|  |  |
| --- | --- |
| Acad Year (24/25)  PROJECT NO. (D014) | **Enhancing supply chain resilience through game theory modelling and reinforcement learning** |
| **Enhancing supply chain resilience through game theory modelling and reinforcement learning** | **Lau Yong**  **U2020638F**  **SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING**  **NANYANG TECHNOLOGICAL UNIVERSITY**  **Year (2024/2025)** |

ENHANCING SUPPLY CHAIN RESILIENCE THROUGH GAME THEORY MODELLING AND REINFORCEMENT LEARNING

SUBMITTED

BY

Lau Yong

U2020638F

SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

A final year project report

presented to

Nanyang Technological University

in partial fulfilment of the

requirements for the

Degree of Bachelor of Engineering (Aerospace Engineering)

Nanyang Technological University

Year (2024/2025)Table of Contents

[1 Introduction 6](#_Toc195654344)

[2 Literature Review 8](#_Toc195654345)

[2.1 Theoretical Background and Foundational Research 8](#_Toc195654346)

[2.1.1 The strategic analysis of logistics service sharing in an e-commerce platform, (Qin et al., 2020) 8](#_Toc195654347)

[2.1.2 Implementing E-Commerce from Logistic Perspective: Literature Review and Methodological Framework, (Zennaro et al., 2022) 9](#_Toc195654348)

[2.1.3 Logistics service sharing in cross-border e-commerce, (Khooban et al., 2025) 9](#_Toc195654349)

[2.1.4 E-Commerce Supply Chain Efficiency: A Case Study of Amazon E-Commerce Company, (Vidani, 2024) 10](#_Toc195654350)

[2.1.5 Defining Supply Chain Management, (Mentzer et al., 2001) 10](#_Toc195654351)

[2.1.6 Designing and Managing the Supply Chain, (Simchi-levi et al., 2003) 10](#_Toc195654352)

[2.1.7 Chopra, Sunil and Peter Meindl. (2010). 'Supply Chain Management: Strategy, Planning, and Operation', (Chopra & Meindl, 2002) 10](#_Toc195654353)

[2.1.8 Managing Supply Chains: A Logistics Approach, (Coyle et al., 2017) 10](#_Toc195654354)

[2.1.9 E-supply chains—Virtually non-existent, (Hoek, 2001) 10](#_Toc195654355)

[2.1.10 Section Summary 11](#_Toc195654356)

[2.2 Strategic Interactions Between Platforms, Sellers, and TPLPs 11](#_Toc195654357)

[2.2.1 Logistics capability, logistics outsourcing and firm performance in an e-commerce market, (Joong‐Kun Cho et al., 2008) 11](#_Toc195654358)

[2.2.2 Compete or cooperate: Intensity, dynamics, and optimal strategies, (Chen et al., 2019) 11](#_Toc195654359)

[2.2.3 Capacity Sharing Between Competitors, (Guo & Wu, 2018) 11](#_Toc195654360)

[2.2.4 The strategic role of logistics in the industry 4.0 era, (Tang & Veelenturf, 2019). 12](#_Toc195654361)

[2.2.5 Logistics choices in a platform supply chain: A co-opetitive perspective, (Wang et al., 2022) 12](#_Toc195654362)

[2.2.6 Section summary 12](#_Toc195654363)

[2.3 Game Theory in Supply Chain Management 12](#_Toc195654364)

[2.3.1 A Stackelberg-game model in a two-stage supply chain, Qin (2012) 12](#_Toc195654365)

[2.3.2 Service competition in an online duopoly market, (Ding et al., 2018) 12](#_Toc195654366)

[2.3.3 Channel Dynamics Under Price and Service Competition, (Tsay & Agrawal, 2000) 13](#_Toc195654367)

[2.3.4 A General Equilibrium Model for Industries with Price and Service Competition, (Bernstein & Federgruen, 2004) 13](#_Toc195654368)

[2.3.5 Game-theoretic analysis of a two-stage dual-channel supply chain coordination in the presence of market segmentation and price discounts, (Zhao & Li, 2023) 13](#_Toc195654369)

[2.3.6 Game theoretical analysis of incumbent platform investment and the supplier entry strategies in an e-supply chain, (Zhuo et al., 2024) 13](#_Toc195654370)

[2.3.7 Section summary 13](#_Toc195654371)

[2.4 Reinforcement Learning for Supply Chains 14](#_Toc195654372)

[2.4.1 Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities, (Yan et al., 2022) 14](#_Toc195654373)

[2.4.2 Reinforcement learning for solving the vehicle routing problem, (Nazari et al., 2018) 14](#_Toc195654374)

[2.4.3 Dynamic Pricing Model of E-Commerce Platforms Based on Deep Reinforcement Learning, (Yin & Han, 2021) 14](#_Toc195654375)

[2.4.4 A multi-agent reinforcement learning model for inventory transshipments under supply chain disruption, (Kim et al., 2024) 15](#_Toc195654376)

[2.4.5 Section Summary 15](#_Toc195654377)

[3 Simulation environment design and experimental framework 15](#_Toc195654378)

[3.1 Overview of the context and key variables 16](#_Toc195654379)

[3.2 Stepwise Approach: Base Model, Game Theory, and Reinforcement Learning Integration 18](#_Toc195654380)

[3.2.1 Reproduction of original model and adaptations to the model 19](#_Toc195654381)

[3.2.2 Game Theory Framework and Implementation 39](#_Toc195654382)

[3.2.3 Reinforcement Learning implementation 43](#_Toc195654383)

[4 Results and Discussion 50](#_Toc195654384)

[4.1 Sharing Zone Analysis from Stackelberg Game 50](#_Toc195654385)

[4.2 TPLP’s Adaptive Strategy under no capacity constraints 52](#_Toc195654386)

[4.3 TPLP’s Adaptive Strategy under capacity constraints 55](#_Toc195654387)

[4.4 Discussion 57](#_Toc195654388)

[5 Conclusion 59](#_Toc195654389)

[5.1 Real-World Implications and Strategic Recommendations 60](#_Toc195654390)

[5.2 Limitations and future works 60](#_Toc195654391)

[6 References 64](#_Toc195654392)

[7 Appendix 67](#_Toc195654393)

[7.1.1 Code for Figure 19,17,18 67](#_Toc195654394)

[7.1.2 Code for Figure 22 69](#_Toc195654395)

[7.1.3 Code for policy extraction (Figure 26) 70](#_Toc195654396)

[7.1.4 Code for PPO RL 78](#_Toc195654397)

Table of Figures

[Figure 1: Model Structure 18](#_Toc195654398)

[Figure 2: Key parameters of the model 20](#_Toc195654399)

[Figure 3: E-tailer and seller pricing strategies under sharing 22](#_Toc195654400)

[Figure 4: E-tailer and seller pricing strategies under no-sharing 23](#_Toc195654401)

[Figure 5: Calculation of logistics service cost 25](#_Toc195654402)

[Figure 6: General demand equation 26](#_Toc195654403)

[Figure 7: E-tailer and seller demand under sharing 26](#_Toc195654404)

[Figure 8: E-tailer and seller demand under no-sharing 26](#_Toc195654405)

[Figure 9: Excess demand calculation 28](#_Toc195654406)

[Figure 10: Excess demand flowchart 29](#_Toc195654407)

[Figure 11: Profits for E-tailer and seller under no-sharing 30](#_Toc195654408)

[Figure 12: Profit function for E-tailer and seller under no-sharing 30](#_Toc195654409)

[Figure 13: Simplified profit equation for E-tailer and seller under no-sharing 31](#_Toc195654410)

[Figure 14: Profits for E-tailer and seller under sharing 31](#_Toc195654411)

[Figure 15: Equation for seller under sharing 31](#_Toc195654412)

[Figure 16: First profit equation for E-tailer under sharing 32](#_Toc195654413)

[Figure 17: Second profit equation for E-tailer under sharing 32](#_Toc195654414)

[Figure 18: Profits for TPLP under sharing and no-sharing 33](#_Toc195654415)

[Figure 19: TPLP cost function 33](#_Toc195654416)

[Figure 20: TPLP profit function under no-sharing 34](#_Toc195654417)

[Figure 21: Profit zones for E-tailer and seller ( 36](#_Toc195654418)

[Figure 22: Profit zones for E-tailer and seller ( when E-tailer makes concessions 37](#_Toc195654419)

[Figure 23: Profit zones for E-tailer and seller ( 39](#_Toc195654420)

[Figure 24: Stackelberg framework 40](#_Toc195654421)

[Figure 25: Stackelberg game function 42](#_Toc195654422)

[Figure 26: AEC 44](#_Toc195654423)

[Figure 27: PPO Architecture 47](#_Toc195654424)

[Figure 28: 3D Plot of sharing and no-sharing regions 52](#_Toc195654425)

[Figure 29: TPLP Simulation Game 53](#_Toc195654426)

Table of Tables

[Table 1: Important parameters and variables 16](#_Toc195654427)

[Table 2: TPLP's actions under varying market potential with capacity constraint 53](#_Toc195654428)

[Table 3: TPLP's actions under varying market potential with capacity constraint ( 55](#_Toc195654429)

**Abstract**

The increasing complexity and vulnerability of modern e-commerce supply chains, exacerbated by global disruptions such as the COVID-19 pandemic, have emphasized the need for adaptive and resilient logistics strategies. This study explores the strategic interactions between e-commerce platforms (E-tailers), third-party logistics providers (TPLPs), and sellers through a novel hybrid framework that integrates game theory and reinforcement learning. Building upon the foundational model by Qin et al. (2020), which analysed logistics service sharing between E-tailers and sellers, this project extends the framework by incorporating TPLPs as dynamic, strategic agents while in combination with the concept of capacity constraints for E-tailers.

A Stackelberg game model is employed to characterise hierarchical decision-making among the stakeholders, with the TPLP acting as the leader and the E-tailer and seller as followers. This static model is further enriched using Proximal Policy Optimization (PPO), a reinforcement learning algorithm, to simulate adaptive strategies of the TPLP in response to changing market conditions and competitive dynamics. The simulation environment is developed in Python using the PettingZoo and Ray RLlib libraries, enabling single-agent training with varying market conditions.

Prior results revealed that logistics service sharing is beneficial under specific conditions, particularly when the market potential and TPLP service level are moderate. From this simulation, the reinforcement learning model allows the TPLP to iteratively improve its service level and pricing decisions, maximizing its own profitability while influencing the equilibrium outcomes of the other agents. Additionally, incorporating E-tailer capacity constraints shifts profit regions and hence strategic behaviour, often incentivizing collaboration through adjusted pricing and service level strategies.

# Introduction

The logistics industry serves as the bedrock of global commerce, enabling the efficient transport of goods across supply chains. With disruptions such as the COVID-19 pandemic and the resultant rapid growth of e-commerce platforms, the logistics sector has faced unprecedented challenges in maintaining efficiency, resilience, and flexibility. These pressures have propelled the development of innovative supply chain management (SCM) solutions. (Khooban et al., 2025).

E-commerce platforms rely heavily on efficient supply chains to meet consumer demand for rapid and reliable deliveries (Vidani, 2024). Hence, to optimise the efficiency of these e-commerce supply chains, it may be impactful to investigate the dynamics between the prevalent stakeholders involved, namely: e-commerce platforms (E-tailers), sellers and third-party logistics providers (TPLPs). A study by Qin et al., (2020) highlights the potential of logistics service sharing between E-tailers and sellers to enhance profits for both parties. However, Qin et al.'s, (2020) framework primarily focuses on the interaction between E-tailers and sellers, ignoring how TPLPs could respond in their own interests. Furthermore, the paper only analyses static scenarios, leaving rooms for opportunity in understanding the dynamic, real-time interactions which reflect modern supply chains.

The analysis of TPLPs in supply chain interactions is crucial as they add notable complications. TPLPs act as independent entities providing specialized services such as warehousing, transportation, and distribution. Unlike platform-owned logistics arms, TPLPs often operate as both competitors and collaborators within the same supply chain ecosystem, complicating dynamics (T. Chen et al., 2024).

To model the interactions between these three players, game theory can be applied. Game theory is a mathematical framework for analysing interactions among rational decision-makers, and it offers a powerful tool to study these dynamics in supply chain interactions (Rzeczycki, 2022). However, amongst the various scientific studies on decision making in supply chains in a survey paper by Rzeczycki, (2022) the E-commerce industry was not present. This indicates potential for research in this area. In particular, the Stackelberg game model, introduced by (Von Stackelberg, 2011) will be used in this study as it is well-suited for hierarchical supply chain structures (Y. Qin, 2012).

While game theory provides valuable insights into static equilibrium strategies, real-world logistics systems are dynamic and require adaptive approaches to account for variances in demand, costs, and competition. In logistics and SCM, reinforcement learning (RL) has the potential to bridge this gap (Yan et al., 2022). Specifically in the field of SCM, Yan et al., (2022) provided evidence of RL use in coordination between cooperating supply chain entities. However, there is a lack of research in the field of e-commerce, where both collaboration and competition can exist (Qin et al., 2020). The Proximity Policy Optimisation (PPO) based RL would be utilised as it has empirical success in many challenging domains and is currently widely used for multi-agent purposes (Feng et al., 2023).

To add on another layer of dynamics, this study also considered the possibility of capacity sharing between the E-tailer and seller. Capacity sharing is a common practice between logistics providers to align excessive capacity with excessive demand (Guo & Wu, 2018). This study investigated the impacts of capacity constrained faced by the E-tailer which will be handled by the TPLP.

Overall, this study aims to identify decision making strategies in e-commerce supply chain dynamics using game theory and reinforcement learning techniques. This study builds upon the analysis of Qin et al., (2020) by integrating the dynamic behaviours of TPLPs into the existing framework. This would be done by employing a hybrid approach of Stackelberg game theory and PPO-based reinforcement learning, offering insights into optimizing supply chain networks in an increasingly interconnected and competitive environment.

# Literature Review

## Theoretical Background and Foundational Research

### The strategic analysis of logistics service sharing in an e-commerce platform, (Qin et al., 2020)

(Qin et al., 2020) provides the foundational study of logistics service sharing within hybrid e-commerce platforms by developing a game-theoretic model that investigates the strategic interaction between an E-tailer and a seller. The model considers two logistics configurations: No-Service sharing, where the seller outsources to a third-party logistics provider (TPLP), and Service sharing, where the E-tailer shares its logistics infrastructure. Through careful comparative statics and equilibrium analysis, the paper demonstrates that the profitability of each player varies significantly depending on the TPLP’s logistics service level and the market potential.

One of the key insights from the paper is the identification of three strategic regions in the θ–service level space: a *win-win* region where both platform and seller benefit from logistics sharing, a *win-lose* scenario where the platform benefits but the seller is worse off, and a *lose-win* scenario where the reverse holds. These distinctions are essential in identifying the incentive alignment needed to make logistics sharing viable. The authors further derive equilibrium prices and profits under both modes, contributing robust analytical benchmarks for more complex future models.

Overall, Qin et al. (2020) lay the essential theoretical foundation for modelling logistics service sharing in platform ecosystems, while this enriches this framework with dynamic, agent-based, and learning-driven elements that are increasingly relevant in modern supply chains​

However, while the model is rigorous, it is primarily static and assumes exogenously fixed logistics costs and capacities. It also treats the TPLP as a passive external entity rather than a strategic player. This limitation is the motivation for this study, which extends Qin et al.'s work by incorporating the TPLP as an active, learning agent using Proximal Policy Optimization (PPO). By introducing reinforcement learning, this research moved beyond static equilibria and into adaptive strategies that better reflect real-world platform-seller-TPLP dynamics under uncertainty and limited information.

### Implementing E-Commerce from Logistic Perspective: Literature Review and Methodological Framework, (Zennaro et al., 2022)

Zennaro et al., 2022 emphasized structural and operational logistics layers in e-commerce but do not model dynamic agent behaviors. This work builds on this by incorporating learning agents into a simulation framework.

### Logistics service sharing in cross-border e-commerce, (Khooban et al., 2025)

Khooban et al., (2025) explored logistics service sharing (LSS) in cross-border e-commerce, highlighting the need for adaptive coordination mechanisms in complex international settings. While they introduced the concept of “logistics convergence zones” to manage cooperation amid uncertainty, their work lacked computational simulation to test dynamic strategy formation. This study fills this gap by using reinforcement learning to model how TPLPs adjust service levels and pricing in volatile cross-border logistics environments.

### E-Commerce Supply Chain Efficiency: A Case Study of Amazon E-Commerce Company, (Vidani, 2024)

Vidani, (2024) real-world analysis of Amazon logistics illustrates the value of internal logistics control but does not offer a modeling framework. This study translates such real-world behaviors into formal strategic simulations.

### Defining Supply Chain Management, (Mentzer et al., 2001)

Mentzer et al., (2001) defined integrated logistics but missed how profit-sharing dynamics evolve in competitive markets. This study models those profit interactions explicitly.

### Designing and Managing the Supply Chain, (Simchi-levi et al., 2003)

Simchi-levi et al., (2003) provided broad supply chain frameworks but did not include third-party logistics influence on platform competition. This is incorporated in this simulation.

### Chopra, Sunil and Peter Meindl. (2010). 'Supply Chain Management: Strategy, Planning, and Operation', (Chopra & Meindl, 2002)

Chopra & Meindl, (2002) covered network design but not adaptive strategies over time. This research simulates long-term learning and adjustment.

### Managing Supply Chains: A Logistics Approach, (Coyle et al., 2017)

Coyle et al., (2017) provided practical insights into logistics outsourcing but missed strategic incentives for coopetition. This study incorporates game theory to explain such incentives.

### E-supply chains—Virtually non-existent, (Hoek, 2001)

Hoek, (2001) explored logistics innovation, but not from a multi-agent interaction lens. This study focuses on innovation through behavioral adaptation of the TPLP.

### Section Summary

This section laid the theoretical groundwork for understanding coopetition and dual-channel dynamics in logistics service sharing. However, many of these foundational models assumed static settings and predefined roles for decision-makers. This created a gap in addressing dynamic strategy adaptation, particularly from the TPLP’s perspective. By incorporating reinforcement learning within a game-theoretic setup, this study addresses the limitations of static models by simulating ongoing strategic decision-making under uncertainty and partial observability. This enhances the realism of modeling how logistics providers react in evolving coopetitive networks.

## Strategic Interactions Between Platforms, Sellers, and TPLPs

### Logistics capability, logistics outsourcing and firm performance in an e-commerce market, (Joong‐Kun Cho et al., 2008)

Joong‐Kun Cho et al., (2008) highlights the strategic impact of outsourcing logistics. However, it does not model the behavior of logistics providers, which this research does by simulating TPLP learning.

### Compete or cooperate: Intensity, dynamics, and optimal strategies, (Chen et al., 2019)

Chen et al. (2024) explores logistics mode choice but lack dynamic learning or cooperative outcomes. This study builds on this with adaptive pricing and service optimization.

### Capacity Sharing Between Competitors, (Guo & Wu, 2018)

Guo & Wu, (2018) discusses the economics of capacity sharing between rivals but does not model long-term learning or reinforcement dynamics. The RL framework in this study adds this missing dimension.

### The strategic role of logistics in the industry 4.0 era, (Tang & Veelenturf, 2019).

Tang & Veelenturf, (2019) reviewed digital platforms but did not formalise TPLP strategic behaviours. This study adds them as explicit policy optimizers.

### Logistics choices in a platform supply chain: A co-opetitive perspective, (Wang et al., 2022)

Wang et al., (2022) included basic coopetition logic but lacked a feedback-driven dynamic system. The RL-enhanced game model aims to target this gap.

### Section summary

This section examined how existing research has addressed strategic dynamics among e-commerce platforms, sellers, and third-party logistics providers. While many studies highlight pricing, channel conflicts, or trust mechanisms, most adopt either a platform-centric or seller-focused view, often treating the TPLP as a passive executor rather than a strategic player. Additionally, few works incorporate the temporal evolution of strategies or the indirect feedback loops that arise in coopetitive contexts. This study fills that void by giving the TPLP an active, decision-making role within a reinforcement learning and game-theoretic framework, enabling it to adaptively shape the balance of cooperation and competition through service levels and pricing.

## Game Theory in Supply Chain Management

### A Stackelberg-game model in a two-stage supply chain, Qin (2012)

Y. Qin, (2012) models Stackelberg leadership in supply chains but in a simplified two-player setting. This study extends this hierarchy to a three-agent structure with added learning dynamics.

### Service competition in an online duopoly market, (Ding et al., 2018)

Ding et al., (2018) consider competitive service dynamics but only in a static two-player setting. This study introduces adaptive strategy updates and triadic coordination.

### Channel Dynamics Under Price and Service Competition, (Tsay & Agrawal, 2000)

Tsay & Agrawal, (2000) identifies coordination issues but does not use game theory to resolve them dynamically. The Stackelberg model introduced builds a hierarchy to reduce such failures.

### A General Equilibrium Model for Industries with Price and Service Competition, (Bernstein & Federgruen, 2004)

Bernstein & Federgruen, (2004) focus on simultaneous-move games in a competitive landscape and lacks asymmetric roles and learning behaviors. This study introduces hierarchical decision-making.

### Game-theoretic analysis of a two-stage dual-channel supply chain coordination in the presence of market segmentation and price discounts, (Zhao & Li, 2023)

Zhao & Li, (2023) provides valuable insights into pricing strategies and coordination mechanisms but does not explore dynamic, adaptive interactions among supply chain participants. This study incorporates RL to model how TPLPs adapt their strategies in response to evolving market conditions.

### Game theoretical analysis of incumbent platform investment and the supplier entry strategies in an e-supply chain, (Zhuo et al., 2024)

(Zhuo et al., 2024) insights into platform-supplier dynamics but does not explore dynamic, adaptive interactions among supply chain participants. This study incorporates RL to model how TPLPs adapt their strategies in response to evolving market conditions.

### Section summary

The literature on game theory approaches has successfully captured multi-agent interactions in supply chain settings, often through Stackelberg or Nash equilibrium formulations. However, these studies typically rely on analytical equilibria often exclude adaptive learning or feedback effects. Additionally, few have examined the TPLP as a strategic leader capable of influencing channel coordination. This study fills that gap by integrating reinforcement learning to enable strategic adaptation over time, allowing the TPLP to test, refine, and optimize its pricing and service levels in reaction to market responses rather than relying on static equilibria.

## Reinforcement Learning for Supply Chains

### Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities, (Yan et al., 2022)

Yan et al., (2022) provide a comprehensive survey of reinforcement learning applications in logistics and supply chain management including the use of PPO. However, there has not been many applications in the e-commerce industry, where this study can contribute to.

### Reinforcement learning for solving the vehicle routing problem, (Nazari et al., 2018)

Nazari et al., (2018) applied RL to vehicle routing, but did not address strategic multi-agent settings. This study extends RL to include inter-agent coordination.

### Dynamic Pricing Model of E-Commerce Platforms Based on Deep Reinforcement Learning, (Yin & Han, 2021)

Yin & Han, (2021) develop a dynamic pricing model using deep reinforcement learning to optimize e-commerce platform revenues under fluctuating consumer demand but does not consider the influence of logistics decisions or multi-agent coordination. This study complements this by integrating logistics service adaptation and inter-agent dynamics between platforms, sellers, and third-party logistics providers.

### A multi-agent reinforcement learning model for inventory transshipments under supply chain disruption, (Kim et al., 2024)

Kim et al., (2024) uses a multi-agent reinforcement learning (MARL) framework to coordinate inventory transshipments among retailers during supply chain disruptions. However, it does not address the strategic interactions between e-commerce platforms and third-party logistics providers. This study builds upon this by incorporating such multi-agent dynamics, focusing on adaptive logistics service strategies in response to such market volatility.​

### Section Summary

Existing studies on supply chain reinforcement learning focus more on qualitative mechanisms or deterministic modeling. Few attempt to simulate how logistics providers may iteratively develop strategies to handle fluctuating demand and cooperation thresholds. By modeling the logistics service sharing setup as a dynamic environment with a learning TPLP agent, this study captures those nuances and extends the realism of such frameworks into adaptive, simulation-based methodologies that reflect actual operational uncertainty.

# Simulation environment design and experimental framework

This study employs a hybrid framework combining game theory and reinforcement learning to analyse the strategic interactions between Third-Party Logistics Providers (TPLPs), E-tailers, and sellers. The methodology builds upon Qin et al. (2020) by introducing a Stackelberg game model and a dynamic simulation environment enabled by reinforcement learning. These components allow for real-time strategy adjustments, capturing the complexities of modern supply chains. This methodology fills key gaps in previous research by combining reinforcement learning with game theory to model the evolving strategies of all players in a logistics service sharing scenario. Earlier studies mainly used static, one-shot game models that did not account for how decisions change over time. Importantly, the TPLP, treated passively in prior work, is given an active role in adjusting service levels and pricing. This allows us to explore how all three players respond to each other’s actions, capturing the strategic nature of logistics service sharing in a cooperative and competitive (coopetive) environment.

## Overview of the context and key variables

Qin et al. (2020) examined logistics service sharing between e-commerce platforms and logistics providers using a static game-theoretic model. The Nash equilibrium approach revealed how cooperative strategies could optimize resource utilization and improve profitability. However, their model assumed static interactions and did not account for dynamic behaviours, fluctuating demand, or competition from TPLPs.

By incorporating hierarchical decision-making (Stackelberg game) and dynamic learning (PPO), this study extends Qin et al.’s framework to address these limitations. The dynamic approach allows stakeholders to adapt to changing market conditions, providing a more realistic and actionable representation of supply chain interactions.

Table 1below summarises the important parameters and variables which will be utilized in the development of this model.

Table 1: Important parameters and variables

|  |  |
| --- | --- |
| **Notation** | **Descriptions** |
|  | Market players, where 1 represents the E-tailer, 2 denotes the seller |
|  | Logistic service providers, where 1 represents the E-tailer, 2 denotes the TPLP |
|  | Alternative modes, where N indicates the No-Service sharing mode and S indicates the Service sharing mode |
|  | Player i’s retail price in mode m |
|  | The service level of player i’s (adopted) logistics service system in mode m. This is in the range of 0-10, where 0 represents 0% and 10 represents 100% service level. Service level is the percentage of customer requests that are fulfilled within the agreed-upon time. |
|  | Player i’s demand in mode m |
|  | Player i’s profit in mode m |
|  | The unit price of the logistics service offered by the E-tailer |
|  | The unit price of the logistics service offered by the TPLP |
|  | The variable cost of the logistics service incurred by the provider . |
|  | The commission rate charged by the E-tailer to seller |
|  | The commission rate charged by the E-tailer to TPLP |
|  | The mean of the random market potential |
|  | The sensitivity of player i’s demand to his/her rival’s retail price, 0 < α < 1 |
|  | The sensitivity of player i’s demand to his/her own logistics service level, β > 0 |
|  | The sensitivity of player i’s demand to his/her rival’s logistics service level, γ < β |
|  | Maximum capacity of E-tailer |

Figure 1 below shows the interactions between the various players in this proposed model.

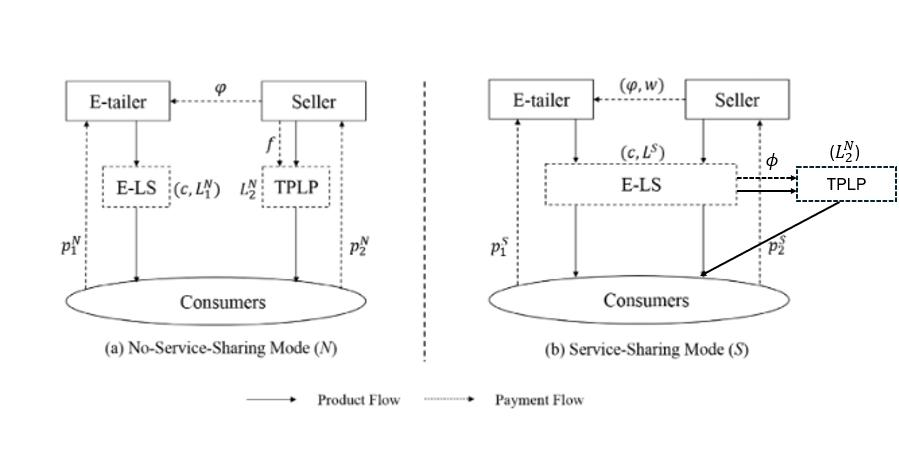


Figure 1: Model Structure

To summarise, in the No-service sharing mode, where the E-tailer does not share his logistics service with the seller, the seller utilizes the TPLP and pays a logistics fee . He also pays a commission rate to the E-tailer for usage of their platform. In the service sharing Mode, the E-tailer shares his logistics service with the seller. The seller pays a logistics fee ontop of the commission rate . In the service sharing mode, however, if the E-tailer exceeds his allocated capacity , he will pass on the excess shipment to the TPLP and charge a commission rate of The specific equations for the simple demand faced by the seller and E-tailer (excluding capacity constraint) and derivations are explained in detail in the original study by Qin et al. (2020).

## Stepwise Approach: Base Model, Game Theory, and Reinforcement Learning Integration

This section outlines the modifications and enhancements made to the original model. It will be decomposed into three sections. Firstly, the reproduction of the model by Qin et al. (2020) and the detailed derivations of the adaptations to the model. The second section will introduce the solving of the Stackelberg game between the E-tailer and seller, and lastly, the reinforcement learning process between the TPLP (leader) and the followers (E-tailer and seller). All the additional codes are provided in the Appendix.

### Reproduction of original model and adaptations to the model

In reproducing the original model, the logistics service sharing decisions described in Qin et al. (2020) is simulated. Using Python’s object-oriented programming, this script encapsulates the strategic decision-making framework into a class-based structure. It defines key economic parameters, equilibrium pricing conditions, demand computations, and profit models to replicate the theoretical findings of the study. This section provides an in-depth explanation of the implementation, focusing on how Python classes define the logistics service sharing environment. Python provides an efficient way to represent real-world systems using classes and objects. In this implementation, the *LogisticsServiceModel* class encapsulates the economic and strategic components required for analysing logistics service sharing. The object-oriented approach allows modular code design, making it easier to modify and extend which will further explained in the adaptations section.

In Section 3.2.1.1, the key parameters, pricing strategies and demand faced by the individual players will be expounded upon. These are reproduced from the equations derived by Qin et al. (2020).

Section 3.2.1.2 will introduce the adaptations made to the original model by Qin et al. (2020). This will include the equations relating to the excess demand faced by the E-tailer (or spare capacity from the perspective of the TPLP) and the profit functions of all the players, namely the E-tailer, seller and TPLP. Additionally, the assumptions to the equations will be stated and explained.

#### Reproduction of the original model

This section will detail the reproduction of the original model. As such, this will only include the equations which describes the interaction between the E-tailer and seller, while taking the actions of the TPLP as exogenous.

##### Class Definition and Initialization

The *LogisticsServiceModel* class is initialized with key parameters that define the logistics environment. These parameters include market potential (), logistics service levels (), fee/cost factors (), and competitive elasticity coefficients (). This is shown in Figure 2 below. These variables serve as the backbone of the entire model as the subsequent equations will reference them.

class LogisticsServiceModel:

    def \_\_init\_\_(self, L\_s,theta,f=1,L\_e=10, phi=0.05, alpha=0.5, beta=0.7, gamma=0.5, c=0.5):

        self.L\_e = L\_e  # E-tailer's logistics service level

        self.L\_s = L\_s  # TPLP's logistics service level

        self.phi = phi  # Commission rate charged to seller

        self.theta = theta # Market potential

        self.alpha = alpha # The sensitivity of player i’s demand to his/her rival’s retail price, 0 < α < 1

        self.beta = beta # The sensitivity of player i’s demand to his/her own logistics service level, β > 0

        self.gamma = gamma # The sensitivity of player i’s demand to his/her rival’s logistics service level, γ < β

        self.c = c      # Variable cost of logistics for E-tailer

        self.f = f      # TPLP logistics fee

        self.max\_capacity = 5 # E-tailer's maximum fulfilment capacity

        self.commission = 0.5 # If E-tailer has unfulfilled demand, he still takes a cut from tplp

Figure 2: Key parameters of the model

The initialization ensures that each instance of the class captures the dynamics of logistics service decisions. These parameters correspond directly to the economic model in Qin et al. (2020), where market potential and service levels influence profitability and strategic decisions.

##### Pricing strategies for E-tailer and seller

Pricing strategies are a crucial part of the logistics decision-making process. The model computes optimal pricing for both in the sharing and no-sharing cases, aligning with the equilibrium conditions described in Qin et al. (2020). The derived prices here will subsequently be used for profit calculation of the E-tailer and seller. The derivations for the optimal prices are detailed by Qin et al. (2020). The functions *p1\_sharing* and *p2\_sharing* shown in Figure 3 below compute the optimal price points set by the E-tailer and seller under the assumption of logistics service sharing to maximise their individual profit, where refers to the E-tailer and the seller.

def p1\_sharing(self,ww):

        top = (

            -self.L\_e \* self.alpha \* self.beta \* self.phi\*\*2 +

            self.L\_e \* self.alpha \* self.beta +

            self.L\_e \* self.alpha \* self.gamma \* self.phi\*\*2 -

            self.L\_e \* self.alpha \* self.gamma -

            2 \* self.L\_e \* self.beta \* self.phi +

            2 \* self.L\_e \* self.beta +

            2 \* self.L\_e \* self.gamma \* self.phi -

            2 \* self.L\_e \* self.gamma -

            self.theta \* self.alpha \* self.phi\*\*2 +

            self.theta \* self.alpha -

            2 \* self.theta \* self.phi +

            2 \* self.theta +

            2 \* self.alpha \* self.c \* self.phi -

            2 \* self.alpha \* self.c -

            self.alpha \* self.phi \* self.calc\_w(ww) +

            3 \* self.alpha \* self.calc\_w(ww) -

            2 \* self.c \* self.phi +

            2 \* self.c

        )

        bottom = self.alpha\*\*2 \* self.phi\*\*2 - self.alpha\*\*2 - 4 \* self.phi + 4

        return top/bottom

    def p2\_sharing(self,ww):

        top = (

            -self.L\_e \* self.alpha \* self.beta \* self.phi +

            self.L\_e \* self.alpha \* self.beta +

            self.L\_e \* self.alpha \* self.gamma \* self.phi -

            self.L\_e \* self.alpha \* self.gamma -

            2 \* self.L\_e \* self.beta \* self.phi +

            2 \* self.L\_e \* self.beta +

            2 \* self.L\_e \* self.gamma \* self.phi -

            2 \* self.L\_e \* self.gamma -

            self.theta \* self.alpha \* self.phi +

            self.theta \* self.alpha -

            2 \* self.theta \* self.phi +

            2 \* self.theta +

            self.alpha\*\*2 \* self.c \* self.phi -

            self.alpha\*\*2 \* self.c -

            self.alpha\*\*2 \* self.phi \* self.calc\_w(ww) +

            self.alpha\*\*2 \* self.calc\_w(ww) -

            self.alpha \* self.c \* self.phi +

            self.alpha \* self.c +

            2 \* self.calc\_w(ww)

        )

        bottom = self.alpha\*\*2 \* self.phi\*\*2 - self.alpha\*\*2 - 4 \* self.phi + 4

        return top/bottom

Figure 3: E-tailer and seller pricing strategies under sharing

These functions implement the equilibrium pricing equations found in Qin et al. (2020). As observed, the equations here are not a function of , the TPLP’s logistics service level, as the seller is utilising the seller’s platform, hence inheriting the E-tailer’s service level, ( instead. Additionally, the consider commission rates, cost functions, and competitive effects to determine the pricing structure that maximizes profit for both parties under logistics service sharing. The variable in the equation represents whether there is a win-win scenario for both the seller and E-tailer prior to and after logistics service sharing. This will be further explained subsequently.

In contrast, the functions *p1\_nosharing* and *p2\_nosharing* shown in Figure 4 provide alternative price computations when logistics services are not shared between E-tailers and sellers.

    def M1(self):

        return (1 - self.phi) \* (self.theta \* (2 + self.alpha \* (1 + self.phi)) + 2 \* self.c +

                                 self.L\_e \* (2 \* self.beta - self.alpha \* self.gamma \* (1 + self.phi)) +

                                 self.L\_s \* (self.alpha \* self.beta \* (1 + self.phi) - 2 \* self.gamma)) + self.alpha \* self.f \* (1 + self.phi)

    def M2(self):

        return (1 - self.phi) \* (self.theta \* (2 + self.alpha) + self.alpha \* self.c +

                                 self.L\_e \* (self.alpha \* self.beta - 2 \* self.gamma) +

                                 self.L\_s \* (2 \* self.beta - self.alpha \* self.gamma)) + 2 \* self.f

def p1\_nosharing(self):

        return self.M1()/((1-self.phi)\*(4-self.alpha\*\*2\*(1+self.phi)))

    def p2\_nosharing(self):

        return self.M2()/((1-self.phi)\*(4-self.alpha\*\*2\*(1+self.phi)))

Figure 4: E-tailer and seller pricing strategies under no-sharing

The functions and simply serve as constants for simplification of subsequent equations. Comparing to the functions *p1\_sharing* and *p2\_sharing*, the functions *p1\_nosharing* and *p2\_nosharing* are now a function of the logistics service level of the TPLP (. This is because the seller is now utilising the TPLP’s platform, inheriting his service level.

By comparing the results of these pricing functions, the model can utilise these set prices to determine the corresponding of the E-tailer and seller in the service sharing and non-sharing mode and determine when service sharing is advantageous based on the exogenous market potential and TPLP service levels.

In the previous equations, a term surfaced in the calculation of variables in the sharing mode. The term *ww* refers to the win-win condition. This refers to when both the seller and E-tailer will see higher profits in the service sharing mode compared to the no-sharing mode. This win-win condition is determined True or False depending on the unit price of the logistics service offered by the E-tailer, *w*. Intuitively, a higher *w* indicates lower profits for the seller and higher profits for the E-tailer in the sharing mode. Based on the findings by Qin et al. (2020), there exists a range of *w* the E-tailer can set, such that the seller will be willing or unwilling to accept logistics sharing, while the E-tailer benefits regardless, albeit at different extents.

When *ww* is True, this is the condition where the E-tailer can charge his profit maximising price *w* to the seller, with the seller still agreeing to logistics service sharing. When *ww* is False, this is the condition where the seller initially suffers from logistics service sharing. To incentivise the seller to accept, the E-tailer will lower *w* up till the point where the seller will agree with logistics service sharing.

In both cases, the E-tailer gains from logistics service sharing. In simpler terms, if there exists a *w* the E-tailer can set such that he benefits from service sharing, logistics service sharing will always occur. The only difference is the extent of benefit the E-tailer and seller gains depending on *w*. This means that *w* will always be lower when *ww* is False, and vice versa.

As such, the function *calc\_w* in Figure 5 calculates the logistics service price *w* under these two different conditions.

    def calc\_w(self,ww):

        if ww == True:

            result = (

                ((self.phi - 1) \* (

                    8 \* self.c + 8 \* self.theta + 8 \* self.L\_e \* self.beta - 8 \* self.L\_e \* self.gamma -

                    8 \* self.alpha \* self.c - 8 \* self.phi \* self.theta + 2 \* self.alpha\*\*2 \* self.c -

                    3 \* self.alpha\*\*3 \* self.c + self.alpha\*\*4 \* self.c + self.alpha\*\*3 \* self.theta -

                    2 \* self.alpha\*\*2 \* self.phi \* self.theta + self.alpha\*\*3 \* self.c \* self.phi\*\*2 -

                    self.alpha\*\*4 \* self.c \* self.phi\*\*2 + 2 \* self.alpha\*\*2 \* self.phi\*\*2 \* self.theta -

                    self.alpha\*\*3 \* self.phi\*\*2 \* self.theta - 8 \* self.L\_e \* self.beta \* self.phi +

                    8 \* self.L\_e \* self.gamma \* self.phi - 4 \* self.alpha \* self.c \* self.phi +

                    4 \* self.alpha \* self.phi \* self.theta + self.L\_e \* self.alpha\*\*3 \* self.beta -

                    self.L\_e \* self.alpha\*\*3 \* self.gamma + 2 \* self.alpha\*\*2 \* self.c \* self.phi +

                    2 \* self.alpha\*\*3 \* self.c \* self.phi - 2 \* self.L\_e \* self.alpha\*\*2 \* self.beta \* self.phi +

                    2 \* self.L\_e \* self.alpha\*\*2 \* self.gamma \* self.phi +

                    2 \* self.L\_e \* self.alpha\*\*2 \* self.beta \* self.phi\*\*2 -

                    self.L\_e \* self.alpha\*\*3 \* self.beta \* self.phi\*\*2 -

                    2 \* self.L\_e \* self.alpha\*\*2 \* self.gamma \* self.phi\*\*2 +

                    self.L\_e \* self.alpha\*\*3 \* self.gamma \* self.phi\*\*2 +

                    4 \* self.L\_e \* self.alpha \* self.beta \* self.phi -

                    4 \* self.L\_e \* self.alpha \* self.gamma \* self.phi

                )) / (

                    2 \* (self.alpha\*\*3 \* self.phi\*\*2 - 2 \* self.alpha\*\*3 \* self.phi +

                        self.alpha\*\*3 - self.alpha\*\*2 \* self.phi\*\*2 + 2 \* self.alpha\*\*2 \* self.phi -

                        self.alpha\*\*2 - 4 \* self.alpha \* self.phi + 8 \* self.alpha +

                        4 \* self.phi - 8)

                )

            )

            return result

        elif ww == False:

            result = (

                (2 \* self.L\_s \* self.beta - 2 \* self.L\_e \* self.beta - 2 \* self.f + self.alpha\*\*2 \* self.c +

                self.alpha\*\*2 \* self.f + self.L\_e \* self.alpha \* self.gamma - self.L\_s \* self.alpha \* self.gamma +

                2 \* self.L\_e \* self.beta \* self.phi - 2 \* self.L\_s \* self.beta \* self.phi - self.alpha\*\*2 \* self.c \* self.phi +

                self.alpha\*\*2 \* self.f \* self.phi - self.L\_e \* self.alpha \* self.gamma \* self.phi +

                self.L\_s \* self.alpha \* self.gamma \* self.phi)

                / (2 \* (self.alpha\*\*2 - 1))

            )

            return result

Figure 5: Calculation of logistics service cost

The derivations of these equations are obtained from the Online Appendix of Qin et al. (2020).

##### Demand Functions

The demand faced by the E-tailer and seller separately are modelled using mathematical expressions that consider both price and logistics service levels. Qin et al. (2020) highlight that demand is influenced by logistics efficiency and competitive pricing as shown in the general equation in Figure 6 below , which is fairly common in marketing and operations literature (Bernstein & Federgruen, 2004), (X. Chen et al., 2019), (Ding et al., 2018), (Tsay & Agrawal, 2000). To put simply, the demand faced by a party is an increasing function of the exogenous market demand, , the competitor’s price level, and his logistics service level, . On the other hand, it is a decreasing function of his own price level, and his competitor’s logistics service level, . This simple model is intuitive. The extent to which demand is affected by these variables are established by the sensitivity coefficients as explained earlier. The derived demand here will subsequently be used for profit calculation of the E-tailer, seller and the TPLP themselves.



Figure 6: General demand equation

The functions *D\_sharing\_etailer* and *D\_sharing\_seller* shown in Figure 7 below calculates the demand faced by E-tailer and seller when the logistics service is shared.

    def D\_sharing\_etailer(self,ww):

        return self.theta - self.p1\_sharing(ww) +self.alpha \* self.p2\_sharing(ww) + self.beta \* self.L\_e - self.gamma \* self.L\_e

    def D\_sharing\_seller(self,ww):

        return self.theta - self.p2\_sharing(ww) +self.alpha \* self.p1\_sharing(ww) + self.beta \* self.L\_e - self.gamma \* self.L\_e

Figure 7: E-tailer and seller demand under sharing

Like the pricing functions, these are not functions of , the TPLP’s logistics service level as the seller is utilising the seller’s logistics platform.

The functions *D\_nosharing\_etailer* and *D\_nosharing\_seller* shown in Figure 7 below calculates demand when sellers rely on TPLPs. Like the pricing functions, these are now functions of , the TPLP’s logistics service level as the seller is utilising the TPLP’s logistics platform. A point to note is that the function *D\_nosharing\_seller* conveniently provides the demand faced by the TPLP under no-sharing condition as they will be equivalent.

    def D\_nosharing\_etailer(self):

         return self.theta - self.p1\_nosharing() +self.alpha \* self.p2\_nosharing() + self.beta \* self.L\_e - self.gamma \* self.L\_s

    def D\_nosharing\_seller(self):

         return self.theta - self.p2\_nosharing() +self.alpha \* self.p1\_nosharing() + self.beta \* self.L\_s - self.gamma \* self.L\_e

Figure 8: E-tailer and seller demand under no-sharing

Overall, these demand computations provide the demand faced by all the players and will be used extensively in the profit calculations.

#### Adaptations to the original model

This section will detail the adaptations to the original model. This will include the equations relating to the excess demand faced by the E-tailer (or spare capacity from the perspective of the TPLP) and the profit functions of all the E-tailer, seller and TPLP.

##### Excess demand calculation

An inherent assumption of this model is that TPLPs have higher service capacity as compared to the E-tailers themselves. This aligns with current literature that E-commerce companies, especially those that are primarily online (net-based firms), often lack the necessary logistics infrastructure and expertise. This deficiency can hinder their operational efficiency. As such, many e-commerce platforms opt to collaborate with TPLPs as these providers offer specialized logistics services, allowing e-commerce companies to focus on their core competencies while ensuring efficient product delivery (Joong‐Kun Cho et al., 2008).

Additionally, this situation of excess demand only occurs when sharing mode exists between the seller and E-tailer. This is because in the no-service sharing mode, it is assumed that the E-tailer will be able to sufficiently cater to all their demand. This is a reasonable assumption as in many real-world hybrid e-commerce models (e.g. Amazon, JD.com), platforms invest heavily in their own logistics infrastructure warehousing, last-mile delivery, etc. precisely to ensure a high level of control and service reliability for orders placed directly with them. This means the E-tailer’s system is typically sized to meet their own demand, not to handle external (seller) demand. As a result, under no-sharing, there should be no excess burden (Zennaro et al., 2022).

The function *calc\_excess\_demand* in Figure 9 below determines whether demand exceeds the logistics service capacity, , of the E-tailer and hence calculates the excess capacity to be passed on to the TPLP for handling.

    def calc\_excess\_demand(self, ww):

        demand\_sharing\_etailer = self.D\_sharing\_etailer(ww)

        demand\_sharing\_seller = self.D\_sharing\_seller(ww)

        total\_demand = demand\_sharing\_etailer + demand\_sharing\_seller

        # Check if the total demand exceeds the maximum capacity

        if total\_demand > self.max\_capacity:

            if demand\_sharing\_etailer <= self.max\_capacity:

                excess = total\_demand-self.max\_capacity

                top\_up = demand\_sharing\_seller - excess

                for\_tplp = max (0,demand\_sharing\_seller - top\_up)

            else:

                for\_tplp = self.max\_capacity - demand\_sharing\_etailer + demand\_sharing\_seller

        else:

            for\_tplp = 0

        return for\_tplp

Figure 9: Excess demand calculation

To explain the equation in simple terms, the function first checks if the combined demand arising from service sharing exceeds the maximum capacity of the E-tailer. If no, no additional capacity will be passed on to the TPLP. If yes, the function checks if the E-tailer has sufficient additional capacity to absorb part of the demand from the seller and determines the left-over amount the TPLP has to handle. If there is no additional capacity, the TPLP will handle all the excess amount from both the seller and E-tailer. Figure 10 below is a flowchart which summarises the logic behind this equation.

A diagram of a flowchart

AI-generated content may be incorrect.

Figure 10: Excess demand flowchart

##### Profit Calculation for seller and E-tailer

Profit functions are integral to the decision-making process, as they determine whether logistics sharing is financially viable. The profit functions are computed for both in the no-sharing mode and sharing mode. Theoretically, if the profits for both the seller and E-tailer are higher in the sharing mode, logistics service sharing will occur between these two parties. Conversely, if either one of the parties suffer a loss in profits in the sharing mode, logistics service sharing will not occur between the two parties.

The functions *profit\_nosharing\_etailer* and *profit\_nosharing\_seller* in Figure 11 below compute profits when logistics services are not shared.

    def profit\_nosharing\_etailer(self):

        M1 = self.M1()

        M2 = self.M2()

        N1 = self.N1()

        N2 = self.N2()

        bottom = (1 - self.phi) \* (4 - self.alpha\*\*2 \* (1 + self.phi))\*\*2

        return max(((N1 \* (M1 - self.c \* (1 - self.phi) \* (4 - self.alpha\*\*2 \* (1 + self.phi))) +

                self.phi \* M2 \* N2 / (1 - self.phi)) / bottom),0)

    def profit\_nosharing\_seller(self):

        N2 = self.N2()

        bottom = (1 - self.phi) \* (4 - self.alpha\*\*2 \* (1 + self.phi))\*\*2

        return max(((N2\*\*2) / bottom),0)

Figure 11: Profits for E-tailer and seller under no-sharing

These functions consider logistics costs, commission rates, and demand levels to compute profitability under independent service operations. The function is derived from the original profit function in Figure 12 below. This is a standard profit function where the profit earned by the E-tailer (Player 1) is simply the revenue earned minus off the costs associated with logistics, adding in the platform commission rate of . For the seller (Player 2), it is the revenue earned minus off the logistics fee paid to the TPLP and the commission rate charged by the E-tailer. As you notice, these profit functions did not use the previously defined demand functions, as these functions are simplified down to the basic variables as shown in Figure 13 below.

A group of math symbols

AI-generated content may be incorrect.

Figure 12: Profit function for E-tailer and seller under no-sharing

A math equations on a white background

AI-generated content may be incorrect.

Figure 13: Simplified profit equation for E-tailer and seller under no-sharing

The functions *profit\_sharing\_etailer* and *profit\_sharing\_seller* in Figure 14 below compute profits when logistics services are shared.

    def profit\_sharing\_etailer(self,ww):

        unfulfilled\_demand = self.calc\_excess\_demand(ww)

        retained\_profit = self.commission \* unfulfilled\_demand \* (self.calc\_w(ww))

        total\_profit = (self.p1\_sharing(ww)-self.c)\*(self.D\_sharing\_etailer(ww)) + \

        (self.phi\*self.p2\_sharing(ww)+self.calc\_w(ww)-self.c)\*(self.D\_sharing\_seller(ww)-unfulfilled\_demand)+retained\_profit

        return max(total\_profit,0)

    def profit\_sharing\_seller(self,ww):

        total\_profit = ((1-self.phi)\*self.p2\_sharing(ww)-self.calc\_w(ww)) \* (self.D\_sharing\_seller(ww))

        if ww == True:

            return max(total\_profit,0)

Figure 14: Profits for E-tailer and seller under sharing

First, looking at *profit\_sharing\_seller,* this follows the standard profit equation in Figure 15 below. Simply put, the profit earned by the seller is simply the revenue earned minus off the costs associated. This is calculated by multiplying the demand he faces under service sharing by a fraction (of the price he charges the consumers due to platform fees, and minus off the logistics service costs he pays to the E-tailer.



Figure 15: Equation for seller under sharing

Looking at *profit\_sharing\_etailer*, the equation is slightly different. The E-tailer earns profit from three outlets:

1. Standard profit from selling goods faced by his own demand (first part of *total\_profit*)
2. Profits earned by platform rates and logistics service fee charged to seller (second part of *total\_profit*)
3. Profits earned from charging commission to the TPLP if any excess demand is passed on to them (third part of *total\_profit or retained\_profit*)

Outlets 1 and 2 are described by the equation in Figure 16 below.



Figure 16: First profit equation for E-tailer under sharing

Outlet 3 is described by the equation in Figure 17 below, where the formula for *excess demand* has been highlighted in Figure 9 previously.

A close-up of a word

AI-generated content may be incorrect.

Figure 17: Second profit equation for E-tailer under sharing

To put in simple terms, the E-tailer takes a cut of the total logistics service fee () he charges the seller.

In the subsequent section, the demand and profit function of the TPLP will be expounded upon.

##### Demand calculation for TPLP

The demand calculation for the TPLP is trivial and has been previously calculated. Firstly, in the no-service sharing mode, the demand faced by the TPLP would just be equal to the demand faced by the seller, as evaluated in Figure 8. In the service sharing mode, the demand would just be equal to the excess demand as evaluated in Figure 9.

##### Profit calculation for TPLP

The profit calculation for the TPLP under both sharing and no-sharing conditions are shown in Figure 18 below. The equations reflect the classic revenue-minus-cost structure as in Figure 15.

    def profit\_sharing\_tplp(self, ww):

        a = 0.05

        b = 0.2

        c = 0.2

        excess\_demand = self.calc\_excess\_demand(ww)

        retained\_revenue\_per\_unit = (1-self.commission) \* (self.calc\_w(ww)) + self.f

        cost\_per\_unit = a \* self.L\_s + b \* self.f + c \* self.f\*\*2

        total\_profit = (retained\_revenue\_per\_unit - cost\_per\_unit) \* excess\_demand

        return max(total\_profit,0)

    def profit\_nosharing\_tplp(self):

        a = 0.05

        b = 0.2

        c = 0.2

        cost\_per\_unit = a \* self.L\_s + b \* self.f + c \* self.f\*\*2

        total\_volume = self.D\_nosharing\_seller()

        total\_profit = (self.f-cost\_per\_unit)\*total\_volume

        return max(total\_profit,0)

Figure 18: Profits for TPLP under sharing and no-sharing

In both scenarios, the TPLP faces the same cost function. The cost per unit of shipment function is shown in Figure 19 below.

A black text with a plus and a black symbol

AI-generated content may be incorrect.

Figure 19: TPLP cost function

This function encapsulates the increasing marginal cost nature of logistics operations and each term is economically motivated:

* **Service Level Cost ():** The TPLP's service level is interpreted as a proxy for delivery speed, reliability, or geographic coverage and is linearly tied to cost. Higher service levels mean more fleet deployment, tighter dispatch schedules, or enhanced infrastructure. The linear term reflects proportional investment or operational expenditure needed to sustain those enhancements. This aligns with the framework by (Bijulal et al., 2011) where they utilised a linear relationship between average system costs and inventory levels (which signifies a higher service level).
* **Price-Linked Cost Terms ()**: Including the logistics price in the cost function recognizes that setting higher logistics prices is not cost-neutral for the TPLP. ​In the context of third-party logistics providers (3PLs), there is evidence suggesting that increasing the price charged for services can lead to higher per-unit delivery costs. This phenomenon is primarily due to the need for enhanced service offerings and the associated operational complexities that arise with premium pricing strategies (Ülkü & Bookbinder, 2012), (Zhang et al., 2015). In terms of the specific functional form of the cost, more research can be done to justify this. However, as this model is a simulation, a simple quadratic function should suffice.

In the sharing scenario, In the logistics sharing scenario, the TPLP acts as a residual service provider, fulfilling excess demand offloaded by the E-tailer due to its capacity limitations. The revenue earned by the TPLP is related to the profit earned by the E-tailer due to commission as evaluated in Figure 17. The revenue earned is simply taking the rest of the amount of ( of the total logistics fee (charged by the E-tailer to the seller) as retained revenue but also adding in the TPLP’s own logistic fee as additional revenue. For simplicity, this additional fee is initially absorbed by the E-tailer but offset through increased prices for the end consumers. As such, this is not reflected in the E-tailer’s profit function.

In the no-sharing scenario, the seller outsources logistics exclusively to the TPLP. The profit function is defined in Figure 20 below. The profit earned by the TPLP is equal to the total shipment multiplied by the revenue earned per unit of shipment (which is equal to the logistics fee charged, ) subtracting the cost per unit of shipment .

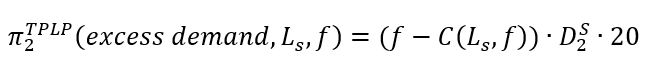


Figure 20: TPLP profit function under no-sharing

#### Graphical representations of preliminary investigations

The *LogisticsServiceModel* script successfully replicates the equilibrium conditions presented in Qin et al. (2020). By encoding pricing strategies, demand computations, and profit models into Python classes, this implementation transforms Qin et al.’s (2020) analytical framework into a computationally testable model using Python. By defining an object-oriented structure, the *LogisticsServiceModel* class systematically encodes economic dynamics, allowing robust simulation of logistics service sharing scenarios. The model provides an enhanced understanding of how logistics service sharing impacts competitive strategy, validating the original findings while offering new insights through computational experimentation. In this section, graphical representations of some of the findings of interactions between the E-tailer and seller are shown.

##### Profit zones for E-tailer and seller prior to capacity constraint

The main research objective in the original paper by Qin et al. (2020) is figuring out the conditions at which logistics service sharing occurs between the E-tailer and seller, specifically with respect to the market potential level, and TPLP service level, . Using the equations derived in the previous sections, Figure 21 below is produced to show the zones at which service sharing occurs, prior to capacity constraint faced by the E-tailer. The assumption here is that both the seller and E-tailer chooses prices which maximises their total profit at the various levels of and . The E-tailer does not make any concessions as mentioned earlier to the level of *w* for the seller.

A red and blue triangle

Description automatically generated

Figure 21: Profit zones for E-tailer and seller (

The figure illustrates the beneficial and non-beneficial regions for the E-tailer and the seller under varying levels of θ and 𝐿ₛ, while keeping the other constants at fixed values. On the graph, “profit regions” refers to having higher profits after logistics service sharing as compared to no-sharing. The blue zone refers to the E-tailer benefitting while seller benefitting. The grey zone refers to the E-tailer benefitting while seller suffering. Lastly, the red zone refers to both the E-tailer and seller benefitting under logistics service sharing. This figure shows the zones of and at which the seller and E-tailer will participate in logistics sharing, assuming the E-tailer makes no concessions to *w*.

To summarise, this plot classifies outcomes into three distinct regions:

* **WW (Win-Win, grey area)**: Both the E-tailer and seller achieve higher profits under logistics service sharing than in the no-sharing scenario.
* **WL (Win-Lose, red area)**: The E-tailer benefits from logistics sharing, but the seller’s profit is reduced.
* **LW (Lose-Win, blue area)**: The seller gains from sharing, but the E-tailer incurs a loss.

The dynamic threshold between the regions is influenced by the trade-off between service level provided by the TPLP and the attractiveness of logistics service sharing for the players.

In the WW region, both parties see improvement, making service sharing an obvious choice. In the WL region, the E-tailer can still enforce sharing by adjusting *w* downward to induce seller participation as mentioned earlier. This is a core result from Qin et al. (2020) suggesting that if the E-tailer’s profit is non-negative and the seller’s profit can be nudged above its no-sharing level, logistics sharing will proceed.

Figure 22 below shows the “profit zones” instead when such a concession is made by the E-tailer.

A red and blue rectangle

AI-generated content may be incorrect.

Figure 22: Profit zones for E-tailer and seller ( when E-tailer makes concessions

As shown in the figure and comparing to Figure 21, the entire grey portion has now become red. This shows that if the E-tailer profits, there always exists a *w\* (< original w)* such that the seller will be incentivized to participate in service sharing.

##### Profit zones for E-tailer and seller prior after capacity constraint

Due to the addition of capacity constraints for the E-tailer, the “profit zones” will inevitably shift. Assuming a capacity constraint, , of 5, Figure 23 shows the changes in the profit zones. There are two main differences that can be observed.

Observation 1: There is a one kink exactly at .

Observation 2: There is a second kink around

The explanation for Observation 1 is intuitive. Due to the addition of capacity constraint of , the E-tailer can no longer handle additional shipments when his own demand exceeds 5. As a result, he experiences a loss in profits. As such, a section of the area above prior to capacity constraint where the E-tailer benefits from service sharing is now not as beneficial for the E-tailer anymore as he will not be able to handle additional shipment from the seller.

For Observation 2, this is due to the difference in profits between the sharing mode and non-sharing mode. From the equations derived earlier, profits for the E-tailer are still increasing in the market potential, in both sharing and non-sharing modes, albeit at different rates. In the sharing mode, although he faces capacity constraint, but he still earns a commission from the seller. On the other hand, profits are decreasing in the TPLP service level, in both modes, also at different rates. As such, due to the interplay between these two variables and hence varying difference in profits in the sharing and non-sharing mode, the slope reverses as higher levels of and .

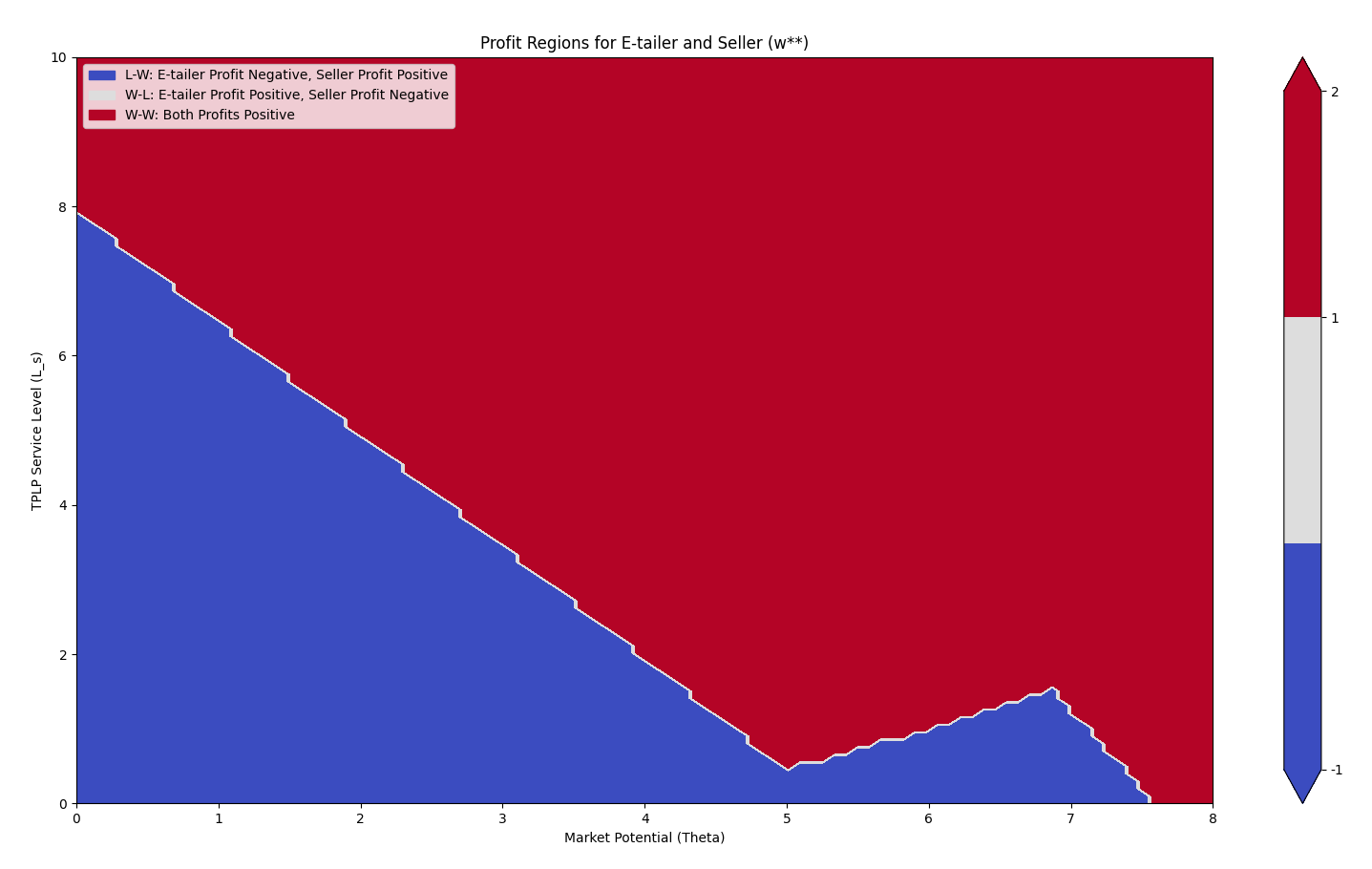


Figure 23: Profit zones for E-tailer and seller (

The addition of capacity constraint could draw new insights into the difference in behaviours of all three players at different levels of .

As mentioned earlier, if the players end up in any portion of the grey area in Figure 21, the E-tailer will be able to adjust to a lower value such that the seller would then be incentivized to engage in logistics sharing. Both players will then profit, albeit a lower amount for the E-tailer. A result from this is as long as the E-tailer profits, the seller will profit as well. This will result in an inevitable sharing of logistics. On the other hand, if the E-tailer does not profit, both the E-tailer and seller will never engage in logistics sharing. This result is essential in solving the Stackelberg Game in the next section.

### Game Theory Framework and Implementation

#### Game Theory Framework

To model the hierarchical interactions among the TPLP, E-tailer, and Seller, this study adopts a Stackelberg game framework. The Stackelberg model is appropriate for this setting due to its ability to capture leader-follower dynamics. This is a common feature in supply chains where upstream actors make decisions that downstream players must respond to. This is especially relevant in e-commerce logistics, where TPLPs often influence the operational decisions of sellers and platforms by setting logistics service prices and capacities (Joong‐Kun Cho et al., 2008).

The overall framework can be visualized in Figure 24 below. At the top of the decision hierarchy is the TPLP, which acts as the Stackelberg leader. It decides on two continuous action variables: the logistics service fee and service level ​. These decisions influence the attractiveness of logistics sharing for the downstream players. The TPLP’s profit depends on whether the E-tailer and seller engages in service sharing. If sharing occurs and the E-tailer exceeds its logistics capacity, the TPLP profits from handling the excess shipment. If not, the TPLP earns based on direct service provision to the seller.

A diagram of a product

Description automatically generated with medium confidence

Figure 24: Stackelberg framework

Below the TPLP, the E-tailer acts as the sub-leader in a nested Stackelberg game. The E-tailer evaluates whether to offer logistics sharing to the seller by considering the market potential , the TPLP’s service level ​, and the associated logistics fee . The E-tailer solves this decision problem anticipating the seller’s response. This decision is modelled as a binary action: either share (1) or not share (0). The corresponding profit is computed based on price decisions, commission rates, cost structures, and potential excess demand. As mentioned earlier, the E-tailer can also earn from commissions on logistics handled by the TPLP when it outsources excess demand.

Finally, the seller, as the Stackelberg follower, decides whether to accept the sharing offer from the E-tailer. The decision again occurs in a binary action space (accept or reject). This decision depends on the same set of variables as the E-tailer and affects the seller’s logistics cost and service quality, influencing its own demand and profit. The seller’s strategic response completes the Stackelberg sub-game.

The nested nature of this framework ensures that each decision level anticipates the optimal reaction of its successor. The TPLP’s reinforcement learning-driven decisions are modelled using PPO, allowing it to adaptively adjust and ​ based on market conditions . Meanwhile, the E-tailer and seller solve a static Stackelberg game via backward induction, optimizing their logistics strategy given the TPLP’s policy.

This structure aligns well with supply chain theory, where service providers (TPLPs) often possess first-mover advantages due to their control over pricing, capacity, and infrastructure (Joong‐Kun Cho et al., 2008). Hence, modelling the TPLP as the Stackelberg leader reflects realistic power dynamics in contemporary supply chains. Overall, this multi-tiered Stackelberg framework, coupled with reinforcement learning, enables a nuanced exploration of strategic service sharing decisions, capturing both the static incentives and the dynamic feedback loops that drive modern logistics cooperation.

The overall Stackelberg equilibrium is derived using backward induction, ensuring that the leader’s decisions anticipate the followers’ optimal responses. This process captures the interdependencies between TPLPs, E-tailers, and sellers, reflecting real-world supply chain hierarchies.

#### Game Theory Implementation

In this section, the interaction between the seller and the E-tailer is modeled as a Stackelberg game, and the solution to this subgame is computed through a simple binary decision rule based on profit differentials. The E-tailer acts as the leader, deciding whether to offer logistics sharing, and the seller acts as the follower, choosing whether to accept the offer. The solving process is implemented in the code shown in Figure 25 below. This code evaluates the profitability of sharing for both parties based on the different parameters.

import numpy as np

from Logistics\_Service\_Model import LogisticsServiceModel

def stackelberg\_game(L\_s, theta, f):

    =

    # Initialize the logistics service model

    model = LogisticsServiceModel(L\_s, theta, f)

    profit\_et\_no\_sharing = model.profit\_nosharing\_etailer()

    profit\_et\_sharing = model.profit\_sharing\_etailer(True)

    profit\_seller\_no\_sharing = model.profit\_nosharing\_seller()

    profit\_seller\_sharing = model.profit\_sharing\_seller(True)

    # Calculate profit differences directly in the loop

    profit\_diff\_et = profit\_et\_sharing - profit\_et\_no\_sharing

    profit\_diff\_seller = profit\_seller\_sharing - profit\_seller\_no\_sharing

    if profit\_diff\_et <0:

        return 0,0

    else:

        return 1,1

Figure 25: Stackelberg game function

The logic within *stackelberg\_game* makes use of the profit functions previously defined in *LogisticsServiceModel*, which compute the profits of both the E-tailer and the seller under two scenarios: with and without logistics sharing. The function begins by initializing the logistics market environment using the current values set by the TPLP. It then calls the four key profit functions, each returning the respective party’s profit under the given market condition. The profit differentials comparing before and after logistics sharing are then calculated directly. If the E-tailer's profit from sharing is negative, the game terminates with a (0, 0) outcome, meaning no sharing occurs. Otherwise, both parties proceed with sharing, returning (1, 1). This logic is tied to the fact mentioned previously where if the E-tailer benefits, sharing will occur. This binary outcome provides a clean, interpretable decision rule that can be repeatedly called by the TPLP’s reinforcement learning loop to evaluate downstream reactions.

### Reinforcement Learning implementation

To explore how the TPLP can strategically adjust its decisions in a dynamic coopetition setting, this section outlines the application of RL within the logistics service sharing framework. The aim is to simulate a learning process where the TPLP continuously refines its decisions in response to changing market conditions and the strategic reactions of other players. This methodology combines the principles of game theory with adaptive learning algorithms to enable realistic modelling of long-term strategy formation. The RL setup is structured into six key components: Environment Design, State Representation, Action Space, Reward Formulation, Policy Optimization, and Training configuration. The code for the entire implementation is in the Appendix below.

#### Environment Design

The RL environment simulates a real-world coopetition setting involving three key players in an e-commerce logistics network: the E-tailer, the Seller, and the TPLP. The environment is implemented using the PettingZoo framework (*PettingZoo Documentation*, n.d.), adopting an agent-environment cycle (AEC) where only one agent acts at a time. Figure 26 below shows how a typical AEC looks like with two players. In this example, Player 1 would make an action which causes a change to the environment in step 1. Player 2 then observes this new environment and makes an action causing a change to the same environment in step 2, and this cycle continues. This setup allows detailed control and modelling of turn-based interactions.

In the current configuration, the environment is restricted to one learning agent, the TPLP, while the E-tailer and seller behaviours are determined by the Stackelberg game as previously outlined. The TPLP learns to optimize its continuous decision variables: Logistics service level ( and logistics fee (), based on evolving market conditions and game-theoretic responses from the other two agents.

A diagram of a diagram

AI-generated content may be incorrect.

Figure 26: AEC

Each episode runs for a fixed number of 50 iterations, which provides a structured yet efficient window for decision-making and learning. This length is long enough to allow the agent to observe the consequences of its choices over a sequence of strategic interactions, yet short enough to enable rapid sampling across diverse market scenarios. It helps balance between learning depth (from within an episode) and breadth (across episodes. This episodic length ensures the TPLP can observe cumulative effects without requiring prohibitively long episodes.

#### State representation

The environment state is represented as a vector that captures all relevant market and strategic information for decision-making:

* : Market potential or demand level
* : Service level chosen by the TPLP
* : Logistics price set by the TPLP
* : E-tailer’s binary decision to offer sharing (0 = no, 1 = yes)
* : Whether logistics service sharing occurs (0 = no, 1 = yes)

The TPLP receives a partial observation of the full state, specifically the market potential and whether sharing was accepted. Based on these observation state, the reinforcement learning agent will then assign the optimal policy for each observation.

#### Action Space

The TPLP's action space consists of two continuous variables:

* : Chosen within a range of [0, 10]. This maps to a 0% to 100% normalized logistics service scale. It allows the TPLP to control how reliable or efficient its logistics offering is. A higher service level is generally more attractive to both the E-tailer and seller but comes at a cost.
* Selected from [0.5, 3.0]. This range is grounded in the original paper, where the logistics price was set as a fixed value of 1.0. By expanding this to a continuous interval, the model allows the TPLP to learn flexible pricing strategies instead of relying on a static assumption. The lower bound reflects minimum pricing to cover baseline costs, while the upper bound provides more opportunities for the TPLP to find a balance between increased revenue but increased costs at the same time.

These two variables interact in a complex way to affect the strategic responses of the other agents in the environment. As the leader in a Stackelberg game, the TPLP sets both values, anticipating the equilibrium behaviour of the e-tailer and seller. The continuous nature of the space ensures that the TPLP can fine-tune its strategy, rather than being limited to coarse or binary decisions. This design supports nuanced learning and allows exploration of a wide variety of strategic options.

#### Reward function

Rewards for the TPLP are based on profit, derived from whether the logistics sharing agreement is activated. Sharing only occurs if both the E-tailer and seller benefit, which is computed by checking if their respective profit differences are non-negative compared to the non-sharing scenario. The reward signal is cumulative across an episode, meaning the agent is incentivized to not only make profitable decisions at a single point in time but to consider how its actions affect long-term outcomes. This helps the TPLP agent learn policies that account for delayed effects—such as increasing service levels to induce cooperation in future rounds or adjusting prices gradually to retain participation. The cumulative reward structure aligns with real-world strategic planning, where benefits from partnerships or pricing decisions may emerge over multiple periods rather than immediately.

Once sharing is confirmed, the profit for the TPLP is computed using the *LogisticsServiceModel* class as described earlier. If sharing is not beneficial to either of the other two agents, the TPLP receives a profit based on standard logistics services. This conditional reward structure encourages the TPLP to learn pricing and service combinations that induce voluntary cooperation among the parties.

#### Policy Optimisation

The TPLP's policy is optimized using Proximal Policy Optimization (PPO), a stable and efficient policy-gradient reinforcement learning algorithm. PPO is well-suited for continuous action spaces and ensures that updates do not deviate excessively from the current policy due to its clipping mechanism.

The objective is to maximize the expected discounted reward over time while ensuring stable policy updates. PPO enables learning robust policies that generalize well across episodes with varying market conditions and responses from the e-tailer and seller. Figure 27 below shows a standard PPO architecture. In a training iteration, PPO performs three major steps:

1. Sampling a set of episodes or episode fragments

2. Converting these into a train batch and updating the model using a clipped objective and multiple stochastic gradient descent passes over this batch

3. Syncing the weights from the Learners (responsible for updating model parameters) back to the EnvRunners (parallel workers who sample actions from the current policy, receive rewards and pass them to the Learners).

A diagram of a program

AI-generated content may be incorrect.

Figure 27: PPO Architecture

The training setup uses Ray RLlib (*Algorithms — Ray 2.44.1*, n.d.), which handles the policy updates, sampling, and rollout collection efficiently across multiple parallel workers.

#### Training Configuration

The training configuration uses Ray RLlib's PPO implementation with the following key recommended settings:

* **Learning rate**: Very small () to stabilize learning in a highly sensitive economic model. This low value ensures gradual and cautious updates to the policy network, which is especially important in economic environments where slight changes in pricing or service levels can lead to large shifts in strategic outcomes. It prevents overshooting and helps the agent converge to stable and reliable behavior over time.
* **Discount factor ()**: 0.99 to value long-term gains. This setting ensures that the agent does not just focus on immediate profits but instead considers the cumulative effect of its actions over the entire episode. A high discount factor like 0.99 is suitable for long-horizon decision-making, allowing the TPLP to evaluate strategies that yield benefits over time.
* **Generalised Advantage Estimator (GAE) ()**: 0.9 for variance reduction in advantage estimation. GAE helps stabilise the learning process by providing a smoothed estimate of how advantageous a particular action was. A λ value of 0.9 balances the trade-off between bias and variance, making the learning process both reliable and sample efficient.
* **Entropy coefficient**: 0.1 to encourage policy exploration. This parameter adds a bonus to the policy loss based on the uncertainty of action selection, promoting diverse behaviour and preventing the policy from converging too quickly to a suboptimal strategy.
* **Value function loss coefficient**: 0.25 to balance policy and value learning. This coefficient determines the importance of the value function’s prediction error in the total loss function. A value function helps the agent estimate the long-term return (expected future rewards) from any given state.
* **Stochastic gradient descent minibatch size**: 64 with 10 epochs per training batch. This means that for every training batch collected (made up of experiences across multiple episodes and environment rollouts), the data is divided into smaller chunks of 64 samples. These minibatches are then cycled through 10 epochs (10 times), allowing the agent to repeatedly update its parameters with the same data.
* **Number of iterations per episode:** 50.

Additionally, the training runs are monitored with custom callbacks that log episode rewards, and a custom stopper that halts training when the reward variance stabilizes below a threshold, preventing unnecessary computation once the agent has converged. For each training loop, that are a set of rules which govern the stopping condition, allowing sufficient time to explore the effect of the TPLP’s pricing and service decisions across multiple simulated interactions, yet also taking into consideration the duration of each training episode. The *PercentageVarianceStopper* class governs the stopping conditions are as follows:

* **Patience (patience)**: 5  
  This means the stopper considers the most recent 5 episode rewards when evaluating whether training should stop.
* **Percentage Threshold (percentage\_threshold)**: 0.01  
  This represents a 1% standard deviation. If the standard deviation of the 5 most recent rewards is less than 1% of their mean, the stopper concludes that the rewards have stabilized and stops training. This will help to reduce the overall time taken for training if the results have already stabilised.
* **Maximum Timesteps (max\_timesteps)**: 5,000,000  
  Regardless of reward variance, if the training reachesa total of 5,000,000 timesteps, it will stop to avoid excessive computation.

In terms of the conditions for each training loop, the agent is trained for different levels of , ranging from to . This is so that the agent will be able to identify the best policy taken for a given specific market condition (low vs high market potential).

Together, the integration of game theory and reinforcement learning brings both structure and adaptability to the model. The game-theoretic layer between the E-tailer and seller ensures that downstream decisions are economically rational and responsive, while the reinforcement learning layer equips the TPLP with the ability to learn optimal upstream strategies over time. This combined framework mirrors real-world logistics dynamics, where strategic interactions and adaptive learning coexist. The results from this model, both in terms of learning performance and emergent behaviours, will be analysed and discussed in the following section.

# Results and Discussion

This section explores the strategic outcomes derived from the hybrid Stackelberg–reinforcement learning framework designed to analyze logistics service sharing in e-commerce supply chains. First, the sharing zones generated from the Stackelberg game are visualized across varying market potentials and service levels, revealing the conditions under which logistics cooperation between E-tailers and sellers is likely to emerge. The analysis then transitions to the reinforcement learning outcomes, highlighting how the TPLP dynamically selects service levels and pricing strategies ( and *f*) in response to different market environments (). Further, the impact of capacity constraints on the E-tailer’s logistics decisions is examined. Lastly, and most critically, this section contextualizes the findings within real-world logistics settings, offering practical insights into how platform-seller-TPLP interactions can be strategically managed to enhance supply chain resilience and coordination in competitive e-commerce environments.

## Sharing Zone Analysis from Stackelberg Game

The 3D region plot below in Figure 28 illustrates the decision outcomes of the Stackelberg game model with respect to logistics service sharing between the E-tailer and the seller. The plot is parameterized over three key dimensions: θ (market potential) on the x-axis, Lₛ (logistics service level offered by the TPLP) on the y-axis, and (logistics price set by the TPLP) on the z-axis. Each data point represents whether a sharing agreement (green, labelled as (1,1)) or no-sharing decision (red, labelled as (0,0)) occurs at the corresponding configuration of θ, Lₛ, and f.

Below are some of the key observations:

1. Dominant Sharing Region:
   * The green region occupies a significant portion of the 3D space, especially at higher values of θ and Lₛ, indicating that as market potential increases and the logistics service level offered by the TPLP improves, the likelihood of both parties engaging in logistics service sharing increases.
   * This aligns with the strategic logic that in high-demand, high-service-quality environments, collaboration is more likely as the E-tailer will benefit more, leading to sharing and win-win outcomes.
2. No-Sharing Zone Characteristics:
   * The red region is concentrated in the lower range of θ (4 to ~6) and Lₛ (0 to ~4), suggesting that when both the market size and third-party logistics service quality are low, the incentive for sharing diminishes.
   * In such scenarios, the E-tailer might prefer to avoid sharing to maintain a competitive edge, while the seller may not perceive sufficient value to justify the logistics cost.
3. Influence of (Price Factor):
   * The z-axis () introduces a third layer of strategic complexity. In general, higher values (TPLP charging more for logistics) correspond to a decrease in the sharing region.
   * This is because a higher indicates that the TPLP’s logistics service is less cost effective. As a result, for the E-tailer, a higher implies a stronger negative effect of logistics service sharing (lost competitive edge in logistics service). Hence, the E-tailer has less incentive to share his logistics service system.
4. Boundary Shift:
   * There is a clear nonlinear transition between the no-sharing and sharing regions, indicating threshold behaviour where a small increase in or can suddenly shift the system from non-cooperation to cooperation.

The plot reveals that service quality improvements and market effects are key levers to unlock cooperative behaviour in competitive e-commerce settings. Moreover, it highlights the critical role of pricing () by TPLPs as a tool to indirectly influence cooperation.

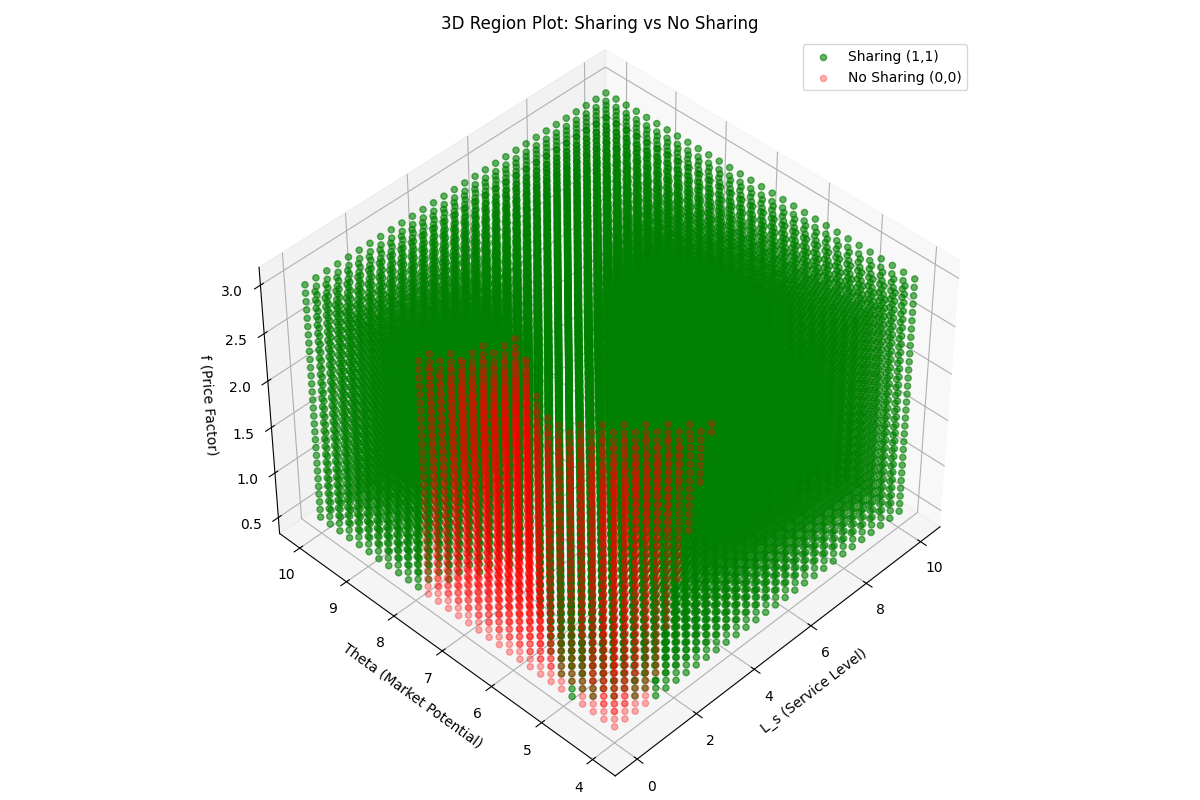


Figure 28: 3D Plot of sharing and no-sharing regions

## TPLP’s Adaptive Strategy under no capacity constraints

To extract and test the trained policies in the context of the TPLP, a Python-based simulation framework with a user-interactive environment is utilised (Appendix). Specifically, the policy corresponding to the TPLP agent at a specific was retrieved. Subsequently, this trained policy was then embedded into a Pygame-based graphical interface as show in Figure 29 below, where users could input service-level () and logistics pricing () values. These user decisions were compared against the trained policies. Both the human-in-the-loop and autonomous agent interactions were simulated in parallel to collect reward and profit data across agents, enabling a comparative analysis of the learned policy’s decision-making quality and validity.

This dual setup allowed us to assess performance of the learned. Through this comparative structure, the simulation validated that the policy could approximate optimal strategies in scenarios previously described in the strategic model of logistics service sharing.

A screenshot of a computer

AI-generated content may be incorrect.

Figure 29: TPLP Simulation Game

Table 2 below presents the reinforcement learning outcomes of the TPLP under different values of market potential () when the E-tailer faces no capacity constraint, highlighting the corresponding logistics service level (), logistics price (), realized profit, and the presence of logistics service sharing.

Table 2: TPLP's actions under varying market potential with capacity constraint

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | 0.13 | 0.05 | - | - | - |
|  | 0.81 | 1.47 | - | - | - |
| Profit | 0.50 | 0.72 | 0 | 0 | 0 |
| Logistics service sharing (Y/N) | N | N | N | N | N |
| Training curve |  |  | - | - | - |

Below are some of the key observations and explanations:

* **Low and no logistics service sharing:**

The TPLP consistently selects low and across all tested levels of across the entire range of θ values and logistics service sharing does not occur. When the E-tailer is unconstrained, it can fulfil all demand internally. This removes the TPLP’s leverage and reduces the attractiveness of outsourcing logistics to the TPLP. Moreover, in this scenario, logistics service sharing results in zero profit for the TPLP. As such, the optimal strategy is to deter sharing altogether. The TPLP achieves this by deliberately offering an uncompetitive logistics bundle: it lowers both and to the point that service sharing is unappealing to both the E-tailer and seller. This combination ensures that the E-tailer and seller opt to avoid sharing, preserving the status quo. This behaviour reflects a strategic withdrawal rather than engagement. Since participation yields no upside, the TPLP efficiently conserves resources and avoids incurring logistics costs that cannot be offset through commission or margin.

* **No results for high values of (6,7,8)**

No results are reported for higher values of market potential in the no-capacity-constraint scenario because the TPLP is effectively unable to influence the outcome. Once market potential becomes sufficiently large, the profitability of logistics service sharing between the E-tailer and seller increases significantly regardless of the TPLP’s chosen or .

Under these conditions, both the E-tailer and seller benefit from cooperation, as the increased market demand leads to higher joint profits in the sharing mode. This makes sharing the default strategy. As a result, the TPLP loses strategic leverage. Even if it sets extremely low values for or high values for , it cannot outcompete the internal logistics solution offered by the E-tailer. This renders its actions effectively meaningless, and the outcome of sharing becomes invariant to the TPLP's policy. From the RL agent's perspective, the environment becomes uninformative, leading to no policy evolution and thus no meaningful results.

## TPLP’s Adaptive Strategy under capacity constraints

When capacity constraint is introduced on the E-tailer’s in-house logistics service, the TPLP adapts its strategy dynamically in response to varying levels of . The actions taken by the TPLP varies under different values as compared to the unconstrained condition, highlighting how capacity constraints affect the decisions made by the TPLP. Table 3 below presents the reinforcement learning outcomes of the TPLP the E-tailer faces capacity constraint, highlighting the corresponding logistics service level (), logistics price (), realized profit, and the presence of logistics service sharing.

Table 3: TPLP's actions under varying market potential with capacity constraint (

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | 7.81 | 7.30 |
|  | 1.78 | 1.79 |
| Profit | 3.67 | 5.15 |
| Logistics service sharing (Y/N) | Y | Y |
| Training curve | A graph with lines and numbers  AI-generated content may be incorrect. | A graph of a stock market  AI-generated content may be incorrect. |

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 5.27 | 2.95 | 2.82 |
| 2.79 | 2.14 | 1.96 |
| 7.76 | 12.42 | 12.48 |
| Y | Y | Y |
|  |  |  |

Below are some of the key observations and explanations:

* **Service Level ()**  
  The TPLP's chosen service level consistently decreases as increases. At lower market potential, the TPLP maintains a high service level to encourage participation in logistics sharing, especially when the E-tailer still retains sufficient capacity. As increases and the E-tailer's fulfillment capacity becomes increasingly constrained, the necessity of logistics sharing reduces the TPLP’s need to offer high service levels. Since cooperation becomes guaranteed, the TPLP begins to lower its service effort to reduce operational costs effectively maximizing profit while minimizing resource deployment.
* **Logistics Price ()**  
  The logistics price remains relatively steady at lower levels of (4.5-5), followed by a sharp increase at , and then gradually tapers off at higher . As the market potential increases, the TPLP's pricing behaviour shows a strategic shift from cost-sharing to value-capture. When θ is at or below 5, the point at which the E-tailer hits its capacity constraint, the E-tailer is still able to manage demand independently. During this phase, the TPLP maintains relatively low and high to encourage the E-tailer to begin sharing logistics services. However, at = 6, the situation changes drastically. The E-tailer is no longer capable of meeting rising market demand without external support and cooperation with the TPLP becomes essential. The TPLP capitalizes this by sharply increasing its price from 1.79 to 2.79, This marks a clear transition to value extraction, where the TPLP maximizes its own profit from the E-tailer's lack of alternatives. Beyond = 6, at = 7 and 8, the TPLP begins to taper its price back down. This is because as increases, the E-tailer becomes more sensitive to cost. To balance this, the TPLP lowers , minimizing its own costs while maintaining cooperation.
* **Profit**  
  TPLP’s profit rises consistently across increasing levels of θ. This is intuitive as higher market potential will increase demand across the board, resulting in higher profits for all parties.
* **Logistics Service Sharing**  
  Sharing mode is enabled across all levels of θ. The persistence of sharing behaviour across all levels suggests that due to the E-tailer’s capacity constraint, logistics sharing is always profitable for the TPLP as he can extract value from the partnership.

## Discussion of results

The differences in observed behaviour between these scenarios and particularly the divergence in chosen logistics fee and service level , provides a set of real-world insights for logistics managers and strategists operating within competitive and cooperate e-commerce ecosystems.

**1. Unconstrained Environment: TPLP as a disabler for cooperation**

In this environment, the E-tailer has sufficient internal logistics capacity to fulfil both its own and the seller’s orders, eliminating the dependency that forms the basis of the TPLP’s value proposition in the constrained case. The TPLP responds by adopting a strategic withdrawal posture. Across all market potential values tested ( = 4.5 to 6), the TPLP consistently selects low values for both and , effectively offering uncompetitive logistics bundles and essentially “forcing” the no-sharing condition between the seller and the E-tailer. This behaviour serves a clear purpose which is to dissuade participation in logistics service sharing. Importantly, in the unconstrained setting, logistics service sharing yields zero profit for the TPLP and the only path to profitability for the TPLP would be through full outsourcing by the seller.

**2. Capacity-Constrained Environment: TPLP as a strategic partner**

Secondly, in the capacity-constrained scenario, the E-tailer has a limited ability to fulfil logistics demand beyond a certain threshold. This capacity ceiling fixed at in this model presents the TPLP with an opportunity to transition to a potential partner for the E-tailer.

Under low market potential ( < 5), the E-tailer remains self-sufficient himself. In this regime, the TPLP plays a supportive role by offering relatively high values and moderate pricing, incentivising logistics sharing and extracting some of the profits due to excess volume from the E-tailer. However, as market potential reaches and surpasses the capacity threshold (above ), the E-tailer can no longer meet his own demand alone. Recognizing the E-tailer’s dependency, the TPLP raises its logistics fee significantly. This pricing spike helps to extract surplus for the TPLP. As the market potential continues to rise ( = 7, 8), the TPLP no longer needs to maintain elevated service levels and logistics fee. The results show a consistent reduction in and , indicating a strategic pivot toward cost minimization instead. Since cooperation is now guaranteed, sharing continues across all levels, the TPLP begins to reduce operational intensity while still profiting from excess volume.

# Conclusion

This study investigates the dynamic interplay between E-tailers, sellers, and TPLPs in e-commerce supply chains through a novel hybrid framework that integrates Stackelberg game theory with PPO-based reinforcement learning. By extending the foundational model proposed by Qin et al. (2020), which primarily analysed logistics service sharing between E-tailers and sellers, this work explicitly incorporates the strategic role of the TPLP which has been overlooked.

To address this gap, game theory was used to model the hierarchical structure of decision-making**,** capturing the sequential interactions between the TPLP (as leader), E-tailer (sub-leader), and seller (follower). This Stackelberg game structure allowed for realistic modelling of how logistics decisions are influenced upstream and resolved downstream. Reinforcement learning, specifically PPO, was introduced to enable the TPLP to learn adaptive strategies in a dynamic environment where market conditions () and downstream responses are continuously evolving. This combination empowered the TPLP to discover optimal combinations of logistics fee () and service level () that either incentivize or deter logistics service sharing depending on market constraints.

Through this approach, the study was able to:

1. Identify value extraction windows for TPLPs when E-tailers are capacity-constrained.
2. Demonstrate strategic withdrawal behaviour under unconstrained conditions where sharing yields no profit.

This work contributes not only to the theoretical understanding of coopetition in e-commerce supply chains but also to the practical development of adaptive logistics strategies by TPLPs in increasingly complex e-commerce ecosystems.

## Real-World Implications and Strategic Recommendations

From the perspective of real TPLPs operating in e-commerce networks, several actionable takeaways emerge from these findings:

* **Monitor Capacity Signals**: TPLPs should track volume growth and delivery backlogs within client platforms. These metrics indicate when capacity constraints may be approaching, opening opportunities for deeper integration and value capture.
* **Time Market Entry Strategically**: Early in a platform’s growth (when is still low), TPLPs should have attractive and combinations which incentivises logistics sharing between the E-tailer and seller. As the E-tailer matures and becomes constrained, they can shift to monetization instead by increasing .
* **Avoid Resource Drain in Mature, Unconstrained Markets**: In ecosystems where platforms have robust in-house logistics (huge capacity), TPLPs should consider limiting investment. Instead, they might focus on niche services or shift toward B2B fulfilment and warehousing.
* **Prepare for Scale Through Efficiency**: Once sharing becomes guaranteed and market potential increases, cost control rather than service expansion drives long-term profitability. Learning when to reduce while preserving sharing incentives is essential.
* **Understand Downstream Incentive Compatibility**: The Stackelberg framework demonstrates that TPLPs must anticipate not just direct profit, but how their pricing structures influence the downstream E-tailer-seller sharing dynamic. Strategic foresight here is vital.

## Limitations and future works

While the proposed hybrid Stackelberg–reinforcement learning framework offers valuable insights into the dynamic strategy of TPLPs in e-commerce supply chains, several limitations are acknowledged. These constraints, both in modelling assumptions and implementation scope, highlight opportunities for future work that could significantly enhance the accuracy, applicability, and robustness of the findings.

**1. Simplified Profit Function for the TPLP**

**Limitation:**  
The current model adopts a quadratic cost-based profit function for the TPLP, which increases with both and . While this captures key economic trade-offs (e.g., diminishing returns and rising marginal costs), the formulation is not empirically derived and may not fully reflect operational realities.

**Future Work:**  
To improve precision, future studies could:

* Calibrate the profit function using actual cost breakdowns from third-party logistics providers, distinguishing between fixed, variable, and step-function costs.
* Incorporate logistics-specific cost drivers such as fuel prices, labor rates, inventory turnover, and regional service densities.
* Extend the model to include multi-period contracts and bulk pricing discounts commonly observed in long-term TPLP partnerships.

**2. Lack of Real-World Validation for Capacity Constraints**

**Limitation:**  
In this study, the E-tailer’s logistics capacity was artificially capped (= 5) to create a constrained environment. While this assumption enables the exploration of tipping-point dynamics in TPLP leverage, it also lacks empirical justification or industry benchmarking.

**Future Work:**  
To address this, future iterations should:

* Use real capacity data (e.g., daily parcel throughput, warehouse capacity, or delivery slots) from logistics or platform case studies.
* Incorporate dynamically evolving capacities, such as scenarios where the E-tailer scales its logistics infrastructure in response to demand or contracts temporary TPLPs.
* Model stochastic or seasonal demand profiles to capture real-world variability in logistics load.

This would improve the realism of constraint-driven transitions and provide stronger validation of the TPLP’s opportunity window for value capture.

**3. Absence of Real-World Case Study or Benchmarking Data**

**Limitation:**  
Although the simulation reveals meaningful behavioural trends, such as the TPLP’s withdrawal under unconstrained conditions or pricing surges at capacity thresholds, these are based solely on simulation outcomes without reference to observable firm behaviour or market data.

**Future Work:**  
To strengthen external validity, future research could:

* Compare results to case studies from logistics partnerships involving Amazon, JD Logistics, Cainiao, etc.
* Collect and analyze public pricing, SLA, and integration announcements to observe how TPLPs adjust service offerings post-onboarding.

This would provide critical grounding for the simulated agent behaviour, helping to distinguish between theoretical optimality and practical feasibility.

**4. Exploration of Alternative Reinforcement Learning Algorithms**

**Limitation:**  
The current implementation relies solely on Proximal Policy Optimization (PPO), chosen for its stability in continuous action spaces.

**Future Work:**  
Future versions of the model could compare with other kinds of RL techniques for example:

* **Soft Actor-Critic (SAC)**: An off-policy actor-critic deep RL algorithm for improved exploration.
* **Multi-Agent Deep Deterministic Policy Gradient (MADDPG)**: To explicitly model and coordinate multiple learning agents (e.g., if sellers or E-tailers become partially learning agents).
* **Hierarchical RL (HRL)**: To capture the multi-level decision-making (e.g., high-level policy to induce sharing, low-level to price competitively).

Benchmarking these alternatives against PPO in terms of convergence speed, stability, and learned policy quality could yield important methodological insights.

**5. Simplification of Competitive Environment**

**Limitation:**  
The model currently assumes a single TPLP, single E-tailer, and a single seller. While this simplifies analysis, it omits important real-world dynamics such as:

* Competition among multiple TPLPs.
* Platform-to-platform competition (e.g., Shopee vs. Lazada).
* Seller heterogeneity in terms of size, delivery region, and fulfillment preference.

**Future Work:**  
An expanded simulation could:

* Introduce agent heterogeneity to capture diversity in logistics preferences.
* Model oligopolistic TPLP markets, where price wars and service differentiation matter.
* Extend the Stackelberg game to a multi-leader–multi-follower structure with more realistic hierarchical dynamics.

This would enhance the relevance of the results in more complex e-commerce ecosystems.

# References

*Algorithms—Ray 2.44.1*. (n.d.). Retrieved 15 April 2025, from https://docs.ray.io/en/latest/rllib/rllib-algorithms.html

Bernstein, F., & Federgruen, A. (2004). A General Equilibrium Model for Industries with Price and Service Competition. *Operations Research*. https://doi.org/10.1287/opre.1040.0149

Bijulal, D., Venkateswaran, J., & Hemachandra, N. (2011). Service levels, system cost and stability of production–inventory control systems. *International Journal of Production Research*. https://www.tandfonline.com/doi/abs/10.1080/00207543.2010.538744

Chen, T., Zhang, X., & Jia, F. (2024). Platform or Third‐Party Service Provider? An Analysis of Logistics Mode Selection Strategy for Manufacturers in E‐Commerce Supply Chains. *Managerial and Decision Economics*, *46*. https://doi.org/10.1002/mde.4419

Chen, X., Luo, Z., & Wang, X. (2019). Compete or cooperate: Intensity, dynamics, and optimal strategies. *Omega*, *86*, 76–86. https://doi.org/10.1016/j.omega.2018.07.002

Chopra, S., & Meindl, P. (2002). *Supply Chain Management. Strategy, Planning & Operation*. https://doi.org/10.1007/978-3-8349-9320-5\_22

Coyle, J. J., Langley, C. J., Novack, R. A., & Gibson, B. J. (2017). *Supply chain management: A logistics perspective* (10 e). Cengage Learning.

Ding, Y., Gao, X., Huang, C., Shu, J., & Yang, D. (2018). Service competition in an online duopoly market. *Omega*, *77*, 58–72. https://doi.org/10.1016/j.omega.2017.05.007

Feng, L., Xing, D., Zhang, J., & Pan, G. (2023). *FP3O: Enabling Proximal Policy Optimization in Multi-Agent Cooperation with Parameter-Sharing Versatility* (arXiv:2310.05053). arXiv. https://doi.org/10.48550/arXiv.2310.05053

Guo, L., & Wu, X. (2018). Capacity Sharing Between Competitors. *Management Science*, *64*(8), 3554–3573. https://doi.org/10.1287/mnsc.2017.2796

Hoek, R. (2001). E-Supply Chains—Virtually Non-Existing. *Supply Chain Management: An International Journal*, *6*, 21–28. https://doi.org/10.1108/13598540110694653

Joong‐Kun Cho, J., Ozment, J., & Sink, H. (2008). Logistics capability, logistics outsourcing and firm performance in an e‐commerce market. *International Journal of Physical Distribution & Logistics Management*, *38*(5), 336–359. https://doi.org/10.1108/09600030810882825

Khooban, Z., Mutlu, N., & Kok, T. D. (2025). Logistics service sharing in cross-border e-commerce. *International Journal of Production Economics*, *279*, 109460. https://doi.org/10.1016/j.ijpe.2024.109460

Kim, B., Kim, J. G., & Lee, S. (2024). A multi-agent reinforcement learning model for inventory transshipments under supply chain disruption. *IISE Transactions*. https://www.tandfonline.com/doi/abs/10.1080/24725854.2023.2217248

Mentzer, J., Dewitt, W., Keebler, J., Min, S., Nix, N., Smith, C., & Zacharia, Z. (2001). Defining Supply Chain Management. *Journal of Business Logistics*, *22*. https://doi.org/10.1002/j.2158-1592.2001.tb00001.x

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). *Reinforcement Learning for Solving the Vehicle Routing Problem* (arXiv:1802.04240). arXiv. https://doi.org/10.48550/arXiv.1802.04240

*PettingZoo Documentation*. (n.d.). Retrieved 10 April 2025, from https://pettingzoo.farama.org/tutorials/rllib/index.html

Qin, X., Liu, Z., & Tian, L. (2020). The strategic analysis of logistics service sharing in an e-commerce platform. *Omega*, *92*, 102153. https://doi.org/10.1016/j.omega.2019.102153

Qin, Y. (2012). A stackelberg-game model in a two-stage supply chain. *Systems Engineering Procedia*, *3*, 268–274. https://doi.org/10.1016/j.sepro.2011.11.029

Rzeczycki, A. (2022). Supply chain decision making with use of game theory. *Procedia Computer Science*, *207*, 3988–3997. https://doi.org/10.1016/j.procs.2022.09.461

Simchi-levi, D., Kaminsky, P., & Simchi-Levi, E. (2003). *Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies*.

Tang, C., & Veelenturf, L. (2019). The strategic role of logistics in the industry 4.0 era. *Transportation Research Part E: Logistics and Transportation Review*, *129*, 1–11. https://doi.org/10.1016/j.tre.2019.06.004

Tsay, A. A., & Agrawal, N. (2000). Channel Dynamics Under Price and Service Competition. *Manufacturing & Service Operations Management*, *2*(4), 372–391. https://doi.org/10.1287/msom.2.4.372.12342

Ülkü, M. A., & Bookbinder, J. H. (2012). Optimal quoting of delivery time by a third party logistics provider: The impact of shipment consolidation and temporal pricing schemes. *European Journal of Operational Research*, *221*(1), 110–117. https://doi.org/10.1016/j.ejor.2012.03.021

Vidani, J. (2024). *E-Commerce Supply Chain Efficiency: A Case Study of Amazon E-Commerce Company* (SSRN Scholarly Paper 4849852). Social Science Research Network. https://doi.org/10.2139/ssrn.4849852

Von Stackelberg, H. (2011). *Market Structure and Equilibrium*. Springer. https://doi.org/10.1007/978-3-642-12586-7

Wang, P., Du, S., Hu, L., & Tang, W. (2022). Logistics choices in a platform supply chain: A co-opetitive perspective. *Journal of the Operational Research Society*, *74*, 1–19. https://doi.org/10.1080/01605682.2021.2023675

Yan, Y., Chow, A. H. F., Ho, C. P., Kuo, Y.-H., Wu, Q., & Ying, C. (2022). Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, *162*, 102712. https://doi.org/10.1016/j.tre.2022.102712

Yin, C., & Han, J. (2021). Dynamic Pricing Model of E-Commerce Platforms Based on Deep Reinforcement Learning. *Computer Modeling in Engineering & Sciences*, *127*(1), 291–307. https://doi.org/10.32604/cmes.2021.014347

Zennaro, I., Finco, S., Calzavara, M., & Persona, A. (2022). Implementing E-Commerce from Logistic Perspective: Literature Review and Methodological Framework. *Sustainability*, *14*(2), Article 2. https://doi.org/10.3390/su14020911

Zhang, J., Nault, B. R., & Tu, Y. (2015). A dynamic pricing strategy for a 3PL provider with heterogeneous customers. *International Journal of Production Economics*, *169*, 31–43. https://doi.org/10.1016/j.ijpe.2015.07.017

Zhao, S., & Li, W. (2023). Game-theoretic analysis of a two-stage dual-channel supply chain coordination in the presence of market segmentation and price discounts. *Electronic Commerce Research and Applications*, *57*, 101222. https://doi.org/10.1016/j.elerap.2022.101222

Zhuo, W., Peng, J., & Wang, J. (2024). Game theoretical analysis of incumbent platform investment and the supplier entry strategies in an e-supply chain. *International Journal of Production Economics*, *273*, 109234. https://doi.org/10.1016/j.ijpe.2024.109234

# Appendix

### Code for Figure 21,17,18

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

from Logistics\_Service\_Model import LogisticsServiceModel

def plot\_profit\_regions(ax,ww,f):

    theta\_values = np.linspace(0, 8, 100)  # Define a range of market potential (theta)

    L\_s\_values = np.linspace(0, 10, 100)    # Define a range of service levels for TPLP (L\_s)

    # Create meshgrid to vectorize the loop

    theta\_grid, L\_s\_grid = np.meshgrid(theta\_values, L\_s\_values)

    # Initialize arrays to hold the profit differences directly in the loop

    profit\_diff\_et = np.zeros\_like(theta\_grid)  # Profit difference for E-tailer

    profit\_diff\_seller = np.zeros\_like(theta\_grid)  # Profit difference for seller

    # Flatten the grids to iterate

    theta\_flat = theta\_grid.ravel()

    L\_s\_flat = L\_s\_grid.ravel()

    # Loop over all combinations of L\_s (seller service level) and theta (market potential)

    for idx in range(len(theta\_flat)):

        theta = theta\_flat[idx]

        L\_s = L\_s\_flat[idx]

        model = LogisticsServiceModel(L\_s, theta,f)

        # Calculate profits for no-sharing and sharing

        profit\_et\_no\_sharing = model.profit\_nosharing\_etailer()

        profit\_et\_sharing = model.profit\_sharing\_etailer(ww)

        profit\_seller\_no\_sharing = model.profit\_nosharing\_seller()

        profit\_seller\_sharing = model.profit\_sharing\_seller(ww)

        # Calculate profit differences directly in the loop

        profit\_diff\_et.flat[idx] = profit\_et\_sharing - profit\_et\_no\_sharing

        profit\_diff\_seller.flat[idx] = profit\_seller\_sharing - profit\_seller\_no\_sharing

    # Reshape the arrays back to grid shape

    profit\_diff\_et = profit\_diff\_et.reshape(theta\_grid.shape)

    profit\_diff\_seller = profit\_diff\_seller.reshape(theta\_grid.shape)

    # Initialize region array (W-W = 2, W-L = 1, L-W = -1)

    region = np.zeros\_like(profit\_diff\_et)

    # Set regions based on conditions

    region[(profit\_diff\_et >= 0) & (profit\_diff\_seller >=0)] = 2  # W-W

    region[(profit\_diff\_et > 0) & (profit\_diff\_seller < 0)] = 1  # W-L

    region[(profit\_diff\_et < 0) & (profit\_diff\_seller > 0)] = -1  # L-W

    # Plot the region map

    cmap = plt.get\_cmap('coolwarm', 3)  # Using a colormap with 3 discrete levels

    # Create the contour plot for regions

    c = ax.contourf(theta\_grid, L\_s\_grid, region, cmap=cmap, levels=[-1, 0, 1, 2], extend='both')

    title = "Profit Regions for E-tailer and Seller (w\*)"

    if ww == False:

        title = "Profit Regions for E-tailer and Seller (w\*\*)"

    # Set labels and titles

    ax.set\_title(title)

    ax.set\_xlabel("Market Potential (Theta)")

    ax.set\_ylabel("TPLP Service Level (L\_s)")

    legend\_labels = {

        2: 'W-W: Both Profits Positive',

        1: 'W-L: E-tailer Profit Positive, Seller Profit Negative',

        -1: 'L-W: E-tailer Profit Negative, Seller Profit Positive'

    }

    # Create custom patches for the legend

    handles = [

        mpatches.Patch(color=cmap(0), label=legend\_labels[-1]),  # L-W

        mpatches.Patch(color=cmap(1), label=legend\_labels[1]),

        mpatches.Patch(color=cmap(2), label=legend\_labels[2])    # W-W

    ]

    # Add a legend to the plot

    ax.legend(handles=handles, loc='upper left')

    # Add a color bar

    plt.colorbar(c, ax=ax, ticks=[-1, 1, 2], format='%d')

    # print(model.calc\_w(ww))

    # print(profit\_diff\_seller.min())

fig,axes = plt.subplots(1,1)

if \_\_name\_\_== '\_\_main\_\_':

    plot\_profit\_regions(axes,False,1)

    plt.show()

### Code for Figure 22

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

from Logistics\_Service\_Model import LogisticsServiceModel

from StackelBerg import stackelberg\_game

# Define parameter ranges

theta\_vals = np.linspace(4, 10, 30)

L\_s\_vals = np.linspace(0, 10, 30)

f\_vals = np.linspace(0.5, 3, 30)

# Create meshgrid

Theta, L\_s, F = np.meshgrid(theta\_vals, L\_s\_vals, f\_vals, indexing='ij')

# Prepare a mask for sharing outcome

Sharing = np.zeros(Theta.shape)

# Evaluate stackelberg\_game for each combination

for i in range(Theta.shape[0]):

    for j in range(Theta.shape[1]):

        for k in range(Theta.shape[2]):

            tplp, seller = stackelberg\_game(L\_s[i, j, k], Theta[i, j, k], F[i, j, k])

            Sharing[i, j, k] = 1 if (tplp, seller) == (1, 1) else 0

# Extract coordinates

theta\_shared, L\_s\_shared, f\_shared = Theta[Sharing == 1], L\_s[Sharing == 1], F[Sharing == 1]

theta\_no\_share, L\_s\_no\_share, f\_no\_share = Theta[Sharing == 0], L\_s[Sharing == 0], F[Sharing == 0]

# Plotting

fig = plt.figure(figsize=(12, 8))

ax = fig.add\_subplot(111, projection='3d')

# Scatter plot for sharing and no-sharing

ax.scatter(L\_s\_shared, theta\_shared, f\_shared, c='green', label='Sharing (1,1)', alpha=0.6)

ax.scatter(L\_s\_no\_share, theta\_no\_share, f\_no\_share, c='red', label='No Sharing (0,0)', alpha=0.3)

# Axes labels

ax.set\_xlabel('L\_s (Service Level)')

ax.set\_ylabel('Theta (Market Potential)')

ax.set\_zlabel('f (Price Factor)')

ax.set\_title('3D Region Plot: Sharing vs No Sharing')

ax.legend()

plt.tight\_layout()

plt.show()

### Code for policy extraction (Figure 29)

import pygame

import numpy as np

from ray.rllib.policy.policy import Policy

from Logistics\_Service\_Model import LogisticsServiceModel

from gymnasium import spaces

import matplotlib.pyplot as plt

from StackelBerg import stackelberg\_game

# Load trained policies

tplp\_policy = Policy.from\_checkpoint("PPO\\Theta\_5\_2\_1\\checkpoint\_000023\\policies\\tplp\_policy")

# Initialize Pygame

pygame.init()

theta\_init = 5

# Constants

SCREEN\_WIDTH = 1000

SCREEN\_HEIGHT = 600

FONT\_SIZE = 24

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

BUTTON\_COLOR = (100, 200, 100)

SLIDER\_COLOR = (200, 200, 200)

KNOB\_COLOR = (50, 150, 250)

# Initialize screen

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption("Human-in-the-Loop TPLP Game")

font = pygame.font.Font(None, FONT\_SIZE)

# Environment class

class HumanInTheLoopEnv:

    def \_\_init\_\_(self, theta=theta\_init):

        self.agents = ["e\_tailer", "seller", "tplp"]

        self.theta = theta

        self.state = np.array([self.theta, 0.5, 1, 0, 0])

        self.rewards = {agent: 0 for agent in self.agents}

        self.cumulative\_rewards = {agent: 0 for agent in self.agents}

        self.model = LogisticsServiceModel(self.state[1], self.state[0], self.state[2])

        self.action\_spaces = {

            "e\_tailer": spaces.Discrete(2),

            "seller": spaces.Discrete(2),

            "tplp": spaces.Box(low=np.array([0, 0.5]), high=np.array([10, 3]), dtype=np.float64)  # L\_s and f both continuous

        }

    def reset(self):

        self.state = np.array([self.theta, 0.5, 1, 0, 0])

        self.rewards = {agent: 0 for agent in self.agents}

        self.cumulative\_rewards = {agent: 0 for agent in self.agents}

        return self.state

    def observe(self, agent):

        if agent == "tplp":

            obs = np.array([self.obstate[0],self.obstate[4]], dtype=np.float64)

            return obs

    def step(self, tplp\_action):

        self.state[1] = tplp\_action["L\_s"]

        self.state[2] = tplp\_action["f"]

        e\_tailer\_action,seller\_action = stackelberg\_game(self.state[1],self.state[0],self.state[2])

        if e\_tailer\_action and seller\_action == 1:

            self.state[4] = 1

        else:

            self.state[4] = 0

        self.model = LogisticsServiceModel(self.state[1], self.state[0], self.state[2])

        self.rewards = self.calculate\_rewards()

        for agent in self.agents:

            self.cumulative\_rewards[agent] += self.rewards[agent]

        print(self.state[4])

        return self.state, self.rewards

    def calculate\_rewards(self):

        e\_tailer\_reward = self.calculate\_profit("e\_tailer")

        seller\_reward = self.calculate\_profit("seller")

        tplp\_reward = self.calculate\_profit("tplp")

        return {

            "e\_tailer": e\_tailer\_reward,

            "seller": seller\_reward,

            "tplp": tplp\_reward,

        }

    def calculate\_profit(self, agent):

        profit\_et\_no\_sharing = self.model.profit\_nosharing\_etailer()

        profit\_et\_sharing = self.model.profit\_sharing\_etailer(True)

        profit\_seller\_no\_sharing = self.model.profit\_nosharing\_seller()

        profit\_seller\_sharing = self.model.profit\_sharing\_seller(True)

        # Calculate profit differences directly in the loop

        profit\_diff\_et = profit\_et\_sharing - profit\_et\_no\_sharing

        profit\_diff\_seller = profit\_seller\_sharing - profit\_seller\_no\_sharing

        self.ww = (profit\_diff\_et >= -1e-8) and (profit\_diff\_seller >=-1e-8)

        theta = self.state[0]  # Market potential

        L\_s = self.state[1]    # Seller's service level

        f = self.state[2]      # Logistics price

        sharing\_status = self.state[4]  # Logistics sharing status

        # Reinitialize the model with updated obstate variables

        self.model.L\_s = L\_s

        self.model.theta = theta

        if agent == "tplp":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_tplp()

            else:

                profit = self.model.profit\_sharing\_tplp(self.ww)

            print(profit)

        if agent == "e\_tailer":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_etailer()

            else:

                profit = self.model.profit\_sharing\_etailer(self.ww)

        elif agent == "seller":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_seller()

            else:

                profit = self.model.profit\_sharing\_seller(self.ww)

        return profit

    def plot\_profit\_regions(self,ax, theta\_value, sharing\_status):

        # Define range for L\_s and f

        L\_s\_values = np.linspace(1, 10, 100)

        f\_values = np.linspace(0.5, 3, 100)

        profit\_matrix = np.zeros((len(L\_s\_values), len(f\_values)))

        # Compute profits

        for i, L\_s in enumerate(L\_s\_values):

            for j, f in enumerate(f\_values):

                model = LogisticsServiceModel(L\_s, theta\_value, f)

                ww = (sharing\_status == 1)  # sharing\_status flag

                profit\_matrix[i, j] = model.profit\_sharing\_tplp(ww) if ww else model.profit\_nosharing\_tplp()

        # Plot profit regions

        c = ax.contourf(f\_values, L\_s\_values, profit\_matrix, cmap='viridis', levels=50)

        ax.set\_xlabel("Logistics Price (f)")

        ax.set\_ylabel("Service Level (L\_s)")

        ax.set\_title(f"TPLP Profit Regions - {'Sharing' if sharing\_status == 1 else 'No Sharing'}")

        plt.colorbar(c, ax=ax, label="Profit")

class Machine:

    def \_\_init\_\_(self, theta=theta\_init):

        self.agents = ["e\_tailer", "seller", "tplp"]

        self.theta = theta

        self.state = np.array([self.theta, 0.5, 1, 0, 0])

        self.rewards = {agent: 0 for agent in self.agents}

        self.cumulative\_rewards = {agent: 0 for agent in self.agents}

        self.model = LogisticsServiceModel(self.state[1], self.state[0], self.state[2])

        self.action\_spaces = {

            "e\_tailer": spaces.Discrete(2),

            "seller": spaces.Discrete(2),

            "tplp": spaces.Box(low=np.array([0, 0.5]), high=np.array([10, 3]), dtype=np.float64)  # L\_s and f both continuous

        }

    def reset(self):

        self.state = np.array([self.theta, 0.5, 1, 0, 0])

        self.rewards = {agent: 0 for agent in self.agents}

        self.cumulative\_rewards = {agent: 0 for agent in self.agents}

        return self.state

    def observe(self, agent):

        if agent == "e\_tailer":

            return np.array([self.state[0], self.state[1], self.state[2]], dtype=np.float64)

        elif agent == "seller":

            return np.array([self.state[0], self.state[1], self.state[2]], dtype=np.float64)

        elif agent == "tplp":

            return np.array([self.state[0], self.state[4]], dtype=np.float64)

    def step(self):

        tplp\_action = tplp\_policy.compute\_single\_action(self.observe("tplp"),clip\_actions=True,explore=False)[0]

        tplp\_action = self.action\_spaces["tplp"].low + (self.action\_spaces["tplp"].high - self.action\_spaces["tplp"].low) \* ((np.tanh(tplp\_action) + 1) / 2)

        print(tplp\_action)

        self.state[1] = tplp\_action[0]

        self.state[2] = tplp\_action[1]

        e\_tailer\_action,seller\_action = stackelberg\_game(self.state[1],self.state[0],self.state[2])

        if e\_tailer\_action and seller\_action == 1:

            self.state[4] = 1

        else:

            self.state[4] = 0

        self.model = LogisticsServiceModel(self.state[1], self.state[0], self.state[2])

        self.rewards = self.calculate\_rewards()

        for agent in self.agents:

            self.cumulative\_rewards[agent] += self.rewards[agent]

        print(self.state[4])

        return self.state, self.rewards

    def calculate\_rewards(self):

        e\_tailer\_reward = self.calculate\_profit("e\_tailer")

        seller\_reward = self.calculate\_profit("seller")

        tplp\_reward = self.calculate\_profit("tplp")

        return {

            "e\_tailer": e\_tailer\_reward,

            "seller": seller\_reward,

            "tplp": tplp\_reward,

        }

    def calculate\_profit(self, agent):

        profit\_et\_no\_sharing = self.model.profit\_nosharing\_etailer()

        profit\_et\_sharing = self.model.profit\_sharing\_etailer(True)

        profit\_seller\_no\_sharing = self.model.profit\_nosharing\_seller()

        profit\_seller\_sharing = self.model.profit\_sharing\_seller(True)

        # Calculate profit differences directly in the loop

        profit\_diff\_et = profit\_et\_sharing - profit\_et\_no\_sharing

        profit\_diff\_seller = profit\_seller\_sharing - profit\_seller\_no\_sharing

        self.ww = (profit\_diff\_et >= -1e-8) and (profit\_diff\_seller >=-1e-8)

        theta = self.state[0]  # Market potential

        L\_s = self.state[1]    # Seller's service level

        f = self.state[2]      # Logistics price

        sharing\_status = self.state[4]  # Logistics sharing status

        # Reinitialize the model with updated obstate variables

        self.model.L\_s = L\_s

        self.model.theta = theta

        if agent == "tplp":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_tplp()

            else:

                profit = self.model.profit\_sharing\_tplp(self.ww)

            print(f"machine:{profit}")

        if agent == "e\_tailer":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_etailer()

            else:

                profit = self.model.profit\_sharing\_etailer(self.ww)

        elif agent == "seller":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_seller()

            else:

                profit = self.model.profit\_sharing\_seller(self.ww)

        return profit

# Helper functions

def draw\_text(surface, text, x, y):

    text\_surface = font.render(text, True, WHITE)

    surface.blit(text\_surface, (x, y))

def draw\_button(surface, x, y, width, height, text):

    """Draw a button with text."""

    pygame.draw.rect(surface, BUTTON\_COLOR, (x, y, width, height))

    text\_surface = font.render(text, True, WHITE)

    text\_rect = text\_surface.get\_rect(center=(x + width // 2, y + height // 2))

    surface.blit(text\_surface, text\_rect)

def draw\_slider(surface, x, y, width, min\_val, max\_val, step, value):

    # Draw the slider background

    pygame.draw.rect(surface, SLIDER\_COLOR, (x, y, width, 10))

    # Map the current value to the slider's knob position

    knob\_x = x + int((value - min\_val) / (max\_val - min\_val) \* width)

    # Draw the knob

    pygame.draw.circle(surface, KNOB\_COLOR, (knob\_x, y + 5), 10)

    return knob\_x

def slider\_value(mouse\_x, x, width, min\_val, max\_val, step):

    # Calculate the relative position of the mouse within the slider

    relative\_pos = min(max(mouse\_x - x, 0), width)  # Ensure within slider bounds

    # Calculate the slider value based on the relative position

    value\_range = max\_val - min\_val

    value = min\_val + (relative\_pos / width) \* value\_range

    # Round to the nearest step

    value = round(value / step) \* step

    # Clip to ensure the value is within the valid range

    return np.clip(value, min\_val, max\_val)

# Initialize environment

env = HumanInTheLoopEnv()

env\_machine = Machine()

state = env.reset()

# Slider properties

slider\_width = 200

slider\_x = 400

slider\_L\_s\_y = 50

slider\_f\_y = 150

slider\_L\_s\_value = 0

slider\_f\_value = 0.5

# Button properties

button\_x = 400

button\_y = 500

button\_width = 150

button\_height = 50

input\_active\_L\_s = False

input\_active\_f = False

input\_text\_L\_s = "1.0"

input\_text\_f = "0.5"

input\_box\_L\_s = pygame.Rect(slider\_x, slider\_L\_s\_y, 100, 32)

input\_box\_f = pygame.Rect(slider\_x, slider\_f\_y, 100, 32)

color\_inactive = pygame.Color('lightskyblue3')

color\_active = pygame.Color('dodgerblue2')

color\_L\_s = color\_inactive

color\_f = color\_inactive

# Main game loop

running = True

iterations = 0

while running:

    screen.fill(BLACK)

    # Display current state

    draw\_text(screen, f"Iteration number: {iterations}", 50, 0)

    draw\_text(screen, f"Theta: {theta\_init}", 50, 30)

    draw\_text(screen, f"Service Level (L\_s): {state[1]:.2f}", 50, 60)

    draw\_text(screen, f"Logistics Price (f): {state[2]:.2f}", 50, 90)

    draw\_text(screen, f"E-tailer Sharing: {state[3]}", 50, 120)

    draw\_text(screen, f"Seller Sharing: {state[4]}", 50, 150)

    # L\_s input box

    draw\_text(screen, "Enter Service Level (L\_s):", slider\_x, slider\_L\_s\_y - 30)

    color\_L\_s = color\_active if input\_active\_L\_s else color\_inactive

    pygame.draw.rect(screen, color\_L\_s, input\_box\_L\_s, 2)

    txt\_surface = font.render(input\_text\_L\_s, True, WHITE)

    screen.blit(txt\_surface, (input\_box\_L\_s.x + 5, input\_box\_L\_s.y + 5))

    # f input box

    draw\_text(screen, "Enter Logistics Price (f):", slider\_x, slider\_f\_y - 30)

    color\_f = color\_active if input\_active\_f else color\_inactive

    pygame.draw.rect(screen, color\_f, input\_box\_f, 2)

    txt\_surface = font.render(input\_text\_f, True, WHITE)

    screen.blit(txt\_surface, (input\_box\_f.x + 5, input\_box\_f.y + 5))

    # Draw the submit button

    draw\_button(screen, button\_x, button\_y, button\_width, button\_height, "Submit")

    # Display rewards

    draw\_text(screen, f"Rewards:", 50, 200)

    for i, (agent, reward) in enumerate(env.rewards.items()):

        draw\_text(screen, f"{agent}: {reward:.2f}", 50, 230 + i \* 30)

    # Display cumulative rewards

    draw\_text(screen, f"Cumulative Rewards You:", 50, 320)

    for i, (agent, cum\_reward) in enumerate(env.cumulative\_rewards.items()):

        draw\_text(screen, f"{agent}: {cum\_reward:.2f}", 50, 350 + i \* 30)

    draw\_text(screen, f"Cumulative Rewards Machine:", 50, 440)

    for i, (agent, cum\_reward) in enumerate(env\_machine.cumulative\_rewards.items()):

        draw\_text(screen, f"{agent}: {cum\_reward:.2f}", 50, 470 + i \* 30)

    # Event handling

    submit\_pressed = False

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

        elif event.type == pygame.MOUSEBUTTONDOWN:

            if input\_box\_L\_s.collidepoint(event.pos):

                input\_active\_L\_s = True

                input\_active\_f = False

            elif input\_box\_f.collidepoint(event.pos):

                input\_active\_f = True

                input\_active\_L\_s = False

            else:

                input\_active\_L\_s = False

                input\_active\_f = False

            # Check if submit button is pressed

            if button\_x <= event.pos[0] <= button\_x + button\_width and button\_y <= event.pos[1] <= button\_y + button\_height:

                submit\_pressed = True

        elif event.type == pygame.KEYDOWN:

            if input\_active\_L\_s:

                if event.key == pygame.K\_RETURN:

                    input\_active\_L\_s = False

                elif event.key == pygame.K\_BACKSPACE:

                    input\_text\_L\_s = input\_text\_L\_s[:-1]

                else:

                    input\_text\_L\_s += event.unicode

            elif input\_active\_f:

                if event.key == pygame.K\_RETURN:

                    input\_active\_f = False

                elif event.key == pygame.K\_BACKSPACE:

                    input\_text\_f = input\_text\_f[:-1]

                else:

                    input\_text\_f += event.unicode

    # Only update the environment and increment iterations if the submit button is pressed

    if submit\_pressed:

        try:

            L\_s\_val = float(input\_text\_L\_s)

            f\_val = float(input\_text\_f)

            # Validate inputs

            if not (0 <= L\_s\_val <= 10 and 0.5 <= f\_val <= 3):

                print("L\_s must be between [0, 10] and f must be between [0.5, 3]")

                continue

            tplp\_action = {"L\_s": L\_s\_val, "f": f\_val}

            state, rewards = env.step(tplp\_action)

            state\_machine, rewards\_machine = env\_machine.step()

            iterations += 1

            # Plot

            fig, ax = plt.subplots(1, 2, figsize=(12, 5))

            env.plot\_profit\_regions(ax[0], theta\_init, state[4])

            env.plot\_profit\_regions(ax[1], theta\_init, 0)

            plt.tight\_layout()

            plt.show()

        except ValueError:

            print("Invalid input. Enter valid numeric values for L\_s and f.")

    # Update display

    pygame.display.flip()

### Code for PPO RL

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.patches as mpatches

import matplotlib.gridspec as gridspec

import os

from Logistics\_Service\_Model import LogisticsServiceModel

from StackelBerg import stackelberg\_game

# Creating the environment

from pettingzoo.utils import AECEnv

from pettingzoo.utils import agent\_selector

from gymnasium import spaces

from gymnasium.envs.registration import register

# Reinforcement learning model

from torch import nn

from ray.tune.registry import register\_env

from ray.rllib.env import PettingZooEnv

from ray import tune

from ray.rllib.algorithms.ppo import PPOConfig

from ray.rllib.policy.policy import PolicySpec

from ray.rllib.models import ModelCatalog

from ray.rllib.models.torch.torch\_modelv2 import TorchModelV2

from ray.rllib.algorithms.callbacks import DefaultCallbacks

from ray.tune import Stopper

def randomise\_conditions():

    theta = 3

    return theta

class CoopetitionEnv(AECEnv):

    metadata = {"render\_modes": ["human"], "name": "LogisticsServiceModel"}

    def \_\_init\_\_(self,theta,max\_iterations=50):

        super(CoopetitionEnv, self).\_\_init\_\_()

        self.theta = theta

        self.agents = ["tplp"]

        self.\_agent\_selector = agent\_selector(self.agents)

        self.possible\_agents = self.agents[:]

        # Observation state: [market\_potential, L\_s, f, 0 -noshare, 1- share x 2]

        self.obstate = np.array([self.theta, 0.5, 1, 0, 0], dtype=np.float64)

        # Action spaces

        self.action\_spaces = {

            "tplp": spaces.Box(low=np.array([0, 0.5]), high=np.array([10, 3]), dtype=np.float64)  # L\_s and f both continuous

        }

        # Observation spaces

        self.observation\_spaces = {

            "tplp": spaces.Box(low=np.array([0,0]), high=np.array([10,1]), dtype=np.float64)  # Second decision (whether sharing is active)

        }

        self.terminations = {agent: False for agent in self.agents}

        self.truncations = {agent: False for agent in self.agents}

        self.dones = {agent: False for agent in self.agents}

        self.infos = {agent: {} for agent in self.agents}

        self.model = LogisticsServiceModel(L\_s=self.obstate[1],f=self.obstate[2], theta=self.obstate[0])

        self.terminate = False

        self.truncate = False

        self.max\_iterations = max\_iterations

    def observation\_space(self,agent):

        return self.observation\_spaces[agent]

    def action\_space(self,agent):

        return self.action\_spaces[agent]

    def observe(self, agent):

        if agent == "tplp":

            obs = np.array([self.obstate[0],self.obstate[4]], dtype=np.float64)

            return obs

    def reset(self, seed = None, options = None):

        self.theta = randomise\_conditions()

        self.obstate = np.array([self.theta, 0.5, 1, 0, 0], dtype=np.float64)

        self.agents = self.possible\_agents[:]

        self.terminate = False

        self.truncate = False

        self.terminations = {agent: False for agent in self.agents}

        self.truncations = {agent: False for agent in self.agents}

        self.infos = {agent: {} for agent in self.agents}

        self.observations = {agent: 0 for agent in self.agents}

        self.\_cumulative\_rewards = {agent: 0 for agent in self.agents}

        self.rewards = {agent: 0 for agent in self.agents}

        self.num\_iterations = 0

        self.model = LogisticsServiceModel(L\_s=self.obstate[1],f=self.obstate[2], theta=self.obstate[0])

        # Agent selector utility

        self.\_agent\_selector.reinit(self.agents)

        self.agent\_selection = self.\_agent\_selector.next()

    def state(self):

        """Returns an observation of the global environment."""

        state = self.obstate.copy()

        return state

    def step(self, action):

        if (self.terminations[self.agent\_selection] or self.truncations[self.agent\_selection]):

            self.\_was\_dead\_step(action)

            return

        action = np.asarray(action)

        agent = self.agent\_selection

        if agent == "tplp":

            self.obstate[1], self.obstate[2] = action

            e\_tailer\_act,seller\_act = stackelberg\_game(self.obstate[1],self.obstate[0],self.obstate[2])

            if e\_tailer\_act and seller\_act == 1:

                self.obstate[4] = 1

        if self.\_agent\_selector.is\_last():

            self.model = LogisticsServiceModel(self.obstate[1], self.obstate[0], self.obstate[2])

            for agent in self.agents:

                self.rewards[agent] = self.calculate\_profit(agent)

            self.num\_iterations += 1

            for i in self.agents:

                self.observations[i] = self.observe(i)

        else:

            self.\_clear\_rewards()

        if self.\_agent\_selector.is\_last():

            self.truncate = self.num\_iterations >= self.max\_iterations

            self.terminate = self.num\_iterations >= self.max\_iterations

            self.terminations = dict(

                zip(self.agents, [self.terminate for \_ in self.agents])

            )

            self.truncations = dict(

                zip(self.agents, [self.truncate for \_ in self.agents])

            )

        self.agent\_selection = self.\_agent\_selector.next()

        self.\_cumulative\_rewards[agent] = 0

        self.\_accumulate\_rewards()

    def calculate\_profit(self, agent):

        profit\_et\_no\_sharing = self.model.profit\_nosharing\_etailer()

        profit\_et\_sharing = self.model.profit\_sharing\_etailer(True)

        profit\_seller\_no\_sharing = self.model.profit\_nosharing\_seller()

        profit\_seller\_sharing = self.model.profit\_sharing\_seller(True)

        # Calculate profit differences directly in the loop

        profit\_diff\_et = profit\_et\_sharing - profit\_et\_no\_sharing

        profit\_diff\_seller = profit\_seller\_sharing - profit\_seller\_no\_sharing

        self.ww = (profit\_diff\_et >= -1e-8) and (profit\_diff\_seller >=-1e-8)

        theta = self.state()[0]  # Market potential

        L\_s = self.state()[1]    # Seller's service level

        f = self.state()[2]      # Logistics price

        sharing\_status = self.state()[4]  # Logistics sharing status

        # Reinitialize the model with updated obstate variables

        self.model.L\_s = L\_s

        self.model.theta = theta

        if agent == "tplp":

            if sharing\_status == 0:

                profit = self.model.profit\_nosharing\_tplp()

            else:

                profit = self.model.profit\_sharing\_tplp(self.ww)

        if np.isnan(profit):

            print(f"Warning: Profit for agent {agent} is NaN")

        return profit

    def render(self, mode="human"):

        print(f"Current obstate: {self.obstate}")

    def close(self):

        return

class CustomCallbacks(DefaultCallbacks):

    def on\_episode\_end(self, \*, worker, base\_env, policies, episode, \*\*kwargs):

        # Log individual agent rewards

        for agent\_id, reward in episode.agent\_rewards.items():

            print(f"Agent {agent\_id} reward: {reward}")

        # Store mean reward across episode for stability checking

        mean\_reward = np.mean([r for \_, r in episode.agent\_rewards.items()])

        episode.custom\_metrics["mean\_episode\_reward"] = mean\_reward

class PercentageVarianceStopper(Stopper):

    def \_\_init\_\_(self, patience=5, percentage\_threshold=0.0, max\_timesteps=5000000):

        self.recent\_rewards = []

        self.patience = patience

        self.percentage\_threshold = percentage\_threshold

        self.max\_timesteps = 50000 if os.environ.get("CI") else max\_timesteps

    def \_\_call\_\_(self, trial\_id, result):

        timesteps = result.get("timesteps\_total", 0)

        reward = result["custom\_metrics"].get("mean\_episode\_reward")

        # Stop if reward variance is below percentage of mean

        if reward is not None:

            self.recent\_rewards.append(reward)

            if len(self.recent\_rewards) > self.patience:

                self.recent\_rewards.pop(0)

                mean = np.mean(self.recent\_rewards)

                std = np.std(self.recent\_rewards)

                if std / mean < self.percentage\_threshold:

                    return True

        # Stop if timesteps exceeded

        if timesteps >= self.max\_timesteps:

            return True

        return False

    def stop\_all(self):

        return False

def env\_creator(args):

    env = CoopetitionEnv(theta=randomise\_conditions())

    return env

# Register environment with Ray

register\_env("coopetition\_env\_single", lambda config: PettingZooEnv(env\_creator(config)))

config = (

    PPOConfig()

    .environment(env="coopetition\_env\_single", clip\_actions=True)

    .rollouts(num\_rollout\_workers=6, rollout\_fragment\_length='auto')

    .training(

        train\_batch\_size=1024,

        lr=1e-7,

        gamma=0.99,

        lambda\_=0.9,

        use\_gae=True,

        clip\_param=0.2,

        grad\_clip=None,

        entropy\_coeff=0.1,

        vf\_loss\_coeff=0.25,

        sgd\_minibatch\_size=64,

        num\_sgd\_iter=10,

    )

    .debugging(log\_level="ERROR")

    .framework(framework="torch")

    .resources(num\_gpus=int(os.environ.get("RLLIB\_NUM\_GPUS", "0")))

    .callbacks(CustomCallbacks)

    )

config.multi\_agent(

    policies={

        "tplp\_policy": PolicySpec(),

    },

    policy\_mapping\_fn=lambda agent\_id, \*args, \*\*kwargs: (

        "tplp\_policy" if agent\_id == "tplp" else f"{agent\_id}\_policy"

    ),

)

# Define log directory

logdir = os.getcwd()

tune.run(

     "PPO", name="PPO",

     stop=PercentageVarianceStopper(patience=5, percentage\_threshold=0.01, max\_timesteps=5000000),

     checkpoint\_freq=10,

     storage\_path=logdir,

     config=config.to\_dict(),

)