universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Machine Learning for Attended Home Delivery

verfasst von | submitted by

Yujie Wang B.Sc.

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 977
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Business Analytics
Betreut von | Supervisor:	Univ.-Prof. Dr. Jan Fabian Ehmke

Abstract

Attended-Home-Delivery(AHD) Systeme, ein Eckpfeiler des modernen E-Commerce, stehen vor erheblichen operativen Herausforderungen, die Balance zwischen Kundenpraeferenzen, Lieferleistung und logistischen Einschrnkungen zu finden. Zeitfenster spielen eine entscheidende Rolle in AHD-Systemen, da sie verpasste Lieferungen minimieren und die Kundenzufriedenheit erhoehen, gleichzeitig jedoch die Komplexitaet in der Routenplanung und Ressourcenallokation steigern. Diese Studie untersucht die Machbarkeit des Einsatzes von Machine-Learning(ML) zur Vorhersage der Zeitfenster-Machbarkeit in AHD-Systemen, wobei der Fokus auf Gradient Boosting, Neural Networks und Random Forests liegt. Die Forschung hebt die zentrale Rolle des Feature Engineerings hervor und zeigt, dass aggregierte Merkmale, die rumliche und zeitliche Beziehungen erfassen, die Modellgenauigkeit signifikant verbessern. Simulationsexperimente, die auf realistischen Lieferszenarien in Wien basieren, zeigen, dass Neural Networks eine ausgewogene Leistung bei der Klassifikation von machbaren und nicht machbaren Zeitfenstern bieten, waehrend Gradient Boosting sich besonders bei der Identifikation nicht machbarer Zeitfenster auszeichnet und Random Forests robuste, interpretierbare Lsungen liefern. Trotz der guten Leistungen identifiziert die Studie Einschrnkungen in der Skalierbarkeit, den Unsicherheiten der realen Welt und der Interpretierbarkeit der Modelle und betont die Notwendigkeit weiterer Forschung.

Keywords: Machine Learning, Zeitfenstermanagement, Problem der Fahrzeugroutenplanung, Logistikoptimierung

Acknowledgments

I would like to express my heartfelt gratitude to my supervisor, **Univ.-Prof. Dr. Jan Fabian Ehmke**, for his invaluable guidance, timely feedback, and continuous support throughout the development of this thesis. His constructive insights and expertise have been instrumental in shaping the direction and quality of this work, and his prompt responses provided clarity during critical phases of this research. I am also deeply thankful to my family and friends for their unwavering encouragement and understanding during the demanding periods of this journey. Their support has been a constant source of strength and motivation. Finally, I extend my appreciation to the faculty and staff at the University of Vienna for creating a supportive academic environment and providing access to resources that made this research possible.

I acknowledge the use of artificial intelligence tools to refine the grammar and clarity of this thesis. These tools were instrumental in ensuring the language of this document met a high standard of academic writing.

Contents

List of Figures

List of Tables

1 Introduction

The COVID-19 pandemic has significantly accelerated the growth of online shopping and home delivery services globally. This trend is particularly pronounced in the grocery sector, where the online share of annual sales surged from 7 percent to 10 percent during the pandemic and has stabilized at 9 percent after the pandemic peak[20]. This sustained demand presents unique challenges for last-mile delivery operations, especially for attended home delivery, where recipients must be present to receive their orders[10]. Attended home delivery is commonly used for groceries, large appliances, and furniture products that often require special handling. To ensure timely and successful deliveries, service providers typically offer customers a choice of delivery time slots. While this approach reduces missed deliveries and improves customer satisfaction, it also adds operational complexity, as it requires balancing customer preferences with delivery efficiency. This integration of customer involvement in service creation is a hallmark of service operations management[19].

In this context, determining feasible time slots for new orders is a crucial aspect of attended home delivery. A time slot is feasible if a route can accommodate the new customer without exceeding capacity or disrupting schedules, after the order cut-off time.[19]. Solving this problem involves addressing the computationally challenging Vehicle Routing Problem with Time Windows (VRPTW), especially for large-scale delivery operations[19]. Traditional methods, such as insertion heuristics, while widely used, often fall short in terms of accuracy, scalability, and adaptability to complex constraints[20]. As a result, there is a growing need for alternative approaches that can efficiently handle the computational and operational demands of modern time slot management systems.

This paper addresses this challenge by exploring the use of machine learning (ML) techniques to predict feasible time slots in attended home delivery. Building on the work of van der Hagen et al.[19], this study replicates their experiments while extending the application to a different location. By leveraging MLs ability to identify patterns in historical data, the proposed approach aims to improve prediction accuracy, reduce computational overhead, and enable real-time decision-making. The contributions of this research are threefold: firstly, it validates the findings of van der Hagen et al.[19] through experimental replication. Secondly, it explores the potential of the generalization of the ML method in different places. And thirdly, it provides operational insights into how ML-based time slot management can improve the route efficiency, especially in large-scale delivery systems with different time slots.

The remainder of this paper is organized as follows. Section 2 reviews relevant literature on attended home delivery systems, time slot management,

VRPTW and the application of ML in operational research. Section 3 details the methodology, including problem complexities, description of the arrival process, and ML model selection. Section 4 describes the simulation framework, followed by the presentation of results and comparative analysis in Section 5. Section 6 concludes with key findings, discusses the studys limitations and suggests avenues for future research.

2 Literature Review

This section provides a focused review of topics central to this study, including attended home delivery systems, dynamic time slot management, the VRPTW, and the role of ML in logistics optimization. These areas are intricately tied to the challenges and opportunities of predicting time slot feasibility, which is the core theme of this research.

2.1 Attended Home Delivery

Attended home delivery (AHD) systems, a key component of modern electronic business, have rapidly evolved to meet the increasing demand for flexible and reliable delivery services[2]. These systems, which require customers to be present during specific delivery time windows, present unique operational challenges in balancing customer satisfaction, cost efficiency, and operational feasibility. Precise time slot offerings are a key customer expectation, yet they restrict routing flexibility, leading to higher operational costs[16]. Real-world uncertainties, such as traffic congestion, fluctuating demand, and last-minute order changes, exacerbate this complexity, making the underlying VRPTW computationally demanding[6]. Additionally, predicting customer behavior and dynamically adjusting time slots to manage peak demand requires sophisticated algorithms and real-time data processing. These challenges are further amplified in high-density urban areas or during peak periods, where scalable solutions are critical[2].

To address these challenges, researchers have extensively studied dynamic time slot management and the application of ML in logistics optimization[5]. ML models offer predictive insights into time slot feasibility, enhancing decision-making and operational efficiency[4]. Dynamic pricing and incentivized off-peak deliveries can balance demand and reduce peak-load issues[16]. Integrating ML with traditional optimization techniques, such as heuristics for VRPTW, provides hybrid solutions that combine predictive accuracy with computational efficiency. Opportunities also lie in real-time data utilization, such as traffic and weather updates, to dynamically adjust delivery routes and schedules[11]. These approaches not only improve last-mile delivery efficiency but also enable more sustainable and customer-centric logistics strategies[20].

2.2 Dynamic Time Slot Management

Dynamic Time Slot Management (DTSM) is a strategic approach in AHD systems designed to dynamically optimize the allocation of delivery time slots in response to real-time data and operational conditions[11]. Unlike traditional static systems, which offer predefined time slots without considering

current demand and logistical constraints, DTSM adapts to fluctuating customer preferences, demand patterns, and operational capacity. This approach enhances both customer satisfaction and cost efficiency by ensuring the flexible and effective use of delivery resources while meeting customer expectations for convenient and precise delivery windows [13].

DTSM addresses critical delivery challenges through multiple innovative mechanisms. One such approach is dynamic pricing strategies, which adjust delivery fees based on time slot demand and operational constraints[16]. By incentivizing customers to choose off-peak slots or strategically less congested times through lower prices, dynamic pricing helps balance demand and improve resource utilization. High-demand slots can command premium prices to ensure profitability and maintain service quality during peak periods [16]. Another key aspect of DTSM is customer behavior analysis, which leverages predictive analytics to anticipate customer preferences and dynamically adjust time slot offerings. By analyzing historical and real-time data, DTSM identifies patterns in customer behavior, enabling the system to allocate time slots effectively and steer demand toward operationally favorable slots. This alignment reduces underutilization of delivery capacity and enhances overall network efficiency [10]. Integration of demand management with vehicle routing is another critical component of DTSM. Real-time routing algorithms account for existing commitments while incorporating new customer requests dynamically. This ensures minimal disruption to planned routes while maximizing vehicle utilization. For example, algorithms can reconfigure routes in real time to optimize resource allocation and meet delivery deadlines, improving service reliability [9]. DTSM also benefits from simulation-based frameworks, which assess the feasibility of offering specific time slots under current operational constraints. These simulations consider factors such as traffic, order density, and vehicle capacity to generate realistic and achievable delivery options. By doing so, DTSM ensures that customers are only presented with feasible choices, reducing the likelihood of failed deliveries and enhancing customer satisfaction [1]. Lastly, tactical and operational planning solutions complement DTSM by addressing uncertainties in customer preferences and delivery conditions. Probabilistic models and ML techniques are used to make informed decisions even in scenarios with incomplete or noisy data. These methods improve the system's adaptability to dynamic environments, ensuring robust performance in diverse delivery contexts [12].

Overall, DTSM integrates dynamic pricing, customer behavior analysis, real-time routing, simulation-based evaluations, and predictive planning to create a robust framework for managing delivery time slots. By combining these strategies, DTSM ensures that AHD systems can effectively balance customer satisfaction with operational efficiency, meeting the complex demands

of modern e-commerce logistics[5].

2.3 Vehicle Routing Problem with Time Windows

VRPTW is an advanced and widely studied extension of the classical Vehicle Routing Problem (VRP), which serves as a foundational problem in logistics and operations research. The classical VRP focuses on determining optimal delivery routes for a fleet of vehicles starting and ending at a central depot, aiming to minimize costs such as travel distance or time while ensuring each customer is visited once and vehicle capacities are respected [18]. VRPTW introduces an additional layer of complexity by imposing strict delivery time windows for each customer, requiring vehicles to arrive within specified time ranges [6]. These constraints make VRPTW significantly more challenging computationally, categorizing it as an NP-hard problem, especially for large-scale instances [5].

The primary challenges of VRPTW stem from its computational complexity, scalability, and adaptability to real-world uncertainties. Exact methods, such as branch-and-bound and branch-and-cut algorithms, although precise, are often computationally infeasible for large problem instances involving hundreds or thousands of customers [18]. The dynamic and stochastic nature of real-world logistics, including factors like traffic, weather, and last-minute customer requests, adds further complexity [9]. Additionally, VRPTW scenarios often involve multiple constraints, such as heterogeneous vehicle capacities, varied service times, and dynamic customer requests, increasing the problems complexity and making static models inadequate [7].

To address these challenges, researchers have explored various solution approaches and opportunities. Heuristic and metaheuristic methods, such as genetic algorithms, tabu search, and simulated annealing, are commonly employed to provide efficient, near-optimal solutions for large-scale problems [15]. ML techniques, including reinforcement learning and supervised learning, have emerged as promising tools for solving VRPTW, particularly in dynamic environments [5]. ML models can enhance adaptability by enabling real-time decision-making and optimizing routes in response to changing conditions [7]. Recent advancements in dynamic VRPs and hybrid approaches that combine ML with traditional optimization techniques have further improved the practicality of solving VRPTW [4]. Moreover, the integration of real-time data and simulation frameworks has enabled more comprehensive and adaptable solutions [21].

By leveraging these advancements, VRPTW solutions are becoming increasingly capable of addressing the complexities of modern logistics[8]. These approaches not only improve operational efficiency but also offer opportunities

to tackle richer and more realistic problem variations, such as multi-depot scenarios or pickup-and-delivery constraints, further enhancing the applicability of VRPTW in real-world delivery systems [6].

2.4 Machine Learning in AHD and VRPTW

ML has emerged as a transformative tool for logistics optimization, particularly in solving VRPs and managing time slot feasibility in AHD[14]. Unlike traditional methods, ML models can learn from historical data, capturing complex patterns and enabling real-time decision-making in dynamic environments [5]. Supervised learning approaches have been widely applied to predict delivery feasibility, with studies demonstrating the effectiveness of ML models in solving VRPs, highlighting their ability to adapt to changing inputs and outperform traditional heuristics [4]. Reinforcement learning has also shown promise in adaptive route planning by enabling models to learn optimal policies under stochastic and dynamic conditions [5]. Hybrid methods that combine ML with traditional optimization techniques are particularly effective in addressing large-scale logistics problems. For instance, Graph Neural Networks (GNNs) have been applied to process complex graph structures in delivery networks, enhancing both scalability and accuracy [3]. Attention mechanisms further improve model performance by focusing on critical decision-making elements, such as high-priority customers or congested routes [5]. The integration of ML into operational frameworks has also proven invaluable, with applications using ML to predict service times and optimize routing under real-world uncertainties, such as traffic conditions and customer behavior [11]. Research has also demonstrated the potential of combining tactical and operational planning using ML to bridge the gap between predictive analytics and decision-making frameworks [12].

Despite significant progress, challenges remain in scaling AHD solutions to handle large networks and dynamic environments. One major issue is ensuring the robustness of ML models in real-world settings with noisy and incomplete data [21]. Most studies focus on static scenarios, while real-world applications require solutions that adapt to dynamic and stochastic factors, such as fluctuating demand and unexpected disruptions [5]. Future research emphasizes the need for multi-agent systems, which enable coordinated decision-making among multiple vehicles. These systems, combined with advanced ML techniques such as multi-agent reinforcement learning, hold potential for solving large-scale VRPs in real-time [7]. Additionally, incorporating sustainability goals, such as minimizing carbon footprints and optimizing fuel consumption, is a growing area of interest in AHD optimization [2].

While traditional methods have made significant strides in optimizing

AHD operations, the integration of ML into time slot management holds the potential to revolutionize this field. By leveraging MLs ability to analyze real-world data and adapt to dynamic scenarios, future research can focus on developing predictive models that enable real-time time slot adjustments. This paper aims to explore the application of ML techniques to enhance time slot management strategies, bridging the gap between theoretical optimization and practical implementation in AHD systems.

3 Methodology

This section outlines the methodological framework used to replicate and expand upon the study by van der Hagen et al. [19]. The research employs a ML approach to determine feasible delivery time slots in AHD systems. This section details the problem complexities, the description of the arrival process and the ML models applied.

3.1 Complexities of the Problem

The central problem in AHD systems lies in the feasibility check of time slot management[19]. This involves determining whether a requested delivery time slot can be offered to a customer without violating operational constraints such as vehicle capacity, routing limitations, and predefined delivery time windows. The problem is modeled as a VRPTW, where a fleet of vehicles must efficiently serve customer demands while adhering to these constraints[18]. Each vehicle starts and ends its route at a central depot, and customers must be served exactly once within their selected time window. This NP-hard problem involves determining optimal delivery routes for a fleet of vehicles while adhering to multiple constraints, such as time windows, vehicle capacities, and routing limitations[6]. The interdependencies between these constraints mean that addressing one may negatively impact another. The problem's complexity escalates exponentially with the number of customers and vehicles, making traditional optimization methods computationally infeasible for large-scale instances[21].

Dynamic and stochastic factors further complicate the problem. AHD systems operate in dynamic environments where customer orders are placed in real-time, requiring frequent updates to precomputed routes[7]. Uncertainty arising from traffic conditions, delivery delays, and customer behavior adds unpredictability, making it difficult to ensure that time slots remain feasible at the point of execution. These dynamic and uncertain elements necessitate real-time adaptability in decision-making[14]. Scalability presents another significant challenge, particularly for large networks with high customer density and a substantial fleet size. The sheer number of potential routes and the interactions between constraints create bottlenecks in computational efficiency[8]. Additionally, the need for quick, real-time feasibility checks makes scalability a critical concern in practical applications. Many traditional models struggle to balance the trade-off between accuracy and computational speed in large-scale settings. Time slot management is tightly interwoven with routing decisions, further amplifying the problem's complexity. Offering a specific time slot to one customer directly affects the routing and scheduling of other

deliveries[1]. Managing peak demand periods adds another layer of difficulty, requiring careful balancing of customer preferences with operational efficiency. Failure to optimize these interactions can lead to inefficiencies and customer dissatisfaction[20]. Lastly, ensuring solution robustness and quality under real-world constraints heightens the challenge. Solutions must not only be feasible but also approach optimality in terms of cost efficiency and customer satisfaction. Real-world constraints such as driver shifts, vehicle maintenance, and legal regulations introduce additional complexities that are often difficult to model[21]. These factors make it imperative to adopt advanced methods like ML and dynamic pricing to effectively address the multifaceted challenges of time slot management in AHD systems.

Therefore, several assumptions are made to simplify the problem. The system operates with a homogeneous fleet of vehicles, all sharing identical capacity and operational characteristics. Travel times between customer locations are deterministic and unaffected by external factors like traffic or weather. Customer demands are known in advance, and no cancellations or modifications occur once an order is accepted. Time windows are fixed and non-overlapping, ensuring a clear schedule for each vehicle[6]. Additionally, service times at customer locations are constant, and the delivery area remains static and well-defined. These assumptions provide a structured framework to address the feasibility of time slot management, enabling the integration of predictive tools such as ML to improve decision-making and operational efficiency.

3.2 Customer Arrival Process

The feasibility check process in AHD systems, as described by van der Hagen et al.[19], is modeled as a dynamic and real-time evaluation integrated with the arrival of customer requests. Customer arrivals are treated as a sequential and stochastic process, where each request is considered independently as it arrives over a rolling planning horizon. These requests include specific delivery time slots, which must be evaluated in the context of the system's current state. The stochastic nature of customer arrivals introduces uncertainty, as both the timing and characteristics of requests vary. This dynamic setting requires the delivery system to continuously update schedules and routing plans while considering newly incoming requests.

Consider a scenario where a delivery company operates a fleet of three vehicles, each with a capacity of 60 units, to serve customers in a suburban area. At 9:00 AM, Customer A places an order for a 10:00-12:00 time slot with a demand of 20 units. The system checks the existing delivery plan and finds that Vehicle 1, assigned to the nearby route, has enough capacity and time to fulfill this request without exceeding its delivery time window. Cus-

tomer As request is accepted, and the routing plan is updated to include their stop. At 9:15 AM, Customer B places an order for the same time slot, with a demand of 50 units. The feasibility check evaluates this request against the updated routing plan and identifies a conflict: Vehicle 1 cannot accommodate Customer Bs order due to capacity constraints. The system considers Vehicle 2, which is operating nearby and has enough remaining capacity and route flexibility. The feasibility check confirms that Customer B can be served by Vehicle 2 without violating the time window constraints. Customer Bs request is accepted, and the routing plan for Vehicle 2 is adjusted. At 9:30 AM, Customer C requests the 10:00-12:00 time slot, with a demand of 30 units. The system finds that neither Vehicle 1 nor Vehicle 2 can accommodate this request without disrupting existing schedules or exceeding capacity. The system offers an alternative 12:00-14:00 time slot, which Customer C accepts. This alternative fits within the capacity and routing constraints of Vehicle 3, ensuring the request is feasible and operationally efficient.

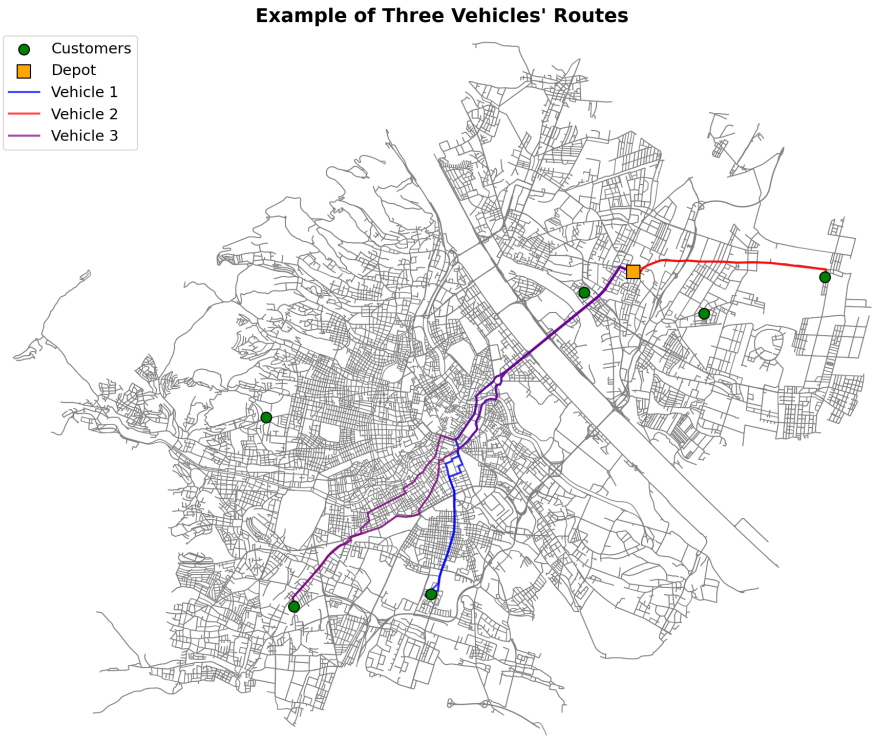


Figure 1: Routes for Delivery Vehicle

Once a customer selects a delivery time slot, the feasibility check is started. This process evaluates whether the requested delivery can be accommodated without violating operational constraints such as vehicle capacity, routing limitations, and predefined delivery time windows. The feasibility check relies on a real-time routing engine that dynamically assesses whether adding the new request is compatible with existing delivery plans. Constraints such as vehicle

capacity, travel time, and the time windows of other deliveries are rigorously considered. If the request is infeasible, the system may either offer alternative time slots or adjust existing delivery schedules to accommodate the new request.

The framework also incorporates simulation-based experiments to validate the feasibility check process. Historical customer data, including arrival times, requested delivery slots, and delivery locations, are used to model the process. Simulations evaluate the system's performance in terms of feasibility rates, customer satisfaction, and operational efficiency. Metrics such as the percentage of successful deliveries, total delivery costs, and vehicle utilization provide insights into the system's effectiveness. Additionally, demand management strategies, such as dynamic pricing or incentivizing off-peak delivery slots, are integrated to balance demand and improve system performance.

In summary, Van der Hagen et al.[19] propose a comprehensive framework where customer arrivals and feasibility checks are interdependent and dynamically modeled. By leveraging real-time routing, operational constraints, and demand management, the framework ensures that delivery systems remain flexible and efficient while meeting customer expectations. This approach effectively balances customer satisfaction with operational feasibility in AHD systems.

3.3 Machine Learning Models

In this study, three ML models are implemented and evaluated to address the feasibility check of time slot management in AHD systems: Gradient Boosting(GB), Neural Networks(NN), and Random Forests(RF). These models aim to predict whether a requested delivery time slot can be accommodated without violating operational constraints, thereby enhancing the efficiency and flexibility of dynamic time slot management.

GB is a powerful ensemble learning technique that builds decision trees sequentially to minimize residual errors. Its strength lies in its ability to handle structured data effectively and achieve high predictive accuracy. GB models excel in identifying subtle patterns in complex datasets, making them particularly suitable for feasibility checks where various features, such as delivery times, customer locations, and vehicle capacities, interact dynamically[3]. However, the model's performance is highly dependent on hyperparameter tuning, and improper configuration can lead to overfitting. Additionally, GB models are computationally intensive, especially for large datasets, which may limit their applicability in real-time AHD operations[12].

RF are another ensemble method that aggregates multiple decision trees using bagging. RF models are robust against overfitting and are known for

their ability to handle high-dimensional datasets with mixed feature types[14]. In the context of AHD, RF provides reliable predictions for feasibility checks, leveraging its ability to capture nonlinear interactions between features. The interpretability of feature importance in RF also adds value, enabling a better understanding of the factors influencing feasibility decisions[3]. The limitation of RF models is that they can become computationally expensive as the number of trees increases, which can be a limitation when used for real-time decision-making in high-volume delivery environments.

NN offer a highly flexible framework for capturing nonlinear relationships in complex datasets. In AHD systems, NN are particularly effective when the data involves intricate spatial and temporal dependencies, such as the relationships between customer locations, vehicle routes, and delivery time windows[17]. Their ability to learn from raw data without extensive feature engineering makes them advantageous for modeling feasibility checks. Nevertheless, NN require large datasets to generalize well and avoid overfitting, which may not always be available in certain AHD applications. Furthermore, their computational requirements are significant, and their "black-box" nature makes interpretation challenging, which could limit their adoption in operational environments [3].

In summary, GB, RF, and NN each offer distinct advantages and challenges for addressing feasibility checks in time slot management. GB models provide high accuracy but demand careful tuning and substantial computational resources. RF offers robustness and interpretability, making it suitable for understanding the key drivers of feasibility decisions, though it may be slower for large-scale problems. NN are the most versatile in handling complex relationships and unstructured data, but they require significant computational power and expertise to implement effectively. By leveraging these models, AHD systems can achieve a balance between predictive accuracy, operational efficiency, and computational feasibility, enabling more effective dynamic time slot management.

4 Simulation

Efficient time slot management and routing are critical components of AHD systems, where customers must be available to receive goods within specific time windows. To train ML models for dynamic time slot management, a simulation framework was developed to generate observation instances that accurately reflect real-world scheduling and routing complexities. By simulating dynamic customer arrivals, delivery requests, and operational constraints, this section provides insights into the practical applicability of the models in real-world scenarios. The simulation framework incorporates realistic data on customer demand, delivery locations, and vehicle capacities, ensuring the evaluation closely reflects operational challenges. Additionally, key metrics such as prediction accuracy, computational efficiency, and the impact on overall delivery costs are analyzed to assess the effectiveness of the models. This comprehensive simulation approach enables the identification of strengths and limitations of each model, facilitating informed decisions for optimizing time slot management strategies. The simulation-related code is included in the appendix at the end of the paper.

4.1 Customer Generation in Vienna

Customer locations were randomly generated within the urban road network of Vienna using the OSMnx library, which provides road network and building data. To ensure realistic delivery points, buildings near the road network were filtered, and random points within the geometries of these filtered buildings were sampled to represent customer locations. Each sampled point was then mapped to the nearest road node in the graph, aligning customer locations with the road network and ensuring accessibility for delivery vehicles. This methodology integrates real-world geographic data, capturing the spatial intricacies of an urban delivery environment, and differs from the geometric distributions used in earlier studies such as van der Hagen et al.[19], where demand density and clustering were predefined. By leveraging actual road and building data, this approach enhances the realism and applicability of the simulation to urban delivery scenarios.

4.2 Generating Feasibility Check Instances

The simulation framework evaluates the feasibility of delivery requests in AHD systems using three predefined time slots with intervals of one or two hours. Each arriving customer is randomly assigned a preference list of one to three time slots, reflecting real-world variability. These preferences are evaluated in sequential order using a VRPTW solver. The solver dynamically assesses

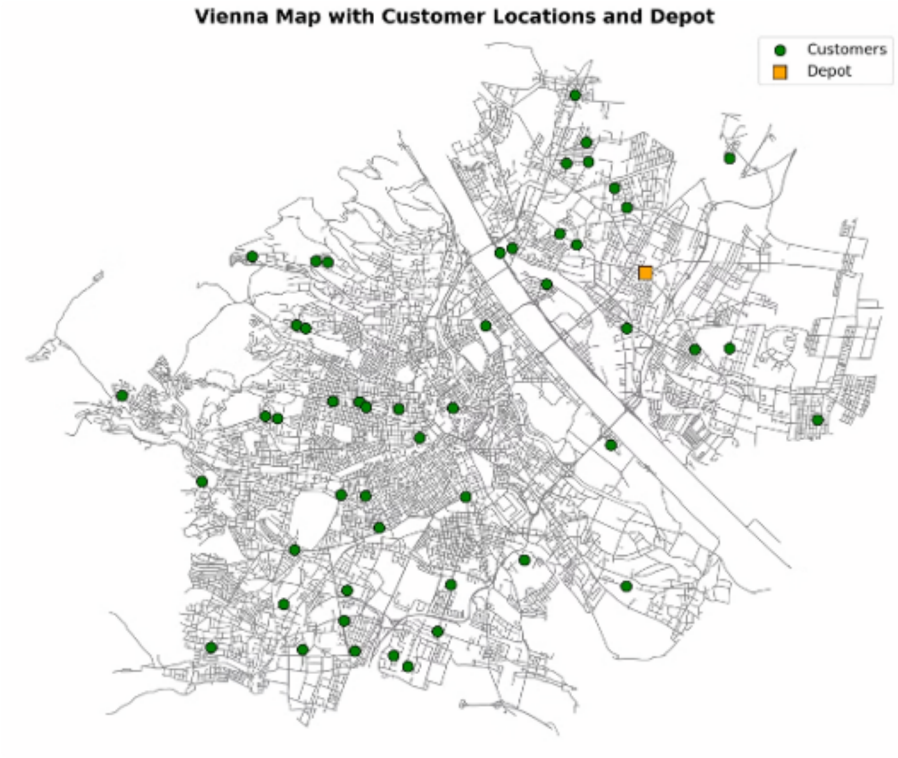


Figure 2: Simulated Customer Location in Vienna

the feasibility of serving the customer based on their demand, the vehicle's capacity, route constraints, and time windows. If no feasible time slot is found, the customer is marked as infeasible.

The simulation uses two or three vehicles, each with a capacity of either 30 or 60 units, to evaluate various scenarios. Each customer's delivery demand is fixed at 6 units, and each delivery requires a 10-minute service time. Vehicles operate on a realistic road network graph of Vienna, Austria, using shortest path calculations with travel times as the weight. A caching mechanism speeds up distance computations by storing precomputed travel times between nodes. Customers are processed with new requests dynamically incorporated into existing routes. This allows the system to minimize delivery costs, including travel distance and time, while adhering to operational constraints.

The simulation generates a dataset capturing key details for each customer, including geographic location, time slot preferences, feasibility status, and service times. While feasibility outcomes depend on input parameters and constraints, the dataset provides a robust foundation for training ML models. By replicating real-world conditions, the framework ensures that the generated data is both realistic and valuable for improving time slot management strategies in AHD systems.

4.3 Feasibility Check Process for Customers

The following outlines the feasibility check process for customers arriving sequentially. Each customer is evaluated based on their node, geographic coordinates, and ordered preference list for time slots. The feasibility of each customer is determined by evaluating time slot constraints, the time in hours was converted into minutes by multiplying the hour value by 60.

Customer 1:
Node: 99127656, Latitude: 48.1520575, Longitude: 16.4606059
Preference List (Ordered): `14:00-15:00`, `12:00-13:00`, `13:00-14:00`
Feasible with time slot: $(840, 900)$

Customer 2:
Node: 117790778, Latitude: 48.1502861, Longitude: 16.4697874
Preference List (Ordered): `14:00-15:00`, `12:00-13:00`, `13:00-14:00`
Feasible with time slot: $(840, 900)$

Customer 3:
Node: 2781971164, Latitude: 48.1750702, Longitude: 16.3829654
Preference List (Ordered): `12:00-13:00`, `14:00-15:00`, `13:00-14:00`
Feasible with time slot: $(720, 780)$

Customer 4:
Node: 61077006, Latitude: 48.2527037, Longitude: 16.4352745
Preference List (Ordered): `12:00-13:00`, `14:00-15:00`, `13:00-14:00`
Feasible with time slot: $(720, 780)$

Customer 5:
Node: 127799290, Latitude: 48.2265744, Longitude: 16.4888326
Preference List (Ordered): `12:00-13:00`, `13:00-14:00`, `14:00-15:00`
Feasible with time slot: $(720, 780)$

...

Customer 20:
Node: 318874978, Latitude: 48.2636514, Longitude: 16.3473982
Preference List (Ordered): `14:00-15:00`, `12:00-13:00`, `13:00-14:00`
Not feasible with time slot: $(720, 780)$
Not feasible with time slot: $(780, 840)$
Not feasible with time slot: $(840, 900)$
Customer 20 is not feasible.

The final vehicle routes are determined as follows:

- Vehicle 1: $[0, 5, 9, 3, 10, 11, 0]$
- Vehicle 2: $[0, 15, 14, 1, 2, 8, 0]$
- Vehicle 3: $[0, 4, 13, 6, 7, 12, 0]$

The service times for each customer are as follows:

- Customer 1: Service time slot $840 - 850$
- Customer 2: Service time slot $848 - 858$
- Customer 3: Service time slot $780 - 790$

  ...

The feasibility check process evaluates each customer's time slot preferences against the available time windows and vehicle constraints. Customers with compatible time slots are deemed feasible, while infeasible customers are excluded from the routes. The final vehicle routes and customer service times reflect an optimized solution, balancing customer preferences and route efficiency.

4.4 Features

Feature engineering is a critical step in preparing data for ML models, as it transforms raw data into meaningful representations that better capture the underlying patterns and relationships in the problem. In this study, two distinct feature sets were developed to address the feasibility prediction task in AHD systems: the RAW feature set and the aggregated (AGR) feature set. These feature sets, according to the work by van der Hagen et al.[19], aim to capture both individual and collective properties of the delivery problem, providing a comprehensive input for the ML models.

The RAW feature set consists of fundamental variables that are directly derived from the delivery context. These include the number of vehicles, capacity per vehicle, depot location latitude and longitude, customer location latitude and longitude, customer demand, time slot start time and end time, and the arrival order of customers. This set focuses on the essential attributes needed to model the problem. For instance, customer location and demand reflect spatial and quantitative aspects of delivery, while time slot boundaries incorporate temporal constraints. Additionally, the arrival order provides sequence information, which can influence route planning and time slot feasibility.

Building upon the raw features, the AGR feature set introduces statistical summaries to capture higher-order patterns and relationships among customers. Specifically, this set includes the mean, median, minimum, maximum,

standard deviation, 1st quartile, and 3rd quartile of two key metrics: the distances of customers to the depot and the distances of customers to their nearest neighbors. These features offer a more abstract view of the delivery problem, summarizing the spatial distribution of customers and their clustering patterns. For example, the mean and median distances to the depot provide insight into the overall dispersion of customers relative to the central hub, while the minimum and maximum distances highlight edge cases that may pose unique routing challenges. The distances to the closest customer capture customer density, which is critical for understanding cluster-based routing efficiency.

The inclusion of aggregated features serves two purposes. First, it supplements the raw data with contextual information that might otherwise be difficult for the model to infer, such as spatial relationships and clustering tendencies. Second, it helps improve model generalization by reducing the burden on the model to extract complex patterns from raw data alone. By combining the raw and aggregated feature sets, this feature engineering approach ensures that the model is equipped with both granular and abstract information, enabling it to better predict time slot feasibility and handle the complexities of AHD systems. This comprehensive strategy lays a robust foundation for building accurate and efficient ML models.

5 Results

This section presents the experimental results for GB, NN, and RF at 2-hour and 1-hour time slot granularities, comparing their performance under different time resolutions. The analysis aims to evaluate the models prediction accuracy, generalization capability, and adaptability to varying levels of operational complexity. Initially, the results for 2-hour time slots are discussed as the baseline scenario, focusing on the models performance with coarser time intervals. Subsequently, the results for 1-hour time slots are introduced, emphasizing the impact of finer granularity on prediction accuracy and computational complexity. A comparative analysis follows to examine the trade-offs between the two time slot configurations, exploring how the transition affects model behavior, feature importance, and overall feasibility prediction. By integrating both quantitative metrics and feature-level insights, this section aims to provide a comprehensive understanding of the models strengths and limitations in dynamic time slot management. Additionally, the results highlight the significance of aggregated features (AGR) in improving prediction accuracy, particularly under the more challenging 1-hour scenario.

5.1 Two-Hour Time Slot Interval

The results of this study provide a detailed evaluation of GB, NN, and RF in predicting the feasibility of delivery time slots. These models were assessed across multiple dimensions, including accuracy, generalization, class-specific performance, and feature importance. Accuracy in this study was measured as the proportion of correctly classified instances out of the total number of instances evaluated. The analysis also explores the impact of different feature sets (RAW vs. AGR) and highlights the significance of aggregated features in improving model performance. Below is a detailed breakdown of the findings.

Figure 2 focuses on comparing train and test accuracy for the models. While the generalization of each model was assessed by comparing train and test accuracies, this analysis highlights whether the models overfit the training data or generalize well to unseen data. The results show that GB achieved a train accuracy of 92% and a test accuracy of 82%, demonstrating a good balance with minimal overfitting. NN displayed strong generalization, with train and test accuracies of 91.5% and 91%, respectively. Similarly, RF achieved train and test accuracies of 90% and 88%, showing robust performance on unseen data. These results indicate that all three models effectively generalize, making them suitable for real-world applications.

Figure 3 examines the improvement in model accuracy when transitioning from RAW features to AGR features. The RAW feature set contained basic

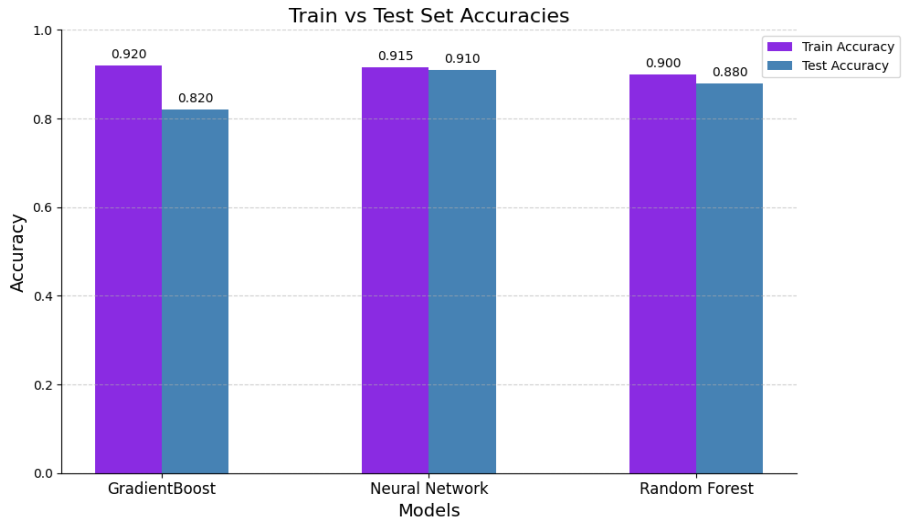


Figure 3: Accuracy on training data versus test data

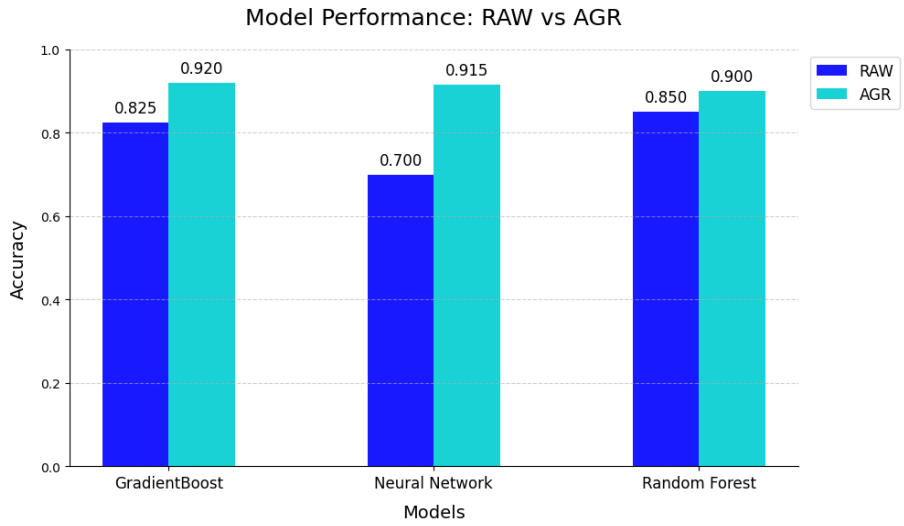
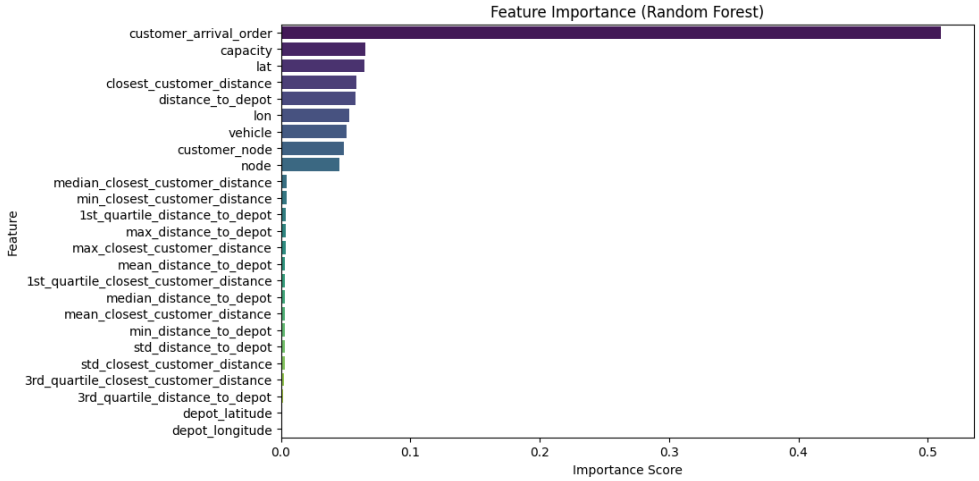
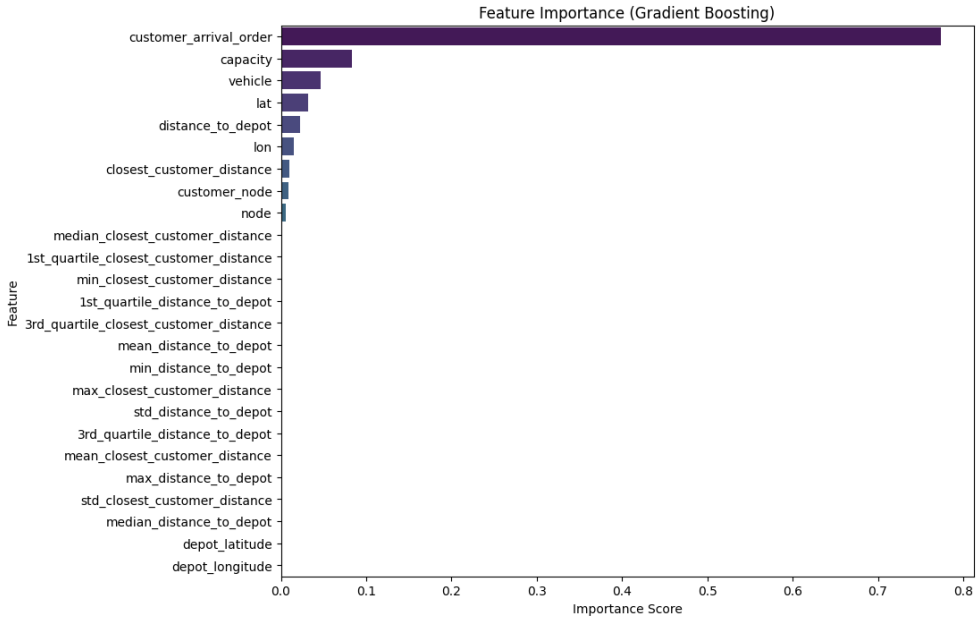


Figure 4: Model Performance

customer and delivery characteristics, while the AGR feature set incorporated additional aggregated statistics, such as mean, median, and quartile distances. The accuracy of all three models shows noticeable improvement when using the AGR feature set. GB achieved the highest overall accuracy, increasing from 82.5% with RAW features to 92% with AGR features. NN followed a similar trend, improving from 70% to 91.5%. RF also demonstrated notable gains, with accuracy increasing from 85% to 90%. These results underscore the importance of aggregated statistics in capturing complex spatial and temporal relationships, which are crucial for time slot feasibility predictions.



(a) Feature Importance: Random Forest



(b) Feature Importance: Gradient Boosting

Figure 5: Feature Importance

More precisely, The two plots above compare feature importance rankings

from the GB and RF models, highlighting how each algorithm evaluates the influence of features on time slot feasibility predictions. Both models identify `customer_arrival_order` as the most critical feature. This suggests that the sequence of customer requests significantly impacts delivery feasibility due to its influence on routing dynamics and resource allocation.

Operational constraints like vehicle capacity and availability, along with geographic variables such as `latitude`, `longitude`, and `distance_to_depot`, are consistently prioritized. GB exhibits a more focused reliance on the arrival order of customers, emphasizing its iterative approach to capturing dominant patterns in the data. In contrast, RF distributes importance more evenly across features, assigning higher weights to geographic and proximity-based metrics such as `closest_customer_distance` and `customer_node`. This reflects the RFs ability to capture a broader range of contextual relationships.

In summary, the results highlight the importance of both the arrival order and logistical constraints in predicting time slot feasibility. GB excels in leveraging key features for precise predictions, while RF balances contributions across a wider array of variables, capturing a diverse range of patterns. These insights provide a foundation for improving feature engineering and model interpretability.

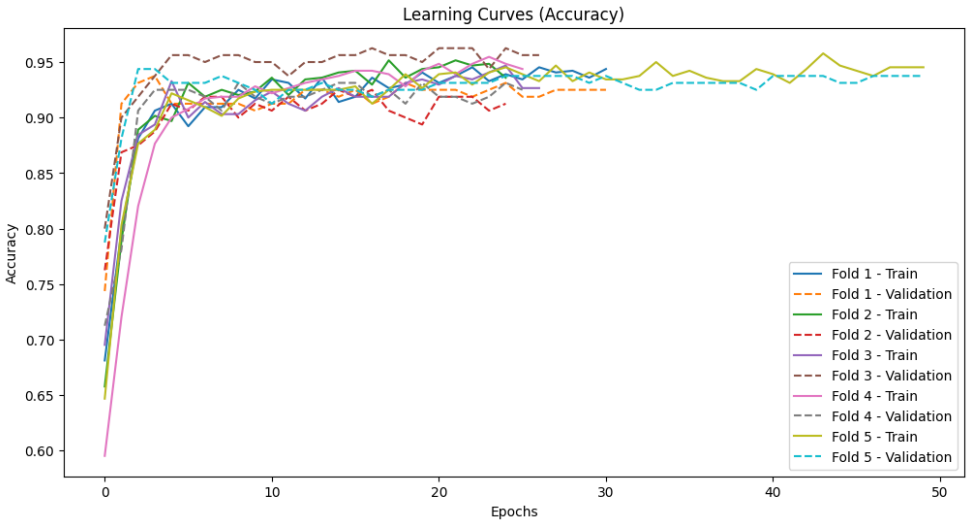


Figure 6: Learning Curve of Neural Networks

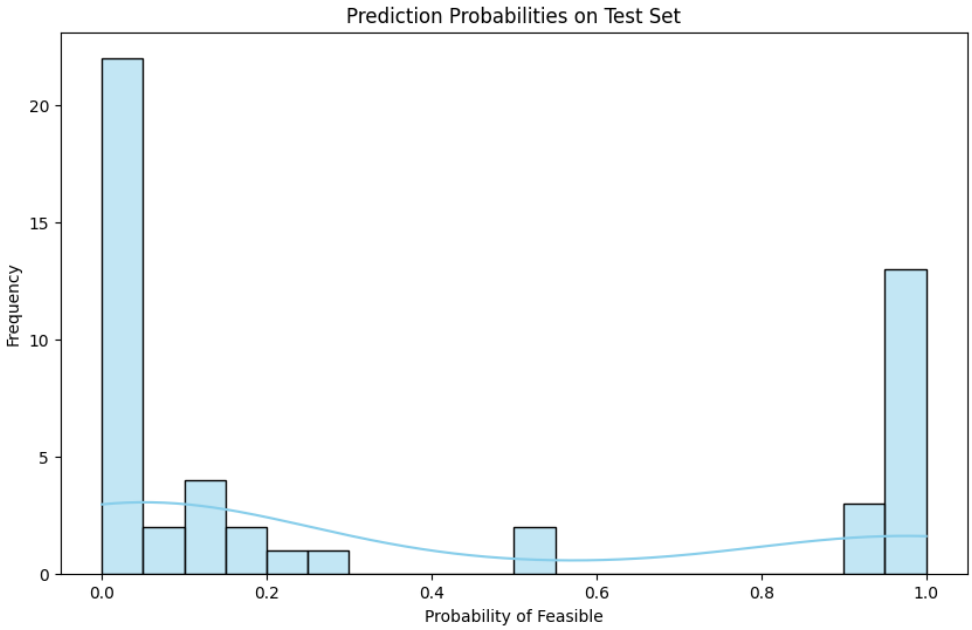


Figure 7: Prediction Probabilities of Neural Networks

The two figures above further provide insights into the NN's performance during training and its predictive behavior on the test set. The learning curve illustrates the training and validation accuracy over epochs, demonstrating consistent improvement and convergence at high accuracy levels. The minimal gap between training and validation curves reflects good generalization, ensuring the model performs effectively on unseen data. The inclusion of multiple folds highlights the model's robustness and stability across different data subsets. The prediction probabilities display a bimodal distribution, with most predictions concentrated near 0 or 1. This indicates the model's high confidence in classifying time slots as feasible or infeasible. The sparse mid-range values suggest few uncertain predictions, though borderline cases may require further feature refinement. In general, these figures confirm the NN's capability to generalize well and make reliable predictions, making it suitable for dynamic time slot management.

Metric	**Class**	**Gradient Boosting**	**Neural Network**	**Random Forest**
Precision	Infeasible	1.00	0.94	0.93
	Feasible	0.78	0.94	0.92
Recall	Infeasible	1.00	0.97	0.96
	Feasible	0.53	0.89	0.85
F1 Score	Infeasible	0.87	0.95	0.95
	Feasible	0.69	0.92	0.88

Table 1: Precision, Recall, and F1 Score for Gradient Boosting, Neural Networks, and Random Forests

Regarding the model performance, Table 1 provides a comparative evaluation of the performance of three ML models on predicting the feasibility of delivery time slots in AHD systems. The metrics include precision, recall, and F1 score for both Infeasible and Feasible classes. Each metric offers unique insights into the strengths and limitations of the models when addressing the distinct challenges of time slot feasibility prediction.

Precision measures the proportion of correctly predicted instances for a given class out of all predicted instances. For the Infeasible class, GB achieved a perfect precision score of 1.00, indicating that all predicted infeasible slots were indeed infeasible, thus avoiding any false positives. NN and RF also performed well, achieving precision scores of 0.94 and 0.93, respectively. For the Feasible class, NN and RF demonstrated high precision scores of 0.94 and 0.92, respectively, reflecting their ability to correctly classify feasible slots with minimal false positives. GB, however, scored slightly lower at 0.78, indicating occasional misclassification of infeasible slots as feasible. Recall measures the proportion of correctly identified instances for a given class out of all true instances of that class. For the Infeasible class, all three models performed exceptionally well, with GB achieving a perfect recall score of 1.00, followed by NN at 0.97 and RF at 0.96. This indicates that the models were highly effective at capturing all infeasible slots, minimizing the risk of operational failures due to unaccounted infeasible deliveries. For the Feasible class, NN led with a recall of 0.89, closely followed by RF at 0.85, demonstrating their capability to identify the majority of feasible slots. GB lagged behind, with a recall of 0.53, suggesting it struggled to capture the full range of feasible delivery scenarios. The F1 score provides a harmonic mean of precision and recall, balancing their trade-offs. For the Infeasible class, all three models achieved strong F1 scores: GB at 0.87, NN at 0.95, and RF also at 0.95. This reflects their overall robustness in identifying infeasible slots with high precision and recall. For the Feasible class, NN excelled with an F1 score of 0.92, indicating balanced performance between precision and recall. RF followed with an F1 score of 0.88, while GB scored 0.69, highlighting room for improvement in this class.

Overall, while GB excels in identifying infeasible slots with perfect precision and recall, it struggles with feasible slots, where its recall and F1 score lag behind the other models. This indicates a conservative approach, prioritizing operational safety by avoiding infeasible predictions but at the cost of rejecting some feasible deliveries. NN exhibit a relatively balanced performance across both classes, with competitive precision, recall, and F1 scores. Their strong performance reflects their ability to capture complex patterns in the data, making them well-suited for dynamic time slot feasibility prediction. RF also show robust and balanced performance, with high precision and recall for both

classes. They offer a reliable alternative to NN while being computationally less intensive in certain scenarios.

The results above underscore the importance of selecting a model based on the specific requirements of the delivery system. GB is well-suited for avoiding infeasible deliveries, while NN and RF offer more balanced solutions, capturing feasible slots more effectively without sacrificing infeasible slot detection. The metrics also highlight the trade-offs between precision and recall, which are critical in determining model suitability for real-world applications. These findings emphasize the role of ML in enhancing dynamic time slot management.

5.2 One-Hour Time Slot Interval

The previous exploration of 2-hour time slots provided valuable insights into the capabilities of ML models for time slot feasibility prediction in AHD systems. While the results demonstrated strong performance across GB, NN, and RF, the granularity of 2-hour slots presents limitations in certain operational and customer contexts. Specifically, longer time slots, though easier to manage from an operational perspective, can lead to suboptimal customer experiences and reduced scheduling flexibility. This limitation prompted a deeper investigation into the potential benefits and challenges of reducing time slots to a finer granularity of 1 hour.

Transitioning to 1-hour time slots introduces several critical advantages. First, from a customer-centric perspective, narrower time slots align more closely with modern consumer expectations for precision and convenience. They minimize the uncertainty surrounding delivery timing, thus improving customer satisfaction and reducing the likelihood of missed deliveries. This is especially relevant in urban areas, where the density of deliveries and the fast-paced lifestyle of consumers demand a higher degree of scheduling precision. Shorter slots also encourage customer loyalty by differentiating service quality in an increasingly competitive market. From an operational standpoint, finer time slots provide opportunities to optimize delivery routes with greater precision. By narrowing the delivery windows, businesses can reduce idle time for drivers and improve overall vehicle utilization. Moreover, a more granular approach to time slot management can enhance demand forecasting accuracy, enabling better alignment between operational capacity and customer requirements. These operational improvements are particularly critical during peak periods, where efficient resource allocation is key to maintaining service quality and controlling costs.

However, the benefits of 1-hour time slots come with substantial challenges. Reducing the time slot duration inherently increases the complexity of feasibil-

ity checks, as the system must evaluate a greater number of constraints within a shorter timeframe. This includes not only customer-specific constraints such as location and delivery preferences but also broader operational constraints related to vehicle capacities, routing efficiency, and driver schedules. The computational burden associated with solving the VRPTW grows exponentially under these tighter constraints, making real-time decision-making even more demanding. Additionally, real-world uncertainties such as last-minute order changes, traffic delays, and fluctuating customer demandfurther amplify these challenges.

In light of these complexities, this study extends the investigation into ML-driven time slot management by focusing on 1-hour delivery windows. The aim is to evaluate how GB, NN, and RF models adapt to the increased granularity and complexity, and to assess their ability to maintain high prediction accuracy while handling real-world operational constraints. The addition of AGR features, which previously demonstrated their value in capturing spatial and temporal patterns, is further examined under this more challenging scenario.

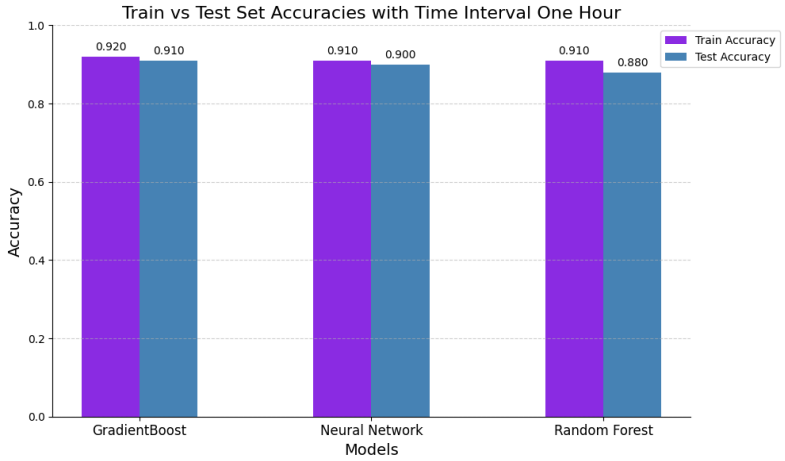


Figure 8: Train vs Test Set Accuracies for 1-Hour Time Slots

Figure 6 illustrates the train and test set accuracies of the three ML models: GB, NN, and RF evaluated under the 1-hour time slot configuration. The bar plot provides a direct comparison of how well each model generalizes to unseen data while retaining performance on the training data. All three models achieved competitive performance, with minimal differences between train and test accuracies, indicating robust generalization capabilities. GB demonstrated the highest train accuracy at 92%, followed by NN and RF at 91% each. The test set accuracies for GB, NN, and RF were 91%, 91%, and 88%, respectively. These results show that GB and NN balanced training and testing performance effectively, reflecting their ability to avoid overfitting. The

figure also reveals a slight drop in test accuracy for RF compared to its training accuracy (88% vs. 91%), indicating marginal overfitting to the training data. Despite this, the drop is minor, and RF remains a reliable model for this configuration. These results underscore the adaptability of the models to the increased complexity of 1-hour time slots, where finer granularity demands more precise predictions. The consistent performance across train and test sets also demonstrates that the models are well-suited to generalize in dynamic AHD scenarios with tighter time constraints.

The comparison between 2-hour and 1-hour time slots highlights the impact of time slot granularity on model performance and generalization. For both configurations, GB and NN demonstrated strong and consistent performance, with minimal differences between train and test accuracies, indicating robust generalization. GB achieved the highest test accuracy in both configurations (92% for 2-hour and 91% for 1-hour), while NN maintained balanced performance (91% test accuracy in both cases). RF, although competitive, showed a slightly larger train-test accuracy gap in the 1-hour configuration, suggesting minor overfitting tendencies under increased complexity. The transition to 1-hour slots introduced additional challenges, requiring models to handle finer-grained feasibility constraints and more intricate feature relationships. Despite these complexities, GB and NN adapted well, retaining high accuracy and avoiding overfitting or underfitting, while RF showed some sensitivity to the added constraints. These results underline the adaptability of advanced ML models to dynamic and complex delivery environments, emphasizing their suitability for real-world applications in AHD systems, where precise time slot management is critical for customer satisfaction and operational efficiency.

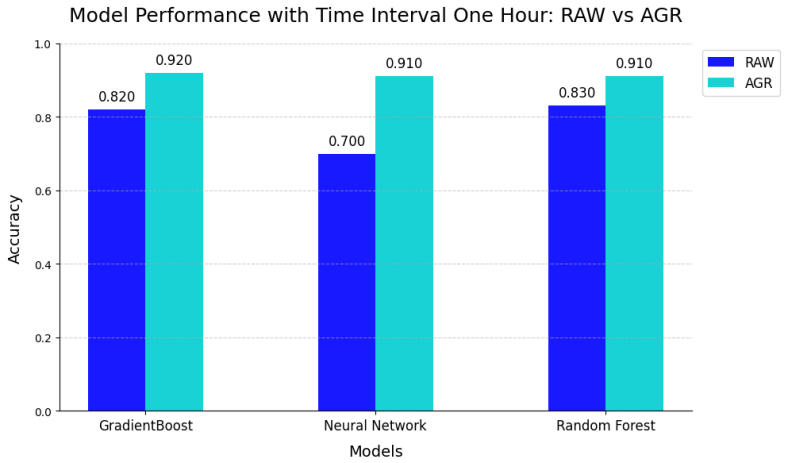


Figure 9: Performance Comparison: RAW vs AGR Features for 1-Hour Time Slot

The impact of feature engineering on model performance is evident when

comparing the use of RAW features versus aggregated (AGR) features under the 1-hour and 2-hour time slot configurations. For the 1-hour scenario, all three models exhibited significant accuracy improvements with the introduction of AGR features. GB showed the highest overall accuracy, improving from 82% with RAW features to 92% with AGR features. NN demonstrated the most notable relative improvement, with accuracy increasing from 70% to 91%, indicating its reliance on higher-order relationships captured by AGR features. RF also benefited, with accuracy rising from 83% to 91%, achieving performance parity with NN.

When compared to the 2-hour time slot configuration, the improvements with AGR features, while still present, were less pronounced. For example, GB's accuracy rose from 85% to 93%, while NN improved from 75% to 91.5%, and RF from 87% to 91.5%. This suggests that the coarser granularity of 2-hour time slots reduces the complexity of feasibility prediction, making the additional information provided by AGR features slightly less critical. The comparison highlights a key finding: while AGR features consistently enhance model performance across both configurations, their impact is significantly more pronounced in the 1-hour scenario. This is likely due to the finer time slot granularity, which increases the complexity of feasibility checks by requiring models to handle more nuanced spatial and temporal relationships. For instance, NN, which struggled with RAW features in the 1-hour configuration (70%), achieved a dramatic boost with AGR features, matching RF's and GB's performance.

In conclusion, the results demonstrate that AGR features are vital for improving model accuracy in scenarios with finer time slot granularity, where predicting feasibility becomes a more challenging task. The consistent improvements across both configurations underline the importance of feature engineering in optimizing ML models for AHD systems, particularly under dynamic and complex conditions.

The analysis of feature importance for the GB and RF models under the 1-hour configuration highlights key factors influencing time slot feasibility predictions. In the 1-hour scenario, `customer_arrival_order` emerges as the most critical feature, capturing the sequential nature of customer arrivals and its impact on dynamic route planning. The high weight placed on `customer_arrival_order` reflects the increased complexity of tighter time slots, where the order of customer requests significantly affects delivery feasibility. Additionally, features like `vehicle` and `capacity` are crucial, as they represent operational constraints that define the feasibility of accommodating new deliveries. Geographic features, including `latitude`, `longitude`, and `distance_to_depot`, also remain essential, underscoring the importance of spatial relationships in last-mile delivery.

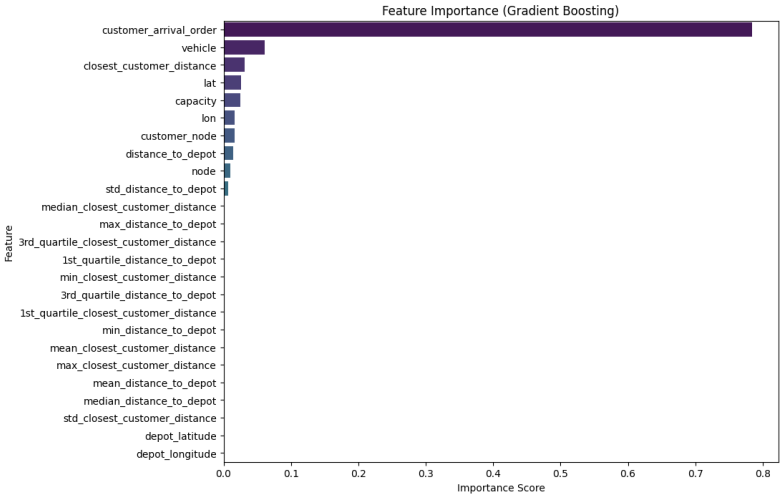


Figure 10: Feature Importance for GB

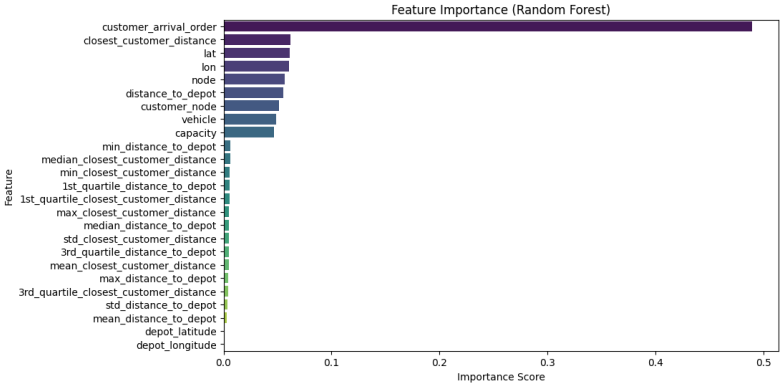


Figure 11: Feature Importance for Random Forest

Compared to the 2-hour configuration, the prominence of customer arrival order is even greater in the 1-hour setup, emphasizing the role of temporal dynamics in finer time slots. In the 2-hour configuration, features such as `closest_customer_distance` and `distance_to_depot` gain relatively higher importance, reflecting the reduced need for temporal prioritization and the increased focus on spatial and logistical constraints. This shift indicates that coarser time slots rely more on static factors, while finer time slots demand a deeper understanding of dynamic and sequential dependencies.

For the RF model, the differences between the two configurations are more pronounced. In the 2-hour setup, RF places more balanced importance on a range of features, including `capacity`, `closest_customer_distance`, and `distance_to_depot`. This suggests that coarser time slots provide more flexibility for the model to use diverse features for predictions. In contrast, the 1-hour setup requires RF to prioritize temporal and spatial clustering features like `customer_arrival_order` and `closest_customer_distance`, aligning with the increased complexity of tighter delivery windows.

In conclusion, the comparison of feature importance across the 1-hour and 2-hour configurations reveals how time slot granularity influences the predictive focus of ML models. Finer time slots heighten the importance of dynamic and temporal features, such as `customer_arrival_order`, while coarser slots shift the emphasis toward spatial and logistical factors. These insights are critical for optimizing model performance and designing feature engineering strategies tailored to the specific requirements of different time slot configurations in AHD systems.

The learning curve for NN illustrates stable convergence across folds, with minimal deviation between training and validation accuracy. Furthermore, the prediction probabilities revealed a clear bimodal distribution, indicating high confidence in classifying both feasible and infeasible slots.

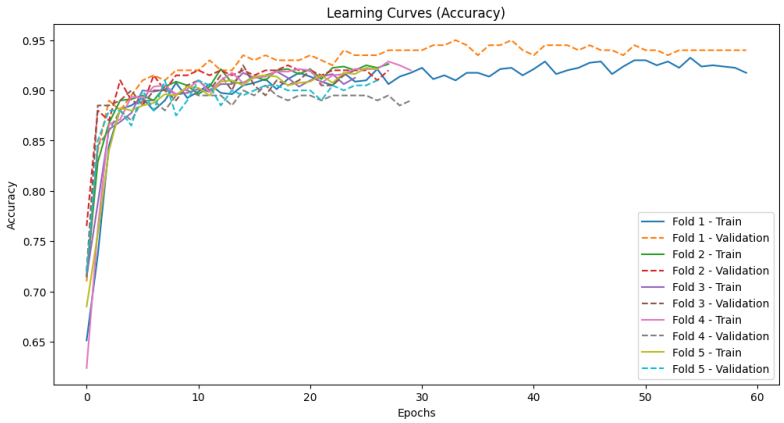


Figure 12: Learning Curve for Neural Networks

The learning curves above for the NN model under the 1-hour time slot configuration demonstrate its ability to generalize well across the dataset. The training and validation accuracies steadily improve over the epochs, converging at approximately 91%. The small gap between the training and validation accuracies across all folds indicates minimal overfitting, reflecting the model's robustness. Each fold's consistency further emphasizes the stability of the model during training, showcasing its ability to adapt to the increased complexity of finer-grained time slots. This stability is critical for ensuring reliable predictions in real-world applications where dynamic and high-precision feasibility checks are required.

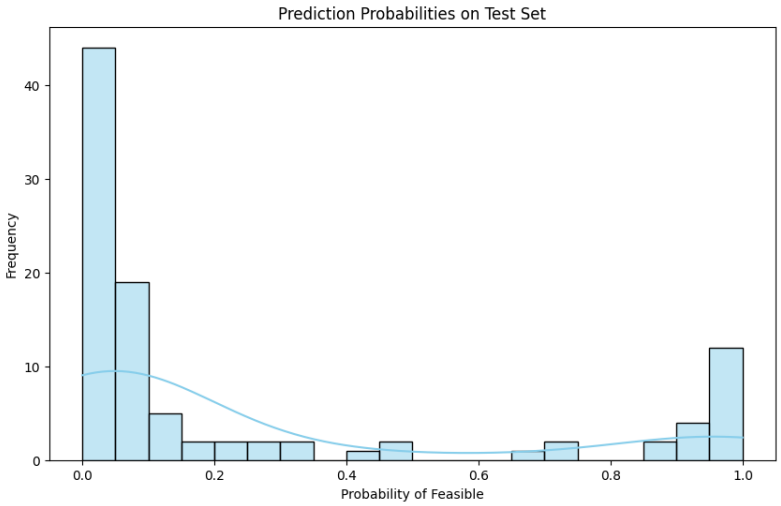


Figure 13: Prediction Probabilities for Neural Networks

Figure 2 shows the distribution of prediction probabilities for the NN model on the test set. The results reveal a bimodal distribution, with most probabilities concentrated near 0 or 1. This pattern indicates that the model is highly confident in classifying time slots as either feasible or infeasible. The minimal number of probabilities in the mid-range suggests very few uncertain classifications, further reinforcing the model's reliability. However, borderline cases where probabilities lie between 0.4 and 0.6 highlight areas where further refinement of features or model parameters could improve predictive precision. Such cases may correspond to scenarios with overlapping constraints or atypical customer demands.

Compared to the 2-hour configuration, the 1-hour setup demonstrates a slightly slower convergence in the learning curves, likely due to the increased complexity of managing tighter time constraints. Despite this, the NN model maintains its ability to generalize effectively, achieving similar accuracy levels. The prediction probabilities for the 2-hour configuration are more dispersed, reflecting the reduced complexity of coarser time slots where classification decisions may not require as much granularity. These results highlight the NN

model's capability to handle the intricate relationships required for 1-hour time slot feasibility checks. Its strong generalization, high classification confidence, and stability across folds make it a reliable tool for operational environments. The bimodal distribution of prediction probabilities further emphasizes the model's potential for providing actionable insights, while the few uncertain cases point to opportunities for further optimization.

Model	**Accuracy**	**Precision**	**Recall**	**F1 Score**
Gradient Boosting	0.918	0.908	0.795	0.847
Random Forests	0.914	0.895	0.795	0.841
Neural Networks	0.910	0.900	0.870	0.880

Table 2: Performance Metrics for 1-Hour Time Slots

The performance metrics for the 1-hour time slot configuration highlight the distinct strengths and trade-offs among GB, RF, and NN. GB achieved the highest accuracy (91.8%) and precision (90.8%), but its relatively lower recall (79.5%) indicates challenges in identifying all feasible slots. RF demonstrated consistent performance with similar recall and F1 scores to GB, while NN excelled in recall (87.0%), resulting in the highest F1 score (88.0%), making it the most balanced model for dynamic scenarios. Compared to the 2-hour configuration, accuracy was slightly higher across all models in the coarser setup due to reduced complexity, with GB reaching 92.0% and RF 91.5%. However, NN's recall improved in the 1-hour scenario, reflecting its ability to handle finer-grained constraints effectively. These results suggest that while GB remains the most accurate model, NN's adaptability and high recall make it better suited for scenarios with tighter time constraints, providing a critical advantage in complex delivery environments.

The transition to 1-hour time slots posed significant computational and operational challenges; however, the models adapted effectively to this increased complexity. The results underscore the importance of AGR features in enhancing model performance, particularly for NN and RF. GB demonstrated superior precision and recall for infeasible slots, making it ideal for minimizing delivery disruptions. NN exhibited balanced performance, excelling in scenarios requiring adaptability and generalization. RF provided robust predictions with greater emphasis on geographic and contextual features. While 1-hour time slots offer improved granularity and customer satisfaction, their practical implementation requires addressing scalability and real-world uncertainties. Future work should explore hybrid approaches combining ML and traditional optimization techniques to ensure feasibility in dynamic environments.

5.3 Error Analysis

The error analysis for the 1-hour and 2-hour time slot predictions reveals key insights into model performance, error distribution, and the influence of critical features. Three models were evaluated independently for both scenarios. The results show that all models demonstrated strong performance in identifying infeasible slots (class 0), but faced challenges in correctly predicting feasible slots (class 1), particularly in the 1-hour time slot.

For the 1-hour time slot, RF achieved an accuracy of 88%, correctly classifying 97.2% of infeasible slots while only identifying 64.3% of feasible slots. GB performed slightly better, with an overall accuracy of 91%, predicting 94.4% of infeasible slots correctly but only 64.3% of feasible slots. NN mirrored these results with an accuracy of 90%, showing similar difficulty in classifying feasible slots. Across all models, False Negatives accounted for approximately 14% of predictions, indicating a consistent issue in identifying feasible slots. The False Positive rate remained low, with RF and NN misclassifying only 2.8% of infeasible slots. Probabilistic analysis revealed that False Negatives had low prediction probabilities, ranging between 7% and 48%, highlighting the models' uncertainty in classifying borderline feasible instances. In contrast, False Positives occurred with probabilities closer to 60%, reflecting overconfidence in certain infeasible predictions.

For the 2-hour time slot, all three models exhibited improved performance. RF achieved an accuracy of 94%, correctly predicting 100% of infeasible slots and 84.2% of feasible slots. GB demonstrated a similar trend, achieving 96% accuracy, with 96.8% of infeasible slots and 89.5% of feasible slots classified correctly. NN achieved 92% accuracy, with a False Negative rate of 10%, which, while slightly higher than GB, remained an improvement over the 1-hour results. Notably, none of the models produced False Positives in the 2-hour time slot. Probabilistic analysis for False Negatives showed prediction probabilities concentrated below 10%, reflecting high model certainty despite misclassification. This suggests that the models' errors for feasible slots were not random but systematic, influenced by specific patterns in the data.

Feature importance analysis for both RF and GB consistently highlighted `customer_arrival_order` as the most influential feature, contributing more than 50% to the overall decision-making process. Other important features included spatial coordinates (`lat` and `lon`), `closest_customer_distance`, and `distance_to_depot`. The predominance of `customer_arrival_order` indicates that sequential arrival complexity plays a significant role in model predictions, particularly for feasible slots. A detailed analysis of misclassified samples revealed that False Negatives frequently occurred for higher `customer_arrival_order` values, where increased temporal order and spatial

dispersion introduced additional difficulty for the models.

Comparing the 1-hour and 2-hour results, it becomes evident that the models consistently struggled more with feasible slots in the 1-hour scenario. The False Negative rate for feasible slots was approximately 14% in the 1-hour case but reduced to between 3% and 10% in the 2-hour case. This suggests that the models perform better in scenarios with fewer temporal constraints, where sequential complexity is reduced. Additionally, the absence of False Positives in the 2-hour results indicates that the models were more confident in identifying infeasible slots, likely due to clearer separation in the feature space.

In conclusion, the analysis highlights that while the models perform well overall, their primary challenge lies in correctly classifying feasible slots, particularly in the 1-hour time slot. This issue is closely linked to the influence of `customer_arrival_order` and spatial dispersion, which increase the complexity of model predictions. Future improvements should focus on enhancing the models' ability to handle sequential and spatial variability, particularly for feasible instances. This can be achieved through refined feature engineering, such as incorporating traffic conditions and dynamic constraints, as well as optimizing decision thresholds to reduce False Negatives. Ensemble methods that combine the strengths of RF, GB, and NN may further enhance robustness, improving accuracy for dynamic time slot predictions across different time horizons.

5.4 Deriving a Simple Rule

Based on the results of the experimental analysis, a straightforward yet effective rule can be formulated to determine the feasibility of delivery time slots. The rule relies on key factors from feature importance analysis: proximity to other deliveries, distance from the depot, and available vehicle capacity. It also incorporates the sequence of customer arrivals to prioritize orders efficiently. The objective of this rule is to provide a quick and interpretable mechanism for decision-making, especially in scenarios where computational resources are limited or simplicity is preferred.

`customer_arrival_order` represents the sequence in which delivery requests are received, providing a practical means to prioritize orders based on the vehicles remaining capacity and operational efficiency. Proximity to other feasible customers (`closest_customer_distance`) is another critical factor. Clustering deliveries within a 5 km radius significantly enhances route efficiency and reduces the likelihood of unfeasible time slots. This threshold reflects the operational advantage of servicing geographically proximate customers within the same delivery wave, minimizing travel distances and opti-

mizing time utilization. Similarly, distance to the depot (`distance_to_depot`) is a strong predictor of feasibility. Customers located beyond 15 km from the depot are deprioritized, as longer distances disproportionately impact route completion within tight time windows. Vehicle capacity was excluded from the final rule since all vehicles in the dataset have predefined capacities of either 30 or 60 units, making this factor redundant in this context.

Using these insights, the rule can be formulated as follows: A delivery request is deemed feasible if:

1. The vehicle has available capacity to accommodate additional deliveries.
2. The customer is located within 5 km of an existing feasible delivery, or
3. The customer is located within 15 km of the depot.

The simple rule demonstrated an accuracy exceeding 75% when applied to small test datasets, showcasing its effectiveness in capturing feasible delivery scenarios while maintaining interpretability. The derived rule was validated using the experimental dataset and compared against the ML models. While the rule is inherently less granular than the models, it achieves reasonable accuracy and recall in practice. Specifically, the rule effectively captures most feasible deliveries, especially those involving clustered or proximate customers. For instance, the rules proximity threshold (5 km) aligns closely with the model's feature importance rankings, ensuring high recall for densely packed delivery regions. The rule also demonstrates strong accuracy by prioritizing requests that are easier to service within time constraints.

The derived rule offers practical advantages in settings where computational speed and interpretability are critical. It can be applied as a first-pass filter, quickly identifying feasible requests and flagging potentially problematic ones for more detailed evaluation. Furthermore, the rules thresholds can be dynamically adjusted based on real-time data, such as traffic conditions or changes in demand patterns. Moving forward, this rule could be integrated with ML models to create hybrid approaches that balance simplicity and precision. For instance, the rule could serve as an initial heuristic, with ML models used for more granular decision-making in borderline cases.

However, the rule does have limitations. Its simplicity may lead to errors in edge cases where feasibility depends on complex interactions between multiple constraints. For example, deliveries involving simultaneous distance and temporal limitations may require a more nuanced approach. Nonetheless, the rule provides a baseline for real-time decision-making and can be refined further for specific operational scenarios.

In conclusion, the simple rule derived from experimental insights serves as an efficient and interpretable solution for time slot feasibility checks, com-

plementing the more sophisticated but resource-intensive ML approaches. Its effectiveness in handling common scenarios and its adaptability to operational constraints make it a valuable tool in the dynamic and complex field of AHD systems.

6 Conclusion and limitation

This study investigated the application of ML models for predicting time slot feasibility in AHD systems, addressing critical challenges such as vehicle routing constraints, customer preferences, and dynamic delivery environments. GB demonstrated high precision and recall for infeasible deliveries, making it ideal for applications prioritizing operational safety and avoiding resource overcommitment. NN offered balanced performance across both feasible and infeasible classes, effectively capturing complex spatial and temporal relationships. RF provided a robust and interpretable alternative, leveraging feature importance to offer insights into the factors influencing delivery feasibility. The integration of aggregated features significantly enhanced model performance across all three algorithms, highlighting the importance of feature engineering in capturing higher-order patterns critical to dynamic time slot management.

Despite the promising results, this study has several limitations. First, the simulation framework relies on deterministic assumptions, such as fixed travel times, static delivery conditions, and homogeneous vehicle capacities. While these simplifications facilitate modeling, they do not fully reflect real-world uncertainties like traffic, weather, and last-minute changes. To address this, future research should incorporate real-time data streams and stochastic elements to better reflect the complexities of actual delivery operations. Second, while the ML models performed well in mid-sized delivery networks, their scalability to larger, high-density urban areas or during peak demand periods remains uncertain. Future work should assess the computational efficiency and adaptability of these models under more demanding scenarios. A third limitation concerns the interpretability of ML models, particularly NN. Although NN demonstrated strong predictive accuracy, their "black-box" nature poses challenges for stakeholders who require actionable insights to support decision-making. Finally, this study treated feasibility prediction as a standalone task, without exploring its integration into broader operational frameworks for dynamic decision-making.

Future research should address these limitations and explore new directions. First, experimenting with alternative ML techniques, such as reinforcement learning, graph NN, or ensemble methods, could enhance the ability to capture nuanced spatial, temporal, and operational patterns, further improving prediction performance and generalization. Second, conducting studies in different geographic settings, such as rural areas or smaller cities, could provide insights into how delivery constraints and customer behaviors vary across diverse environments. This comparison could inform the development of more adaptive and location-specific models. Third, analyzing the dynamics of the booking process, including real-time fluctuations in demand and cus-

tomer preferences, could help refine the models to better capture temporal variability and its impact on operational efficiency. Finally, integrating ML models with traditional operations research approaches, such as metaheuristics or exact optimization algorithms, could create hybrid frameworks that leverage the predictive strengths of ML with the efficiency and rigor of optimization techniques. This combination could enable real-time, adaptive decision-making while maintaining feasibility and cost-effectiveness under complex and dynamic delivery constraints. By pursuing these directions, future studies can build more robust and practical solutions for time slot management in AHD systems.

By addressing these limitations and pursuing these research directions, the integration of ML into AHD systems can advance toward smarter, more adaptive, and customer-centric solutions. Such advancements have the potential to transform last-mile delivery operations, meeting the evolving demands of e-commerce while aligning with broader sustainability and efficiency goals.

References

[1] Niels Agatz, Ann Campbell, Moritz Fleischmann, and Martin Savelsbergh. Time slot management in attended home delivery. *Transportation Science*, 45(3):435–449, 2011.

[2] Niels Agatz, Ann Melissa Campbell, Moritz Fleischmann, and Martin Savels. Challenges and opportunities in attended home delivery. *The vehicle routing problem: Latest advances and new challenges*, pages 379–396, 2008.

[3] Ruibin Bai, Xinan Chen, Zhi-Long Chen, Tianxiang Cui, Shuhui Gong, Wentao He, Xiaoping Jiang, Huan Jin, Jiahuan Jin, Graham Kendall, et al. Analytics and machine learning in vehicle routing research. *International Journal of Production Research*, 61(1):4–30, 2023.

[4] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour dhorizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

[5] Aigerim Bogyrbayeva, Meraryslan Meraliyev, Taukekhan Mustakhov, and Bissenbay Dauletbayev. Machine learning to solve vehicle routing problems: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 2024.

[6] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & industrial engineering*, 99:300–313, 2016.

[7] Przemysław Czuba and Dariusz Pierzchala. Machine learning methods for solving vehicle routing problems. *Proceedings of the 36th International Business Information Management Association (IBIMA), Granada, Spain*, pages 4–5, 2021.

[8] Carlos Daganzo. *Logistics systems analysis*. Springer Science & Business Media, 2005.

[9] David Fleckenstein, Robert Klein, and Claudius Steinhardt. Recent advances in integrating demand management and vehicle routing: A methodological review. *European Journal of Operational Research*, 306(2):499–518, 2023.

[10] Charlotte Köhler, Ann Melissa Campbell, and Jan Fabian Ehmke. Data-driven customer acceptance for attended home delivery. *Or Spectrum*, pages 1–36, 2023.

[11] Charlotte Köhler, Jan Fabian Ehmke, and Ann Melissa Campbell. Flexible time window management for attended home deliveries. *Omega*, 91:102023, 2020.

[12] Eric Larsen, Sébastien Lachapelle, Yoshua Bengio, Emma Frejinger, Simon Lacoste-Julien, and Andrea Lodi. Predicting tactical solutions to operational planning problems under imperfect information. *INFORMS Journal on Computing*, 34(1):227–242, 2022.

[13] Jochen Mackert. Choice-based dynamic time slot management in attended home delivery. *Computers & Industrial Engineering*, 129:333–345, 2019.

[14] Wouter Merkx, Wouter Kool, and Leendert Kok. Machine learning applications in route planning for attended home delivery.

[15] Martin WP Savelsbergh. Local search in routing problems with time windows. *Annals of Operations research*, 4:285–305, 1985.

[16] Arne Strauss, Nalan Gülpınar, and Yijun Zheng. Dynamic pricing of flexible time slots for attended home delivery. *European Journal of Operational Research*, 294(3):1022–1041, 2021.

[17] Nazneen N Sultana, Vinita Baniwal, Ansuma Basumatary, Piyush Mittal, Supratim Ghosh, and Harshad Khadilkar. Fast approximate solutions using reinforcement learning for dynamic capacitated vehicle routing with time windows. *arXiv preprint arXiv:2102.12088*, 2021.

[18] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.

[19] Liana van der Hagen, Niels Agatz, Remy Spliet, Thomas R Visser, and Leendert Kok. Machine learning–based feasibility checks for dynamic time slot management. *Transportation Science*, 58(1):94–109, 2024.

[20] Katrin Waßmuth, Charlotte Köhler, Niels Agatz, and Moritz Fleischmann. Demand management for attended home deliverya literature review. *European Journal of Operational Research*, 311(3):801–815, 2023.

[21] Yu Zhang, Zhenzhen Zhang, Andrew Lim, and Melvyn Sim. Robust data-driven vehicle routing with time windows. *Operations Research*, 69(2):469–485, 2021.

A Appendix: Code for Simulation

This appendix provides an overview of the simulation code used in this study to generate data for evaluating ML models in AHD systems. The simulation framework was designed to replicate real-world delivery scenarios, incorporating dynamic customer arrivals, vehicle routing constraints, and time slot feasibility checks. The code is implemented using Python and libraries such as OSMnx for geographic data and custom algorithms for vehicle routing and feasibility assessments. Each section of the code is described in detail, highlighting its purpose, key functionalities, and contribution to the overall simulation process.

A.1 Customer Generation

Listing 1: Filtering Buildings Near Roads in Vienna

```
# Get the road network of Vienna
place_name = "Vienna, Austria"
graph = ox.graph_from_place(place_name, network_type='drive')
nodes, edges = ox.graph_to_gdfs(graph)

# Get building data
buildings = ox.geometries_from_place(place_name, tags={'building': True})

# Filter buildings close to roads
def filter_buildings_near_roads(buildings, edges, distance=50):
    road_buffer = edges.buffer(distance).unary_union
    buildings_near_roads = buildings[buildings.intersects(road_buffer)]
    return buildings_near_roads

buildings_near_roads = filter_buildings_near_roads(buildings, edges)
```

The code above demonstrates the process of generating realistic customer locations for a delivery simulation in Vienna. Using spatial data from OpenStreetMap (OSM) via the OSMnx library, the objective is to identify buildings that are near roads, ensuring potential delivery points are accessible and practical for real-world scenarios. First, the road network of Vienna is retrieved using the OSMnx function `graph_from_place`, focusing on drivable roads. The road network is converted into GeoDataFrames of nodes and edges using the `graph_to_gdfs` function, enabling spatial analysis and operations on the road

network. Next, building geometries within Vienna are extracted from OSM using the `geometries_from_place` function, filtering for elements tagged as buildings. This ensures the inclusion of all potential customer locations. Finally, the function `filter_buildings_near_roads` is implemented to select buildings located within a specified buffer distance (default: 50 meters) from the road network. A buffer zone is created around the roads, and spatial intersection operations filter buildings within this zone. This step produces a refined set of buildings near roads, representing feasible delivery points for the simulation. This code ensures that the simulation aligns potential customer locations with accessible road infrastructure, reflecting realistic delivery scenarios.

Listing 2: Generating and Mapping Customer and Depot Locations

```
# Generate 50 random customer locations
num_customers = 50
np.random.seed(0)
customer_locations = get_random_points_within_buildings(
    buildings_near_roads, num_customers)

# Project customer locations to the nearest road nodes and
    convert to lat/lon
customer_nodes = [ox.distance.nearest_nodes(graph, lon,
    lat) for lat, lon in customer_locations]
customer_latlons = [(graph.nodes[node]['y'], graph.nodes[
    node]['x']) for node in customer_nodes]

# Set the depot location to Vienna's suburbs and project
    to the nearest road node
depot_location = (48.2500, 16.4500)
depot_node = ox.distance.nearest_nodes(graph,
    depot_location[1], depot_location[0])

# Print customer locations and depot location in lat/lon
for i, (lat, lon) in enumerate(customer_latlons, 1):
    print(f"Customer {i}: Lat {lat}, Lon {lon}")
print(f"Depot: Lat {graph.nodes[depot_node]['y']}, Lon {
    graph.nodes[depot_node]['x']}")
```

This section of the code generates 50 random customer locations within the filtered building geometries near roads, ensuring realistic spatial distribution within the delivery area. Using the `get_random_points_within_buildings` function, these points are generated with a fixed random seed for reproducibility. Each location is then mapped to the nearest road node in the network using the OSMnx `nearest_nodes` function, ensuring alignment with the road

network required for routing. The latitude and longitude of these mapped nodes are extracted for reference. Additionally, the depot location is set in Viennas suburbs at coordinates (48.2500, 16.4500) and similarly projected to the nearest road node to integrate seamlessly into the road network. Finally, the latitude and longitude of both customer and depot locations are printed, providing transparency and clarity for their use in subsequent simulation tasks. This setup ensures that all locations are practically aligned with the road infrastructure, forming a robust foundation for routing simulations.

Listing 3: Generating Customer Data with Preferences

```
# Generate customer data with preferences
def generate_customers_with_preferences(num_customers):
    time_slots = [(12*60, 13*60), (13*60, 14*60), (14*60,
        15*60)]  # Use minutes to represent time slots
    customers = []
    for i in range(num_customers):
        lat, lon = customer_latlons[i]
        node = customer_nodes[i]
        demand = 6
        preferences = random.sample(time_slots, len(
            time_slots))  # Randomly sort time slots
        customers.append({
            'arrival_order': i,
            'lat': lat,
            'lon': lon,
            'node': node,
            'demand': demand,
            'preferences': preferences
        })
    return customers

customers = generate_customers_with_preferences(
    num_customers)

# Save customer data and depot node
customer_data = {
    'customers': customers,
    'depot_node': depot_node
}

# Save customer data to file
pd.to_pickle(customer_data, 'customer_data.pkl')
```

The code above demonstrates the process of generating customer data with associated delivery preferences for a simulation scenario. The primary objec-

tive is to assign each customer a set of delivery preferences, represented as time slots, along with their geographic details, demand, and routing information. The generated data is saved for future use in the simulation. Firstly, the function `generate_customers_with_preferences` is defined to create a list of customers. It takes the number of customers (`num_customers`) as an input. Each customer is assigned a unique identifier (`arrival_order`), latitude (`lat`), longitude (`lon`), and the corresponding nearest road node (`node`). Their demand is fixed at 6 units, and their delivery preferences are represented by randomly shuffled time slots. Time slots are stored in minutes for better precision and compatibility. The generated customer data is stored in a dictionary under the key `customers`, along with the depot's nearest road node (`depot_node`). Finally, the dictionary is saved as a `.pkl` file using `pandas` serialization capabilities for efficient storage and subsequent retrieval. This implementation ensures that customer data, including their location, demand, and delivery preferences, is properly structured and preserved for use in later stages of the simulation framework.

A.2 Generating Feasibility Check Instances

Listing 4: Calculating and Caching Road Network Distances

```
# Load customer data and depot node
customer_data = pd.read_pickle('customer_data.pkl')
customers = customer_data['customers']
depot_node = customer_data['depot_node']
graph = ox.graph_from_place("Vienna, Austria",
    network_type='drive')

# Calculate actual road network distance and cache it
def compute_route_time(G, source, target):
    try:
        route_length = nx.shortest_path_length(G, source,
            target, weight='travel_time')
        return route_length
    except nx.NetworkXNoPath:
        return float('inf')  # Return infinity to indicate
            no path

# Cache distances
distance_cache = {}

def cached_compute_route_time(G, source, target):
    if (source, target) in distance_cache:
        return distance_cache[(source, target)]
```

```
    distance = compute_route_time(G, source, target)
    distance_cache[(source, target)] = distance
    return distance
```

This section of the code is responsible for loading customer data and computing the shortest path distances within a road network, using the OpenStreetMap data of Vienna. Initially, customer data and the depot node are loaded from a pre-saved file, and the road network graph is generated using the OSMnx library. The primary function `compute_route_time` calculates the shortest path between two nodes in the graph based on travel time. If no path exists, it returns infinity to indicate inaccessibility. To improve efficiency, a caching mechanism is implemented through the `cached_compute_route_time` function. It first checks if the distance between two nodes is already stored in the `distance_cache`. If the value exists, it is retrieved directly, avoiding redundant calculations. Otherwise, the distance is computed using `compute_route_time`, stored in the cache, and then returned. This mechanism ensures faster computations, particularly when handling repetitive queries in large datasets. This approach enables efficient and scalable calculations of travel times, which are crucial for solving the vehicle routing problem within time constraints.

Listing 5: VRPTW Solver

```
# VRPTW solver to check feasibility
def vrptw_solver(customers, vehicle_capacity, num_vehicles):
    def create_data_model(customers, vehicle_capacity, num_vehicles):
        data = {}
        data['distance_matrix'] = generate_distance_matrix(customers)
        data['time_windows'] = [(0, 1440)] + [customer['preferences'][0] for customer in customers]
        data['demands'] = [0] + [customer['demand'] for customer in customers]
        data['service_times'] = [0] + [service_time] * len(customers)
        data['vehicle_capacities'] = [vehicle_capacity] * num_vehicles
        data['num_vehicles'] = num_vehicles
        data['depot'] = 0
        return data

    def generate_distance_matrix(customers):
        num_customers = len(customers)
        all_nodes = [depot_node] + [customer['node'] for customer in customers]
```

```
        distance_matrix = np.zeros((num_customers + 1,
            num_customers + 1))
        distances = Parallel(n_jobs=-1)(delayed(
            cached_compute_route_time)(graph, all_nodes[i],
             all_nodes[j]) for i in range(num_customers +
            1) for j in range(num_customers + 1) if i != j)
        k = 0
        for i in range(num_customers + 1):
            for j in range(num_customers + 1):
                if i != j:
                    distance = distances[k]
                     # If the distance is infinity, set it
                         to a very large value to indicate
                         unreachable
                    if np.isinf(distance):
                        distance_matrix[i][j] = 1e9  # Set
                            to a large value
                    else:
                        distance_matrix[i][j] = distance
                    k += 1
        return distance_matrix

    data = create_data_model(customers, vehicle_capacity,
        num_vehicles)
    manager = pywrapcp.RoutingIndexManager(len(data['
        distance_matrix']), data['num_vehicles'], data['
        depot'])
    routing = pywrapcp.RoutingModel(manager)

    def distance_callback(from_index, to_index):
        from_node = int(manager.IndexToNode(from_index))
        to_node = int(manager.IndexToNode(to_index))
        if from_node < len(data['distance_matrix']) and
            to_node < len(data['distance_matrix']):
             return int(data['distance_matrix'][from_node][
                 to_node])
        return int(1e9)  # Return a very large value to
            handle index overflow cases

    transit_callback_index = routing.
        RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(
        transit_callback_index)

    def demand_callback(from_index):
        from_node = int(manager.IndexToNode(from_index))
```

```
48          if 0 <= from_node < len(data['demands']):
49              return int(data['demands'][from_node])
50          return int(1e9)
51
52      demand_callback_index = routing.
            RegisterUnaryTransitCallback(demand_callback)
53      routing.AddDimensionWithVehicleCapacity(
54          demand_callback_index,
55          0,
56          data['vehicle_capacities'],
57          True,
58          'Capacity')
59
60      time = 'Time'
61      routing.AddDimension(
62          transit_callback_index,
63          30,
64          1440,
65          False,
66          time)
67      time_dimension = routing.GetDimensionOrDie(time)
68
69      for location_idx, time_window in enumerate(data['
            time_windows']):
70          if location_idx == 0:
71              continue
72          index = manager.NodeToIndex(location_idx)
73          time_dimension.CumulVar(index).SetRange(
                time_window[0], time_window[1])
74
75      # Use PATH_CHEAPEST_ARC as the initial solution
            strategy and increase time limit
76      search_parameters = pywrapcp.
            DefaultRoutingSearchParameters()
77      search_parameters.first_solution_strategy =
            routing_enums_pb2.FirstSolutionStrategy.
            PATH_CHEAPEST_ARC
78      search_parameters.local_search_metaheuristic =
            routing_enums_pb2.LocalSearchMetaheuristic.
            GUIDED_LOCAL_SEARCH  #
79      search_parameters.time_limit.seconds = time_limit  #
            Set solving time limit
80
81
82      solution = routing.SolveWithParameters(
            search_parameters)
```

```
83
84      if solution:
85          routes = []
86          service_times = {}
87          for vehicle_id in range(data['num_vehicles']):
88              index = routing.Start(vehicle_id)
89              route = []
90              while not routing.IsEnd(index):
91                  node_index = manager.IndexToNode(index)
92                  route.append(node_index)
93                  if node_index != 0:
94                      service_start = solution.Min(
                            time_dimension.CumulVar(index))
95                      service_end = service_start +
                            service_time
96                      service_times[node_index - 1] = (
                            service_start, service_end)
97                  index = solution.Value(routing.NextVar(
                        index))
98              route.append(manager.IndexToNode(index))
99              routes.append(route)
100         return True, solution, routing, manager, routes,
                service_times
101     else:
102         return False, None, None, None, None, None
```

The `vrptw_solver` function is a robust implementation of the VRPTW using Google's OR-Tools library. It is designed to determine the feasibility of delivery plans while adhering to operational constraints such as vehicle capacity, customer time windows, and service times. This section summarizes its key components and processes. First, the solver initializes a data model using the `create_data_model` function. This data model includes essential components such as a distance matrix representing travel times between the depot and customer locations, time windows specifying the permissible delivery times for each customer, and fixed service times. Additionally, it incorporates vehicle-specific details like capacities and the number of available vehicles, as well as customer demands. This structured data model forms the foundation for solving the VRPTW. The distance matrix is generated using the `generate_distance_matrix` function. This function calculates pairwise travel times between all nodes (depot and customers) while leveraging a caching mechanism to improve computational efficiency. If no path exists between two nodes, a large value is assigned to indicate unreachability. This approach ensures that the solver can efficiently handle dynamic routing scenarios. The routing process is managed by the OR-Tools `RoutingIndexManager`

and `RoutingModel`. These tools facilitate mapping real-world customer nodes to internal indices and enable the optimization of routes. Callback functions are used to dynamically evaluate constraints during routing, such as distance costs and vehicle capacities. The solver incorporates additional constraints, such as ensuring deliveries occur within specified time windows, by leveraging OR-Tools' time dimension features. To find a solution, the solver employs the `PATH_CHEAPEST_ARC` strategy for the initial solution, followed by the `GUIDED_LOCAL_SEARCH` metaheuristic for further optimization. The solving process is bounded by a user-defined time limit, ensuring practical computation times even for complex instances. The output of the solver includes detailed routes for each vehicle, specifying the sequence of customer deliveries and their corresponding service times. If no feasible solution exists, the solver provides feedback on infeasibility, enabling decision-makers to adjust parameters or consider alternative delivery strategies. In summary, the `vrptw_solver` function integrates advanced routing algorithms and constraints to effectively handle the complexities of VRPTW in AHD systems. Its flexible and efficient design makes it suitable for dynamic and real-world delivery environments.

Listing 6: Simulate customer arrivals and generate time slot selections

```
# Simulate customer arrivals and generate time slot selections
def simulate_customer_arrival(customers, time_slots, vehicle_capacity, num_vehicles):
    accepted_customers = []
    data = []
    all_routes = []
    all_service_times = {}

    for i in range(0, len(customers)):

        for customer in customers:
            feasible_time_slots = []
            print(f"\nCustomer {customer['customer_arrival_order']} arrives:")
            print(f"Node: {customer['node']}, Latitude: {customer['lat']}, Longitude: {customer['lon']}")

            preferences = random.sample(time_slots, len(time_slots))
            customer['preferences'] = preferences

            print(f"Customer {customer['customer_arrival_order']}'s preference list
```

```
                (ordered):")
            for preferred_slot in preferences:
                print(f"{preferred_slot[0]//60:02d}:{
                    preferred_slot[0]%60:02d}-{
                    preferred_slot[1]//60:02d}")

            for preferred_slot in preferences:
                customer['preferences'] = [preferred_slot]
                feasible, solution, routing, manager,
                    routes, service_times = vrptw_solver(
                    accepted_customers + [customer],
                    vehicle_capacity, num_vehicles)
                if feasible:
                    feasible_time_slots.append(
                        preferred_slot)
                    break
                else:
                    print(f"Customer {customer['
                        customer_arrival_order']} is not
                        feasible for time slot {
                        preferred_slot}")

            if feasible_time_slots:
                accepted_customers.append(customer)
                print(f"Customer {customer['
                    customer_arrival_order']} is feasible
                    with time slot: {feasible_time_slots
                    [0]}")

                if service_times is not None:
                    service_start, service_end =
                        service_times.get(customer['
                        customer_arrival_order'] - 1, (None
                        , None))
                    all_service_times.update(service_times
                        )
                else:
                    service_start, service_end = (None,
                        None)

                data.append({
                    'customer_arrival_order': customer['
                        customer_arrival_order'],
                    'node': customer['node'],
                    'lat': customer['lat'],
                    'lon': customer['lon'],
```

```
                    'feasible': 1,
                    'time_slots': [f"{feasible_time_slots
                        [0][0]//60:02d}:{
                        feasible_time_slots[0][0]%60:02d}-{
                        feasible_time_slots[0][1]//60:02d}"
                        ],
                    'service_start': service_start,
                    'service_end': service_end
                })

                all_routes.append(routes)
            else:
                print(f"Customer {customer['
                    customer_arrival_order']} is not
                    feasible.")
                data.append({
                    'customer_arrival_order': customer['
                        customer_arrival_order'],
                    'node': customer['node'],
                    'lat': customer['lat'],
                    'lon': customer['lon'],
                    'feasible': 0,
                    'time_slots': None,
                    'service_start': None,
                    'service_end': None
                })


        gc.collect()

    return data, all_routes, all_service_times
```

The `simulate_customer_arrival` function simulates the dynamic process of customer arrivals in an AHD system, generating time slot preferences and evaluating their feasibility. This process mimics real-world scenarios where customers request deliveries and the system must dynamically assess and accommodate these requests based on operational constraints. The simulation treats customers to process them iteratively. Upon arrival, each customer is assigned a randomly ordered list of preferred delivery time slots, reflecting real-world variability in customer preferences. These preferences are printed for reference. To evaluate feasibility, the function iteratively assigns each customer their most preferred time slot and uses the `vrptw_solver` to determine if the slot is operationally feasible given the current set of accepted customers, vehicle capacities, and routing constraints. If a feasible slot is found, the customer is added to the list of accepted customers, and their details, including

the assigned time slot and service times, are recorded. The routing information and service times are also stored for further analysis. For customers whose requests are not feasible, the function records their information as infeasible, noting the absence of feasible time slots and service times. After processing each customer, garbage collection is triggered to manage memory usage and prevent leaks during iterative computations. This simulation function is essential for testing and validating the feasibility of delivery time slots in dynamic and operationally constrained AHD environments. It provides a practical framework for assessing the adaptability and efficiency of the system under varying customer demands and constraints.