based learning on state-based potential games for distributed self-optimization of manufacturing systems. J Manuf Syst 2023;71:474–93.
[29] Yuwono S, Löppenberg M, Schwung D, Schwung A. Gradient-based learning in state-based potential games for self-learning production systems. 2024, arXiv preprint arXiv:2406.10015.
[30] Yuwono S, Schwung D, Schwung A. Transfer learning of state-based potential games for process optimization in decentralized manufacturing systems. 2024, arXiv preprint arXiv:2408.05992.