People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Yahya Fares Medea



Faculty of Sciences
Department of Mathematics and Computer Science

Submitted for obtaining **Master Degree in Computer Science**
Applied Artificial Intelligence

Entitled

End-to-end price prediction model for CAMIO

Presented by:

- *Chikh Imad Eddine*
- *Titraoui Akram*

Committee Members

President	Dr Abdoun Mohamed Nabil	University YF Medea
Examiner	Dr Ahmed Cherif Mazari	University YF Medea
Supervisor	Dr Benaissa Azzeddine Rachid	University YF Medea

Academic Year: **2024–2025**

Dedication

Dedication by Imad Eddine Chikh

I dedicate this humble work with deep gratitude and heartfelt appreciation to:

My beloved parents, whose unwavering support, sacrifices, and encouragement have been the foundation of every step I have taken.

My dear sister and cherished family, for being a source of strength and warmth throughout this journey.

My teachers and professors, especially **Dr. Benaissa Azzedine Rachid**, for his invaluable supervision, trust, and encouragement. Your guidance has been instrumental.

My fellow students and colleagues, whose collaboration and support made this academic journey meaningful and memorable — particularly my thesis partner, **Akram Titraoui**, for his dedication, cooperation, and true team spirit.

My teammates and co-workers at **CAMIO** — Mr. Mustapha, for his trust; Fouad, for the opportunity; Arabi, for his guidance; and Waheb, Nacereddine, Lalla, Lyna, and Wafa, for their encouragement. Your contributions were essential to my growth and the success of this work.

The entire university staff, for fostering a learning environment that encourages creativity and excellence.

To all who contributed, directly or indirectly — this achievement is also yours.

Dedication

Dedication by Akram Titraoui

I dedicate this humble work with sincere gratitude to:

My beloved parents, for their unconditional love, sacrifices, and constant encouragement that have guided me every step of the way.

My dear family and sister, for being a source of strength and warmth throughout this journey.

My teachers and professors, whose guidance and knowledge have deeply shaped my academic path.

My colleagues and fellow students, for their support, collaboration, and shared motivation.

A special thanks to **Dr. Benaissa Azzedine Rachid**, for his invaluable supervision, trust, and encouragement.

And to my thesis partner **Imad Chikh**, for his dedication, cooperation, and true team spirit.

To all who contributed, directly or indirectly — this achievement is also yours.

Acknowledgements

We would like to express our sincere gratitude to CAMIO Company and its representatives for their valuable support in fostering innovation and scientific research. Their contribution through the provision of essential resources—including relevant datasets, cloud computing services, and dedicated staff—created a productive environment that greatly facilitated our work.

We are deeply thankful to our supervisor, Dr. Benaissa Azzedine Rachid, for his commitment, continuous support, and wise guidance throughout this journey. His dedication and insightful advice played a key role in the successful completion of this project.

We also extend our appreciation to the faculty and staff of Yahia Fares University of Médéa for their efforts in maintaining a supportive academic environment.

Finally, we warmly thank all our fellow students and colleagues, whose collaboration, encouragement, and team spirit provided motivation and enriched our learning experience.

ملخص

في هذه الدراسة، نقدم إطار عمل مدفوعاً بالتعلم الآلي للتنبؤ الشامل بأسعار الشحن على منصة كاميو اللوجستية. نعتمد على نموذج انحدار طست لمحاكاة العلاقات غير الخطية بين مسافة الرحلة ونوع الشاحنة والمناطق الجغرافية. من خلال استخدام تعزيز التدرج لأشجار القرار المجمعة مع تضمين تنظيم مدمج والقدرة على التعامُل مع البيانات المتفرقة، يعالج هذا النهج قصور الخوارزمية التقليدية القائمة على القواعد. على مجموعة الاختبار المنفصلة، حقق نموذج التعزيز التدرّجي المتطرِّف خطأ الجذر التربيعي المتوسط (ار ام اس اي) بقيمة ٢٠٥٩٩٢، وخطأ المتوسط المطلق (ام اي اي) بقيمة ٩٦٦٧٧١، ومعامل التحديد (ار مربع) بقيمة ٤٩٠، مما يعكس تحسناً ملموساً في دقة التقدير ومتانة النموذج تجاه القيم الشاذة. علاوة على ذلك، تؤكد مقاييس أهمية المتغيرات في النموذج أن مسافة الرحلة والعوامل الإقليمية هما المحركان الرئيسيان لتغير الأسعار، مما يوفر رؤى قابلة للتطبيق في التخطيط التشغيلي. تسلط هذه النتائج الضوء على كفاءة نماذج أشجار التعزيز التدرجي في بيئات التسعير المعقدة وتشير إلى مسار واضح نحو تحسين رضا العملاء وموثوقية العمليات لشركة كاميو.

الكلمات المفتاحية: التنبؤ بالأسعار، التعلم الآلي، الانحدار، تحليل البيانات.

Abstract

In this study, we present a machine learning–driven framework for end-to-end price prediction within the CAMIO logistics platform. We employ XGBoost regression to model the inherently non-linear relationships among trip distance, truck type, and geographic zones. Through gradient boosting of decision-tree ensembles—with integrated regularization and sparsity awareness—our approach addresses the limitations of the prior rule-based algorithm. On a held-out test set, the XGBoost model achieved a root-mean-square error (RMSE) of 2.995.02, a mean absolute error (MAE) of 1.776.69, and an R^2 of 0.94, reflecting substantial gains in estimation accuracy and resilience to outliers. Moreover, the model's feature-importance metrics confirm that distance and regional factors are the primary drivers of price variation, offering actionable insights for operational planning. These results underscore the efficacy of gradient-boosted tree models in complex pricing environments and suggest a clear path toward enhanced customer satisfaction and operational reliability for CAMIO.

Keywords: Price prediction, machine learning, regression, data, data analysis.

Résumé

Dans cette étude, nous présentons un cadre piloté par l'apprentissage automatique pour la prédiction de bout en bout des tarifs de transport sur la plateforme logistique CAMIO. Nous utilisons la régression XGBoost pour modéliser les relations non linéaires entre la distance du trajet, le type de camion et les zones géographiques. Grâce au gradient boosting d'ensembles d'arbres de décision—intégrant la régularisation et la gestion de la parcimonie des données—notre approche pallie les limites de l'algorithme basé sur des règles antérieur. Sur un ensemble de test indépendant, le modèle XGBoost a atteint une erreur quadratique moyenne (RMSE) de 2995,02, une erreur absolue moyenne (MAE) de 1776,69 et un coefficient de détermination (R^2) de 0,94, reflétant des gains substantiels en précision d'estimation et en robustesse face aux valeurs aberrantes. De plus, les métriques d'importance des fonctionnalités confirment que la distance et les facteurs régionaux sont les principaux leviers de variation des tarifs, offrant des perspectives opérationnelles concrètes. Ces résultats soulignent l'efficacité des modèles d'arbres renforcés par gradient dans des environnements de tarification complexes et ouvrent une voie vers une meilleure satisfaction client et une fiabilité opérationnelle accrue pour CAMIO.

Mots clés : Prédiction des prix, apprentissage automatique, régression, analyse de données.

Contents

List of Figures i

List of Tables iii

Nomenclature iv

General Introduction 1

Research Question . . . 1

Overview of the Chapters . . . 2

Foundations and Context of Price Prediction 5

1.1 Introduction . . . 5

1.2 Historical Background of Price Prediction . . . 5

1.3 Understanding Prediction in AI . . . 6

1.3.1 Types of Price Prediction . . . 6

1.3.2 Factors Influencing Price Prediction . . . 6

1.3.3 Prediction versus Forecasting . . . 7

1.3.4 Prediction Pipelines in AI . . . 7

1.4 Domains of Price Prediction . . . 7

1.5 Case Studies in Price Prediction . . . 9

1.6 Conclusion . . . 10

Introduction to Machine Learning and Deep Learning 12

2.1 Introduction . . . 12

2.2 Machine Learning . . . 12

2.2.1 What is Machine Learning? . . . 12

2.2.2 Learning Algorithms . . . 12

2.2.3 Supervised Learning . . . 13

2.2.4 Unsupervised Learning . . . 13

2.2.5 Capacity, Overfitting, and Underfitting . . . 14

2.2.6 Hyperparameters and Validation Sets . . . 16
2.3 Deep Learning . . . 17
2.3.1 Linear Neural Networks . . . 17
2.3.2 Feedforward Neural Networks (FNNs) . . . 17
2.3.3 Regularization Techniques . . . 18
2.3.4 Hyperparameters in Deep Learning . . . 18
2.3.5 Convolutional Neural Networks (CNNs) . . . 18
2.3.6 Recurrent Neural Networks (RNNs) . . . 19
2.3.7 Other Architectures and Techniques . . . 19
2.4 Prediction in Machine Learning . . . 19
2.4.1 Regression vs. Classification . . . 19
2.4.2 Evaluation Metrics . . . 20
2.4.3 Common Challenges and How to Solve Them . . . 21
2.5 Machine Learning Lifecycle . . . 22
2.6 Related Works . . . 23
2.7 Conclusion . . . 24

Technical Methodology and Proposed Approach **27**
3.1 Introduction . . . 27
3.2 Project Objectives . . . 27
3.3 Architecture Overview . . . 27
3.4 Technologies and Tools . . . 29
3.4.1 Python . . . 29
3.4.2 Pandas . . . 29
3.4.3 NumPy . . . 29
3.4.4 Scikit-learn . . . 29
3.4.5 Microsoft Azure . . . 30
3.4.6 Azure DevOps . . . 30
3.4.7 MLflow . . . 30
3.5 Data Overview and Exploration . . . 31
3.5.1 Data Source and Acquisition . . . 31
3.5.2 Exploratory Data Analysis (EDA) . . . 31
3.6 Data Preprocessing . . . 34
3.6.1 Handling missing/invalid data . . . 34
3.6.2 Outlier Detection and Removal . . . 36
3.6.3 Normalization/Standardization of Features . . . 38
3.7 Feature Design and Engineering . . . 39

3.7.1 Overview of the Feature Engineering Pipeline . . . 40
3.7.2 Custom Feature Creation . . . 40
3.7.3 Preprocessing Pipeline . . . 42
3.7.4 Feature Selection . . . 43
3.7.5 Pipeline Architecture and Deployment Readiness . . . 43
3.7.6 Visualization and Diagnostic Tools . . . 43
3.7.7 Advantages of the Approach . . . 44
3.8 Model Design and Training . . . 44
3.8.1 Problem Formulation . . . 44
3.8.2 Model Candidates . . . 44
3.9 Deployment Strategy for Machine Learning Models . . . 49
3.9.1 Key MLOps Practices . . . 50
3.9.2 Cloud-Based Deployment with Azure ML . . . 50
3.10 Conclusion . . . 51

Implementation and Results **53**
4.1 Introduction . . . 53
4.2 Comparison of Experimental Model Results . . . 53
4.2.1 Tested Models . . . 53
4.2.2 Experiment Tracking in Azure ML . . . 53
4.2.3 Performance Comparison . . . 55
4.2.4 Feature Importance Analysis . . . 56
4.3 Model Selection and Justification . . . 57
4.3.1 Selection Criteria . . . 57
4.3.2 Chosen Model . . . 58
4.3.3 Justification for Selection . . . 58
4.4 Comparison of the Selected Model with the Classical Algorithm . . . 59
4.4.1 Reproduction of the Classical Performance . . . 59
4.4.2 Quantitative Performance Comparison . . . 60
4.4.3 Visual Comparison . . . 60
4.4.4 Practical Impact . . . 63
4.5 Deploying and Model Integration . . . 63
4.5.1 Automated Deployment via CI/CD Pipeline . . . 63
4.5.2 Endpoint Overview in Azure ML . . . 64
4.5.3 Model Versioning and Endpoint Traffic Management . . . 65
4.5.4 Integration into CAMIO System . . . 67
4.5.5 Monitoring and Maintenance . . . 68

4.6 Use-Case Simulation within the CAMIO Application 68
4.7 Challenges and Obstacles 72
4.8 Future Work 73
4.8.1 Expansion to Other Markets and Regions 73
4.8.2 Bid Count Prediction 73
4.8.3 Cargo Matching Using NLP 73
4.8.4 Multi-Objective Optimization 74
4.8.5 Operationalization with Azure ML Pipelines 74
4.8.6 Real-Time Feedback Loops and Online Learning 74
4.9 Conclusion 75

General Conclusion **76**

Bibliography **77**

List of Figures

2.1 The difference between overfitting and underfitting 15
2.2 The full ML lifecycle integrating ML, DEV, and OPS processes 23
3.1 Architecture Overview 28
3.2 Geographic Heatmap of pickup location density based on historical shipment data 33
3.3 Correlation matrix of selected features highlighting relationships with freight price 33
3.4 Boxplot of freight prices by truck type, color-coded using Z-score-based outlier labeling (threshold = 1.5) 37
4.1 Overview of regional experiment groups in Azure ML 54
4.2 Detailed view of runs inside the `dz_trips_experiment` experiment 54
4.3 Metrics and parameters logged for a single run in Azure ML 55
4.4 Top feature importances used by the model in predicting trip prices. 56
4.5 Residual plot for XGBoost predictions 57
4.6 Comparison of Actual Prices vs. Predictions from ML and Classical Models 61
4.7 Actual vs. Estimated Prices using Old and ML Methods. 62
4.8 Comparison of Classical and ML models on key performance metrics: MAE, RMSE, R^2, and MAPE. 62
4.9 CI/CD pipeline overview for training and deployment 64
4.10 Azure ML Studio interface showing deployed endpoints and configuration . 65
4.11 Model registry in Azure ML Studio listing multiple versions of the freight estimation models 66
4.12 Example of traffic distribution between deployed models 67
4.13 Integration flow of the deployed model with CAMIO's backend system . . . 67
4.14 Monitoring interface for the online endpoint in Azure 68
4.15 Selection of truck type and cargo capacity 69
4.16 User sets pickup and drop-off points using the map 70
4.17 Trip information entry and order preview before confirmation 70

4.18 Estimated price returned from ML model and shown to user 71
4.19 Truck owner bids submitted in response to the posted trip 72

List of Tables

1.1 Comparison between Prediction and Forecasting [17] 7
2.1 Related Works on Price Prediction (Regression-Based) with Reported Errors and R^2 24
4.1 Comparison of model performance 56
4.2 Performance metrics comparison between classical and ML models 60

Nomenclature

Abbreviations / Acronyms

AI	Artificial Intelligence
CD	Continuous Deployment
CI	Continuous Integration
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DevOps	Development and Operations
DL	Deep Learning
KMeans	K-Means Clustering
LIME	Local Interpretable Model-agnostic Explanations
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
PCA	Principal Component Analysis
RF	Random Forest
RFE	Recursive Feature Elimination
RMSE	Root Mean Squared Error
SHAP	SHapley Additive exPlanations
SVM	Support Vector Machine
SVR	Support Vector Regression

t-SNE	t-Distributed Stochastic Neighbor Embedding
XGB	Extreme Gradient Boosting

General Introduction

In recent years, the logistics and freight industry has witnessed a profound transformation driven by advances in artificial intelligence and data-driven decision-making. *Price prediction*—the task of estimating the cost of a shipment before service initiation—has become a critical component of customer satisfaction, operational efficiency, and competitive differentiation. Traditional rule-based algorithms often rely on fixed tables of distances, fuel surcharges, or simplistic load factors, which can yield estimates that deviate significantly from market reality. As a result, shippers and carriers experience reduced trust, lower conversion rates, and suboptimal resource utilization.

In collaboration with **CAMIO**—a leading Algerian platform that connects individual cargo owners with professional carriers via a mobile and web application—we propose an end-to-end neural network–based solution for dynamic price prediction. Our mission is twofold: first, to empower CAMIO's customers with accurate, transparent fare estimates at the point of booking; and second, to replace the legacy pricing module, which has produced erratic quotes that undermine both user experience and business performance.

Using a fully automated CI / CD machine learning pipeline deployed in **Azure ML**, we integrate robust data preprocessing, advanced feature engineering, state-of-the-art regression models and scalable API endpoints. This holistic workflow ensures that new data, model refinements, and infrastructure updates flow seamlessly from development to production, maintaining high availability and traceability.

Research Question

Primary Question:

> How can a machine learning–driven price prediction model, integrated within a CI / CD-enabled Azure machine learning environment, improve the accuracy and reliability of shipment cost estimates in the CAMIO platform compared to the existing rule-based algorithm?

Sub-questions:

1. Which preprocessing and feature engineering strategies maximize model performance on heterogeneous logistics data (e.g., route distance, cargo weight, truck type)?
2. Which machine learning algorithms or architectures provide the best balance of bias and variance for the price prediction task?
3. How do continuous integration and continuous deployment impact the maintainability, monitoring, and overall lifecycle of the machine learning model in production?

Overview of the Chapters

Chapter 1: Foundations and Context of Price Prediction

This chapter reviews the historical evolution of price estimation systems, differentiates the core concepts of prediction versus forecasting, and surveys domains and case studies where machine learning–based dynamic pricing has delivered tangible business value.

Chapter 2: Introduction to Machine Learning and Deep Learning

This chapter introduces the fundamental principles of supervised regression, compares traditional machine learning algorithms (e.g., decision trees, support vector machines) with deep learning architectures (e.g., feedforward and convolutional networks), and outlines evaluation metrics pertinent to price prediction tasks.

Chapter 3: Technical Methodology and Proposed Approach

This chapter details the end-to-end solution pipeline, covering data acquisition and preprocessing (handling missing values, outlier detection, normalization), feature engineering, model selection, and tuning (across various machine learning techniques), and the design of a CI/CD pipeline within Azure Machine Learning.

Chapter 4: Implementation and results

This chapter presents the experimental setup, performance comparisons among candidate machine learning models, justification for the final model selection, and deployment

procedures. It also includes visual analyses, real-world use case simulations, and discussions of challenges encountered during integration and operationalization.

Chapter 1

Foundations and Context of Price Prediction

1.1 Introduction

This chapter introduces the importance of price prediction in sectors such as logistics, finance, and e-commerce, focusing on its role in forecasting future product or service values based on historical data. In logistics platforms like **CAMIO**, price prediction plays a crucial role in providing users with realistic insights and expectations before receiving bids. By offering estimated delivery prices upfront, it helps guide users in making informed decisions and proceeding confidently with their trips. Accurate predictions can influence user behavior—when estimates are too far off from expectations, users may reconsider or abandon their choices. Thus, predictive models help enhance trust, usability, and engagement on the platform.

The chapter traces the evolution of prediction techniques from traditional econometric models to modern AI and machine learning approaches while outlining key concepts such as prediction types, influencing factors, and modeling paradigms. It also presents real-world applications and case studies, forming the foundation for developing a tailored machine learning model for the prediction of delivery prices in the chapters that follow.

1.2 Historical Background of Price Prediction

The practice of predicting prices dates back to early trading civilizations, where merchants used observation and experience to anticipate changes in supply and demand. Over centuries, more structured approaches were developed:

- **Early Trade Practices:** Traders historically relied on seasonal patterns and anecdotal evidence.
- **Economic Theories:** The advent of economic science introduced models considering supply, demand, and market equilibrium.
- **Statistical Methods:** In the 19th and 20th centuries, statistical models like moving averages and regression analysis were adopted.
- **Modern Developments:** The expansion of computing power and data availability in the late twentieth century allowed for more complex forecasting methods, enabling multi-variable analyses and more accurate predictions.

1.3 Understanding Prediction in AI

Price prediction is the process of forecasting future market prices based on historical data, statistical patterns, and external influences. It serves as a critical tool for businesses aiming to stabilize revenue streams, set competitive pricing, and adapt swiftly to market fluctuations.

1.3.1 Types of Price Prediction

The following section outlines the primary types of price prediction methodologies.

- **Short-Term Prediction:** Focused on immediate or near-future price changes. This is vital for businesses engaging in dynamic pricing strategies that require responsiveness to real-time demand and supply variations.
- **Long-Term Prediction:** Involves forecasting prices over extended periods, supporting strategic decisions such as inventory management, investment planning, and budgeting.
- **Real-Time Prediction:** Incorporates live data streams to instantly adjust pricing, crucial in sectors such as ride-sharing, airline ticketing and last-mile delivery.
- **Batch Prediction:** Conducted periodically, useful for generating regular pricing updates based on historical patterns rather than immediate market fluctuations.

1.3.2 Factors Influencing Price Prediction

Price prediction is influenced by a multitude of factors:

- **Supply and Demand Dynamics:** Fundamental economic forces; an imbalance leads to price volatility.
- **Market Volatility:** Political events, economic crises, and natural disasters can create sudden, significant price shifts.
- **Seasonal Variations:** Industries such as agriculture and retail are heavily influenced by seasonal demand and supply cycles.
- **Competitor Strategies:** Monitoring and reacting to competitor pricing policies is essential for maintaining market share.

- **Consumer Behavior:** Trends in consumer preferences and spending habits affect pricing structures.
- **Regulatory Changes:** New laws or tariffs can alter market prices substantially.

1.3.3 Prediction versus Forecasting

Although terms are often used interchangeably, prediction often implies a more general-purpose approach applicable to various domains, while forecasting typically refers to time-series data. For example, demand forecasting or sales forecasting often implies a temporal dimension that traditional prediction models may not capture.

Item	Prediction	Forecasting
Concept	Estimating a value for an individual case based on known features.	Estimating future trends or aggregate values over time using historical data.
Data Type	Cross-sectional or feature-rich data for individual instances.	Time-series data indexed by time, often without many covariates.
Common Techniques	Regression (Linear, Random Forest, XGBoost); Neural Networks (MLP).	ARIMA or Exponential Smoothing; LSTM or RNN for sequential patterns.

Table 1.1: Comparison between Prediction and Forecasting [17]

1.3.4 Prediction Pipelines in AI

An end-to-end predictive system typically includes data collection, pre-processing , feature engineering, model training, evaluation, and deployment. With recent advances, pipelines also include model monitoring, retraining mechanisms, and real-time inference [3].

1.4 Domains of Price Prediction

Price prediction models are widely used and tailored based on domain-specific needs:

Financial Markets

In finance, price forecasting helps investors and institutions make informed decisions about buying, selling, and holding assets. Stock markets, forex markets, and commodities trading rely heavily on accurate price prediction to maximize returns and manage risks. Ex-

ample: Hedge funds use predictive models to forecast asset prices and dynamically adjust investment portfolios [19].

E-commerce and Retail

Price forecasting is increasingly integral to the operations of e-commerce platforms and retail businesses. Dynamic pricing strategies—driven by real-time data on supply, demand, and competitor behavior—allow firms to adjust product prices with precision. Moreover, predictive models support the design of targeted promotions and discounting schemes, maximizing customer engagement and revenue. Additionally, machine learning techniques are employed to personalize pricing and recommendations by analyzing consumer behavior, enhancing user experience and conversion rates.

Transportation and Logistics

In transportation and logistics, price prediction models are instrumental in optimizing operational efficiency and cost management. These models assist in forecasting route-specific shipment costs and contribute to the strategic planning of delivery schedules. Innovations such as CAMIO's predictive pricing system exemplify how such technologies can reduce the prevalence of empty truck returns by improving load allocation and utilization. As a result, companies can achieve significant savings in fuel, labor, and time while enhancing overall sustainability in freight movement.

Energy Sector

The energy sector utilizes price forecasting models to address both short-term market volatility and long-term strategic planning. Accurate predictions of electricity and oil prices enable utility providers and consumers to optimize energy consumption and manage procurement costs. Furthermore, the rise of renewable energy sources has intensified the need for advanced models that can forecast the variability of solar and wind energy generation. These predictions support grid stability and load balancing efforts, which are crucial for maintaining a reliable energy supply in systems increasingly reliant on intermittent sources.

Healthcare and Insurance

In healthcare and insurance, price prediction plays a pivotal role in cost management and risk assessment. Predictive analytics help estimate the cost of medical treatments,

aiding providers and insurers in budgeting and resource allocation. Insurance companies apply similar models to determine premium pricing based on individual and demographic risk profiles. Additionally, machine learning algorithms are deployed to detect anomalies indicative of fraudulent claims, improving operational efficiency and reducing financial losses.

1.5 Case Studies in Price Prediction

The following case studies illustrate the application and efficacy of various price prediction models in real-world scenarios.

Amazon's Dynamic Pricing Model

Amazon has developed one of the most sophisticated price prediction and adjustment systems globally. Using historical data, real-time competitor pricing, and customer behavior insights, Amazon can update prices for millions of products dynamically, ensuring optimal competitiveness and profitability.

Uber's Surge Pricing

Uber employs real-time prediction models to anticipate supply-demand mismatches. When predicted demand outstrips supply, surge pricing is automatically activated, encouraging more drivers to service the area and balancing the marketplace.

Energy Market Forecasting

Electricity providers predict energy demand and adjust supply strategies to stabilize grid operations and pricing. For example, ISO New England uses advanced models to forecast hourly electricity prices, accounting for factors such as weather and consumption trends.

Used Car Price Prediction Platforms

Several online marketplaces, such as Kelley Blue Book and CarGurus, utilize regression models to estimate the fair market value of used vehicles. These models incorporate a variety of features, including vehicle age, mileage, brand, model, condition, geographic location, and historical sale prices. Linear regression, ridge regression, and advanced ensemble models like XGBoost are commonly employed to capture non-linear relationships and interactions between variables. Accurate pricing predictions not only assist sellers in

setting competitive prices but also help buyers assess whether a listed price is reasonable. This use case highlights how regression modeling enhances transparency and efficiency in peer-to-peer and dealer-based car sales.

1.6 Conclusion

Price prediction is essential for strategic foresight and operational success in all industries. Through historical development, diverse applications, and future trends, the discipline of forecasting prices continues to evolve. For CAMIO, leveraging comprehensive, accurate price prediction models offers the opportunity to enhance service efficiency, minimize operational costs, optimize delivery matching processes, and ultimately gain a significant competitive advantage in the logistics market. Future chapters will focus on detailing the methodologies and practical implementation strategies tailored specifically to CAMIO's operational context.

Chapter 2

Introduction to Machine Learning and Deep Learning

2.1 Introduction

This chapter provides a foundation for understanding machine learning (ML) and its evolution into deep learning (DL). We begin with core definitions, explore the principal learning paradigms, and clarify key concepts such as model capacity, over-fitting / under-fitting, and hyperparameter tuning where relevant seminal academic references are cited to guide a deeper study.

2.2 Machine Learning

Machine Learning (ML) has become a cornerstone of modern artificial intelligence, enabling systems to learn and make predictions from data. In this section, we introduce the fundamental concepts of machine learning, exploring key paradigms, learning algorithms, and challenges such as overfitting and model selection. These foundational elements are critical for understanding how price prediction systems can be designed and optimized.

2.2.1 What is Machine Learning?

Machine learning (ML) is a subset of artificial intelligence (AI) focused on developing algorithms that enable computers to learn patterns from data without being explicitly programmed. Arthur Samuel defined machine learning as "the field of study that gives computers the ability to learn without being explicitly programmed" [33].

Tom Mitchell provided a more formal definition:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [28].

This paradigm has found application across a wide range of domains, such as healthcare, finance, logistics, and e-commerce, particularly in tasks such as price prediction.

2.2.2 Learning Algorithms

Learning algorithms lie at the heart of machine learning systems and are typically categorized into three main paradigms: supervised learning, unsupervised learning, and reinforcement learning. Each paradigm addresses different types of problems and leverages specific algorithmic strategies to learn from data. Among the most widely used algorithms are decision trees, support vector machines (SVMs), ensemble methods (such as random forests and boosting techniques), and neural networks, particularly deep learning models.

These algorithms function by optimizing a predefined loss function, which quantifies the error between predicted and actual outputs. For example, mean squared error (MSE) is commonly employed in regression tasks, while cross-entropy loss is widely used in classification problems. Optimization methods such as gradient descent are utilized to iteratively adjust model parameters and minimize this prediction error.

Machine learning algorithms are applied across a variety of domains, each with distinct objectives:

- **Classification:** Assigning input data to predefined categories or classes, such as identifying spam emails or diagnosing medical conditions.
- **Regression:** Estimating continuous outputs, such as predicting house prices or forecasting stock values.
- **Natural Language Processing (NLP):** Allow machines to comprehend, interpret, and produce human language, including tasks such as sentiment analysis, language translation, and text summarization.
- **Computer Vision:** Extracting and interpreting meaningful information from images and videos, such as object detection, facial recognition, and scene segmentation.

Together, these applications illustrate the versatility and growing impact of machine learning in both academic research and real-world industry use cases.

2.2.3 Supervised Learning

Supervised learning tasks involve labeled datasets, where each input x is associated with an output label y. The goal is to learn a function $f(x) \approx y$ that generalizes well to unseen data. This is particularly applicable in regression problems such as price prediction, where the target variable is continuous.

Algorithms commonly used include linear regression, decision trees, support vector regression, and neural networks.

2.2.4 Unsupervised Learning

Unsupervised learning operates on data without explicit output labels, with the objective of discovering hidden structures or patterns. Common techniques include clustering algorithms like K-Means and DBSCAN, and dimensionality reduction methods such as PCA and t-SNE.

Though not directly applied in price prediction, unsupervised learning can support preprocessing stages, such as customer segmentation or identifying similar product categories, which may influence feature engineering.

2.2.5 Capacity, Overfitting, and Underfitting

Understanding the behavior of machine learning models requires careful consideration of their capacity and how well they generalize from training data to unseen examples. Two common challenges that arise in this context are overfitting and underfitting—both of which can significantly affect model performance. This section explores these concepts, their implications, and methods to mitigate them.

Model Capacity

The capacity of a machine learning model refers to its ability to fit a wide variety of functions or to capture complex patterns from data. High-capacity models can represent intricate relationships between features and target variables, making them well-suited for modeling complex datasets. Conversely, low-capacity models are limited in their ability to capture such patterns, making them more appropriate for simpler tasks or when data is scarce.

Overfitting

Overfitting arises when a model learns the training data too well, including its noise and outliers, rather than generalizing from the underlying distribution. This leads to excellent performance on the training dataset but poor generalization to unseen data. Overfitting is typically a consequence of high model capacity and insufficient regularization or data diversity.

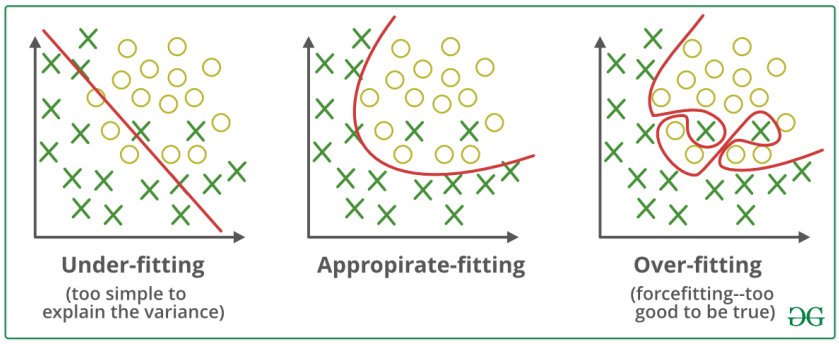
Underfitting

Underfitting, on the other hand, occurs when the model does not capture the underlying structure of the data. This is often due to low capacity, insufficient training time, or overly simplistic assumptions about the data. Under-fit models perform poorly on both training and test data, indicating that they have not learned enough from the available information.

Appropriate fitting

Appropriate fitting ensures the model captures real patterns in the data without being too simple (underfitting) or too complex (overfitting). It strikes a balance between bias and variance, promoting generalization to unseen data. In price prediction, this results in more accurate, reliable estimates that reflect meaningful trends rather than noise. Techniques like cross-validation and regularization help achieve this balance.

The Figure. 2.1 shows the difference between overfitting and underfitting.



[20]

Figure 2.1: The difference between overfitting and underfitting

Mitigation Strategies

To reduce the risk of overfitting and underfitting, several strategies can be employed:

- **Regularization:** Techniques such as L1 (Lasso) and L2 (Ridge) regularization introduce penalties for model complexity, discouraging the model from fitting noise in the data.
- **Cross-validation:** Using methods such as k-fold cross-validation enables more reliable estimation of model performance and aids in detecting over-fitting or under-fitting.
- **Data Augmentation:** In domains such as image or text processing, artificial expansion of the training dataset through augmentation can improve generalization and robustness.
- **Early Stopping:** Monitoring model performance in a validation set during training and stopping when performance begins to degrade can prevent overfitting.

2.2.6 Hyperparameters and Validation Sets

The performance of machine learning models is highly influenced by the choice of hyperparameters and the evaluation strategy used during model development. Unlike model parameters, which are learned from data during training, hyperparameters must be set prior to training and play a critical role in shaping the learning process and the model's generalization capabilities.

Hyperparameters

Hyperparameters are external configurations that control the learning process of machine learning algorithms. Examples include the learning rate, number of hidden layers in a neural network, regularization coefficients, batch size, and tree depth in decision trees. These settings are not learned from the training data, but rather must be defined beforehand and tuned to optimize model performance. Common methods for hyperparameter optimization include: [3] [6]

- **Grid Search**: Exhaustively evaluating combinations of predefined hyperparameter values.
- **Random Search**: Randomly sampling hyperparameter values within specified ranges.
- **Bayesian Optimization**: Using probabilistic models to guide the search for optimal configurations more efficiently.

Validation Sets

To ensure that a model generalizes well to new data, a separate validation set is used during the training process. This subset of data is withheld from the training phase and used to monitor performance, particularly during hyperparameter tuning. Evaluating on the validation set helps in detecting overfitting and selecting the most appropriate model configuration.

Cross-Validation

To further improve the reliability of model evaluation and hyperparameter tuning, cross-validation techniques are employed. One of the most widely used methods is k-fold cross-validation, where the dataset is split into k equal parts, and the model is trained and validated k times—each time using a different fold as the validation set and the remaining

folds for training. This approach provides a more robust estimate of model performance and reduces variance due to data partitioning.

2.3 Deep Learning

Deep learning is a specialized branch of machine learning that leverages neural networks with multiple layers to learn representations of data at various levels of abstraction. It has become a fundamental technology behind many modern AI applications, including image recognition, natural language processing, and, increasingly, predictive analytics such as price prediction. In this section, we delve into the core components and architectures of deep learning systems.

2.3.1 Linear Neural Networks

Linear neural networks represent the simplest form of neural networks, composed of layers with linear activation functions. Despite their simplicity, they lay the groundwork for understanding more complex deep learning models.

A linear neural network maps an input vector $\mathbf{x} \in \mathbb{R}^n$ to an output y using a function of the form:

$$y = \mathbf{w}^T \mathbf{x} + b \tag{2.1}$$

These models are essentially equivalent to linear regression and are limited in their ability to capture non-linear relationships, but serve as a basis for gradient-based optimization and network structure.

2.3.2 Feedforward Neural Networks (FNNs)

Feedforward Neural Networks (FNNs) extend linear models by incorporating non-linear activation functions and multiple layers. These models are also known as Multilayer Perceptrons (MLPs).

Each hidden layer transforms the data using the equation:

$$\mathbf{h}^{(l)} = \sigma(\mathbf{W}^{(l)} \mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}) \tag{2.2}$$

Where σ is a non-linear activation function such as ReLU, sigmoid, or tanh. These models are universal function approximators and form the backbone of many deep learning systems [16].

2.3.3 Regularization Techniques

Deep networks are prone to overfitting due to their high capacity. Regularization techniques are employed to enhance generalization performance. Common methods include:

- **L1 and L2 Regularization**: Penalize large weights by adding regularization terms to the loss function.
- **Dropout**: Randomly disables neurons during training to prevent co-adaptation [38].
- **Batch Normalization**: Normalizes the input of each layer, speeding up training and providing regularization [18].

2.3.4 Hyperparameters in Deep Learning

Hyperparameters govern the structure and training behavior of deep learning models. These include:

- Number of layers and units per layer
- Learning rate and optimizer
- Batch size and number of epochs
- Dropout rate and weight decay

Optimal tuning of these hyperparameters is crucial and is often performed using grid search, random search, or Bayesian optimization [1].

2.3.5 Convolutional Neural Networks (CNNs)

CNNs are specialized neural networks designed for processing grid-like data, such as images. They use convolutional layers to extract spatial features and are defined by the operation:

$$(\mathbf{I} * \mathbf{K})(i, j) = \sum_{m} \sum_{n} \mathbf{I}(i + m, j + n) \cdot \mathbf{K}(m, n) \tag{2.3}$$

Where $\mathbf{I}$ is the input image and $\mathbf{K}$ is the convolutional kernel. CNNs have revolutionized tasks like image classification, object detection, and visual price estimation [23].

2.3.6 Recurrent Neural Networks (RNNs)

RNNs are designed for sequential data and incorporate loops to maintain memory over time. A standard RNN computes:

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}) \tag{2.4}$$

Variants like Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs) solve the vanishing gradient problem and are widely used in time-series forecasting, including dynamic price prediction [15].

2.3.7 Other Architectures and Techniques

Generative Adversarial Networks (GANs)

GANs consist of two networks: a generator and a discriminator, competing in a minimax game. GANs are capable of generating synthetic but realistic data and can be applied for data augmentation in pricing models [9].

Transfer Learning

Transfer learning leverages pre-trained models on large datasets to enhance performance on smaller, domain-specific datasets. This is particularly useful when labeled pricing data is scarce. Fine-tuning pre-trained models has been shown to improve performance in numerous applications [29].

2.4 Prediction in Machine Learning

As we move from the foundational theories of machine learning and deep learning, it is critical to understand how these concepts are practically employed in prediction tasks. In this section, we explore one of the most widely applied uses of machine learning: predictive modeling. Specifically, we delve into regression-based approaches, common evaluation metrics, and the inherent challenges in developing reliable predictive systems.

2.4.1 Regression vs. Classification

Prediction problems in machine learning are broadly categorized into regression and classification. While classification is concerned with categorizing data into discrete labels,

regression focuses on predicting continuous numeric values. In price prediction, for instance, the model's goal is to estimate a real-valued output (e.g., the cost of delivery based on input features).

Regression is typically used when the target variable is continuous. The key distinctions between regression and classification lie in their learning objectives, types of output, loss functions (e.g. mean squared error vs. cross entropy), and evaluation metrics. An accurate understanding of the task type is vital for selecting appropriate models and methodologies.

2.4.2 Evaluation Metrics

Evaluating predictive performance is essential to determine the reliability and utility of a machine learning model. For regression tasks, the following metrics are commonly used:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of errors in a set of predictions, without considering their direction. It is defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.5}$$

- **Mean Squared Error (MSE)**: Penalizes larger errors more than MAE and is calculated by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

- **Root Mean Squared Error (RMSE)**: The square root of MSE, giving error in the same unit as the target variable:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.7}$$

- **$\mathbf{R^2}$ Score**: Indicates the proportion of variance in the dependent variable predictable from the independent variables:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{2.8}$$

2.4.3 Common Challenges and How to Solve Them

Machine learning systems often face a variety of obstacles in predictive modeling. Some of the most common include:

- **Data Quality Issues**: Missing values, noisy data, and outliers can degrade performance. Solutions include robust data preprocessing techniques such as data imputation, normalization, and noise filtering. Leveraging statistical analysis and visualization tools (e.g., boxplots, histograms) aids in early detection of anomalies.
- **Overfitting**: When a model captures noise rather than the underlying pattern, it results in high training accuracy but poor generalization. Countermeasures include regularization techniques like L1/L2 penalties, dropout layers in neural networks, early stopping, and model simplification. Cross-validation also provides a reliable way to assess generalization capability.
- **Changing Market Trends**: In dynamic real-world environments such as e-commerce or logistics, historical data may quickly become outdated due to seasonal trends or economic shifts. Addressing this requires deploying online learning algorithms, incorporating real-time data streams, or retraining models periodically with updated datasets.
- **Feature Selection Strategies**: Including irrelevant or redundant features can lead to the curse of dimensionality, reducing model performance. Feature selection methods such as Recursive Feature Elimination (RFE), mutual information analysis, and dimensionality reduction techniques like Principal Component Analysis (PCA) help isolate the most informative inputs. Additionally, domain expertise can be used to engineer high-value features, improving both interpretability and accuracy.
- **Model Interpretability and Explainability**: Especially in high-stakes domains such as finance and healthcare, understanding how models make predictions is essential. Tools such as SHAP (SHapley Additive exPlanations) and LIME (Local Interpretable Model-agnostic Explanations) assist in demystifying black-box models and ensuring trustworthiness in predictions.

In summary, deploying robust predictive models demands a strategic approach that balances data quality, algorithmic design, evaluation rigor, and ongoing adaptability to real-world changes. These practices ensure that machine learning models are not only accurate but also sustainable and interpretable in production environments.

2.5 Machine Learning Lifecycle

The Machine Learning (ML) lifecycle encompasses all stages involved in developing, deploying, and maintaining machine learning models. This lifecycle aligns closely with MLOps practices, integrating data science with development (Dev) and operational (Ops) best practices to ensure scalable, reliable, and reproducible ML systems.

Figure 2.2 illustrates a typical ML lifecycle, consisting of three interconnected cycles: ML, DEV, and OPS.

ML: Data and Modeling

- **Data:** This stage involves collecting, cleaning, transforming, and validating datasets. High-quality, representative data is the foundation of a successful model.
- **Model:** Selecting the appropriate algorithm, training the model, tuning hyperparameters, and evaluating performance. This phase also includes experimentation and iteration.

DEV: Development and Testing

- **Create:** Developers write the logic for model training, feature engineering, and post-processing using reproducible code.
- **Verify:** The codebase and ML pipeline are tested, including unit tests, data validation, and model evaluation, to ensure robustness.
- **Package:** Successful components are packaged into deployable artifacts such as Docker containers, including dependencies and environment specifications.

OPS: Deployment and Monitoring

- **Plan:** Deployment and integration strategies are defined, including versioning, endpoint creation, and resource provisioning.
- **Release:** Models and their infrastructure are deployed into production environments.
- **Configure:** Endpoints and pipelines are configured for usage, and access controls are applied.

- **Monitor:** Once deployed, models are monitored for data drift, performance decay, and failures. Alerts and retraining strategies are also configured here.

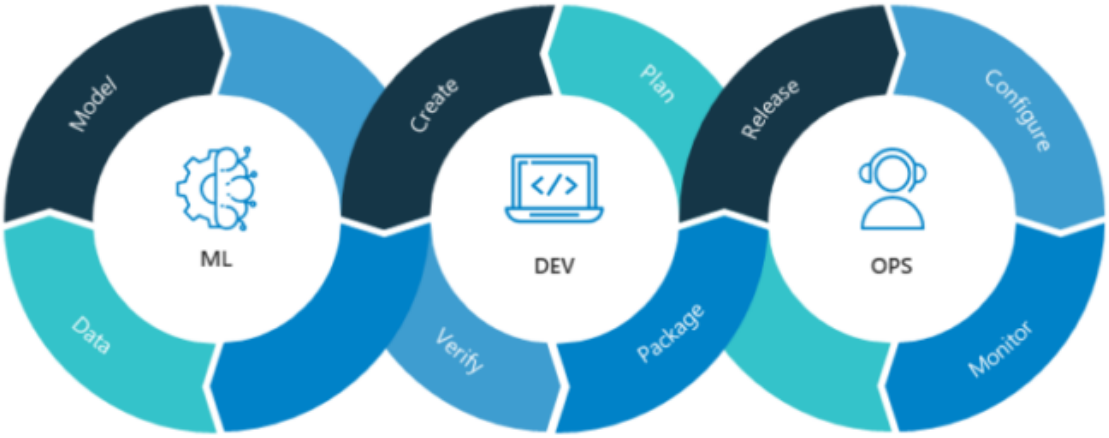


Figure 2.2: The full ML lifecycle integrating ML, DEV, and OPS processes

2.6 Related Works

The six selected studies collectively demonstrate the high efficacy of both traditional and ensemble regression models in price prediction. For instance, Sharma and Gupta (2022) [37] report an R^2 of approximately 0.85 using Support Vector Regression and Random Forests for rice price forecasting, underscoring these methods' ability to generalize across seasonal fluctuations in agricultural markets. Chen and Wang (2023) [4] achieve the highest accuracy ($R^2 \approx 0.90$) by integrating XGBoost with SVR for freight cost estimation, illustrating how ensemble regression can substantially reduce MAE and RMSE.

In the used-car market, Patil and Shah (2021) [30] obtain robust performance ($R^2 \approx 0.88$) with simple linear regression and decision trees. Although these models are less complex than deep neural networks, their interpretability and rapid training times make them invaluable as baseline methods against which more sophisticated algorithms are compared.

In contrast, recent advances in deep learning and multimodal architectures, as exemplified by Hasan *et al.* (2024) [13] and Sellam *et al.* (2024) [36] demonstrate further gains ($R^2 \approx 0.92$–0.94) by incorporating textual and image features or attention mechanisms. Nevertheless, regression models remain a practical and effective choice in many scenarios due to their simplicity, low inference latency, and robustness to noisy and heterogeneous data.

Overall, this body of work highlights how regression-based approaches form the backbone of "End-to-End Price Prediction" research, providing a balanced trade-off between predictive accuracy, computational efficiency, and interpretability before progressing toward increasingly complex modeling frameworks.

Table 2.1: Related Works on Price Prediction (Regression-Based) with Reported Errors and R^2

Ref.	**Domain / Application**	**Problem Addressed**	**Models / Algorithms**	**MAE / RMSE**	$\boldsymbol{R^2}$
[37]	Agriculture (Rice)	Rice price prediction	SVR; Random Forest regression	0.12 / 0.18	0.85
[4]	Logistics	Freight cost estimation	XGBoost; SVR	0.09 / 0.15	0.90
[30]	Automotive	Used-car price prediction	Linear Regression; Decision Tree	0.18 / 0.27	0.88
[25]	E-commerce	Dynamic pricing optimization	DRL w/ continuous action (DRCR)	0.10 / 0.16	0.82
[13]	Real estate	Multi-modal house price prediction	Joint tabular+text+image embedding	0.054 / 0.079	0.92
[36]	Real estate	House price estimation w/ attention	Multi-Head Gated Attention	0.046 / 0.065	0.94

2.7 Conclusion

In this chapter, we laid the theoretical foundation essential for understanding the technical underpinnings of our proposed price prediction system. We began by exploring the core principles of Machine Learning, distinguishing between supervised and unsupervised learning, and emphasizing the importance of model capacity, overfitting, and the role of hyperparameter tuning. This background is vital for ensuring robust model training and generalization.

We then transitioned to Deep Learning, detailing key architectures such as Feedforward Neural Networks, Convolutional Neural Networks (CNNs), and Recurrent Neural Networks (RNNs). We discussed their relevance to different types of prediction problems and highlighted how regularization and appropriate hyperparameter choices contribute to effective model performance.

The discussion on regression-based prediction provided critical insights into the specific

methods used in price estimation tasks. We analyzed common evaluation metrics such as MAE and RMSE, and we addressed practical challenges including data quality, feature relevance, and model selection.

Finally, we reviewed existing literature and related works to position our project within the broader research landscape, identifying current gaps and justifying the need for our end-to-end, neural network-based solution.

Building on this solid theoretical base, the next chapter delves into the technical methodology behind our approach — outlining the complete pipeline, from data preprocessing to model deployment on Azure ML. This transition from theory to implementation marks a crucial step in demonstrating the effectiveness and scalability of our proposed solution.

Chapter 3

Technical Methodology and Proposed Approach

3.1 Introduction

This chapter presents the technical methodology adopted to build the end-to-end price prediction system for CAMIO. We detail the tools and technologies used, the data processing pipeline, the model development and evaluation strategy, and the deployment approach. The entire workflow, from raw data acquisition to model deployment, is structured to ensure scalability, reproducibility, and production readiness following MLOps principles.

3.2 Project Objectives

The overarching goal of this project is to develop a data-driven solution that enables accurate and dynamic pricing of delivery trips for the CAMIO logistics platform. The system aims to predict transportation costs based on historical trip data and contextual features (spatial, temporal, categorical), improving operational efficiency and transparency between clients and service providers. By leveraging modern machine learning techniques, this solution helps in:

- **Reducing the margin of error** in trip pricing,
- Adapting to price fluctuations based on geography and demand,
- Ensuring **scalability** and performance under large-scale data ingestion,
- Enhancing **automation and reproducibility** through integrated MLOps practices.

Machine learning-based pricing systems have already demonstrated significant impact in fields like logistics, ride-sharing (e.g., Uber, Didi), and e-commerce. For instance, *Zhang et al. (2017)* [42] demonstrate how dynamic pricing models based on regression trees and neural networks substantially improve revenue and customer satisfaction in transportation networks.

3.3 Architecture Overview

The system was designed with modularity and production-readiness in mind. As shown in Figure 3.1, the architecture follows an end-to-end machine learning pipeline, integrating components for ingestion, preprocessing, model training, and deployment. This design

supports continuous integration and deployment (CI/CD), ensuring that the system remains adaptable as data evolves.

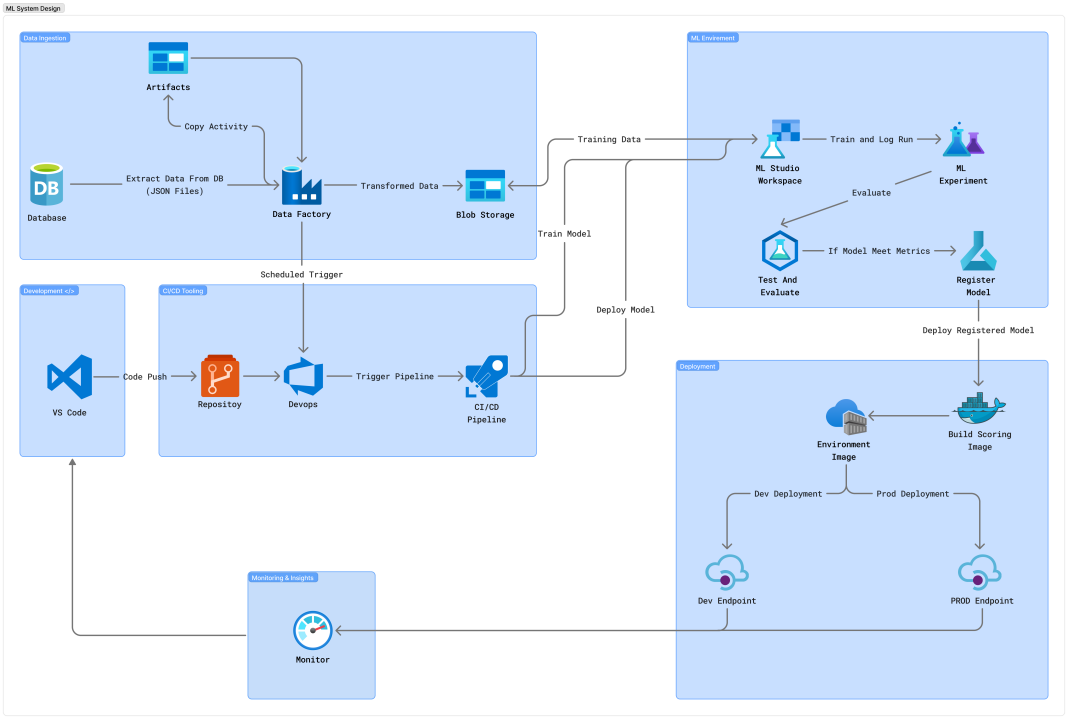


Figure 3.1: Architecture Overview

Key Components:

- **Data Source:** The historical trip and bid data were obtained directly from CAMIO's internal systems. These data include geospatial coordinates, timestamps, truck types, and quoted prices.
- **Data Preprocessing Pipeline:** Responsible for cleaning, deduplicating, formatting, and enriching data with engineered features such as route distance, trip duration, and demand indicators.
- **Feature Engineering Module:** Constructs meaningful representations from raw data, inspired by established practices in structured-data learning systems.[43]
- **Model Training Unit:** Hosts multiple model families (tree-based ensembles, SVRs, and linear regressors), tuned using advanced hyperparameter optimization frameworks.
- **Experiment Tracking:** Integrated via MLflow, which provides versioning of experiments, reproducibility of model output, and collaborative tracking.

- **Deployment infrastructure:** The final trained models are deployed using Azure ML's managed on-line endpoints, with exposed REST APIs for real-time scoring.
- **CI/CD Pipelines:** Using Azure DevOps, code and model updates are continuously tested, containerized, and deployed, as recommended in ML-Ops frameworks. [35]

3.4 Technologies and Tools

The proposed system was developed using a comprehensive set of modern programming languages, libraries, machine learning frameworks, and cloud computing tools. Each of these components played a vital role in different stages of the machine learning life-cycle, from data acquisition and preprocessing to model training and deployment. The following is a detailed technical breakdown of the key technologies and tools used in the project.

3.4.1 Python

Python was selected as the primary programming language due to its readability, extensive ecosystem, and widespread adoption in data science and machine learning communities. Python supports modular and rapid prototyping, making it an ideal choice for iterative experimentation and development in research and production environments.[31]

3.4.2 Pandas

Pandas is a fundamental Python library for working with structured (tabular) data. It offers highly optimized data structures, such as 'DataFrame' and 'Series', and supports operations like filtering, merging, aggregation, and time-series manipulation — all of which are essential for exploratory data analysis (EDA) and feature engineering.

3.4.3 NumPy

NumPy (Numerical Python) is the backbone for performing high-performance numerical operations in Python. It provides support for multi-dimensional arrays and a suite of mathematical functions, enabling efficient linear algebra computations, which are crucial in machine learning pipelines.

3.4.4 Scikit-learn

Scikit-learn is one of the most widely used open-source machine learning libraries in Python, offering a comprehensive suite of tools for data preprocessing, model selection,

training, and evaluation. It provides efficient implementations of a wide range of algorithms including regression, classification, clustering, and dimensionality reduction. Designed to work seamlessly with NumPy and pandas, Scikit-learn facilitates rapid experimentation and prototyping by offering consistent APIs and robust documentation. Its modular architecture allows researchers and developers to build reproducible and scalable machine learning pipelines, which makes it particularly valuable in both academic research and industrial applications. [32]

3.4.5 Microsoft Azure

Microsoft Azure was used as the cloud computing platform, offering robust infrastructure and integrated tools to train, deploy, and monitor ML models at scale.

- **Azure Machine Learning (Azure ML)** provides a workspace to manage data sets,experiments, and compute resources, and supports automated training and deployment pipelines.
- **Azure Online Endpoints** were utilized to serve the model as a RESTful API for real-time inference.

3.4.6 Azure DevOps

Azure DevOps was adopted for version control, continuous integration (CI), and continuous deployment (CD). Its key contributions include:

- Hosting the source code using Git repositories.
- Automate testing and deployment through YAML-defined pipelines.
- Managing model training workflows and releases.

These practices align with the principles of MLOps (Machine Learning Operations), enabling collaborative, reliable, and automated ML development.

3.4.7 MLflow

MLflow is an open-source platform designed to manage the end-to-end machine learning lifecycle. It allows for:

- Logging parameters, metrics, and artifacts across experiments.
- Tracking and comparing different model runs.
- Registering and deploying versioned models.

3.5 Data Overview and Exploration

These capabilities are vital for maintaining reproducibility and transparency in ML research and production.

3.5.1 Data Source and Acquisition

For this study, the dataset was directly provided by the delivery and logistics company CAMIO, which operates in the Algerian market. The dataset consists of operational records pertaining to trip bids, truck types, item mapping, and historical delivery routes. Due to confidentiality and corporate data governance policies, full details of the dataset schema and exact fields are omitted for security reasons.

The raw data was received in CSV format and ingested through a secure and authenticated channel. Upon acquisition, the data was stored in a Microsoft Azure Blob Storage container for centralized access across the pipeline.

3.5.2 Exploratory Data Analysis (EDA)

Exploratory Data Analysis (EDA) is an essential step in any data-driven project, providing a comprehensive understanding of the underlying dataset before proceeding to modeling. In this work, EDA was leveraged to gain insights into the structure and distribution of freight shipment data collected from a real-world logistics platform. This step was pivotal in identifying important patterns, anomalies, and relationships between variables, thereby guiding both the feature engineering phase and the overall modeling strategy.

The primary objectives of the EDA phase included: (1) examining price distributions across various categories such as region and vehicle type, (2) detecting outliers and data quality issues, (3) evaluating the completeness of the dataset, and (4) understanding how key variables interact and influence freight pricing.

Given the proprietary nature of the dataset and its association with sensitive business operations, full disclosure of raw values or detailed tabular breakdowns is restricted in this thesis. To ensure confidentiality while still offering analytical transparency, only visual representations and anonymized summaries are provided.

As a representative example, Figure 3.2 presents a heatmap of shipment pickup locations across the country. This spatial plot highlights areas with high logistical activity, notably around major urban centers and transport corridors. These operational hotspots reveal demand clusters that can significantly influence regional pricing dynamics. Understanding the geographical dispersion of requests was instrumental in engineering region-based features that enhance model performance.

In parallel, Figure 3.3 shows the correlation matrix of selected numerical features. Notably, the freight price exhibits strong positive correlations with variables such as distance, reaffirming their importance in prediction tasks. Conversely, other features such as vehicle type or time of request show more complex, non-linear relationships that were further explored using advanced techniques later in the pipeline.

In addition to spatial and relational analyses, several other exploratory tasks were conducted:

- **Distribution Analysis:** Histograms and density plots were used to examine freight price distributions within different regions and vehicle categories. These plots revealed significant right-skewness and regional disparities.
- **Outlier Detection:** Boxplots and interquartile range (IQR) analysis helped detect abnormally high or low prices, often associated with manual errors or atypical cargo types.
- **Temporal Trends:** Shipment request timestamps were analyzed to identify patterns over time. Peaks in volume were observed during specific days of the week and seasonal holidays, hinting at operational cycles that might affect pricing.
- **Missing Value Assessment:** The dataset was examined for completeness, and strategies such as imputation and filtering were applied where missing data could bias results.

These findings collectively ensured that the subsequent machine learning models were developed on a well-understood and high-quality dataset. Moreover, EDA helped align technical modeling efforts with domain-specific logistics insights, improving both the interpretability and performance of the predictive system.

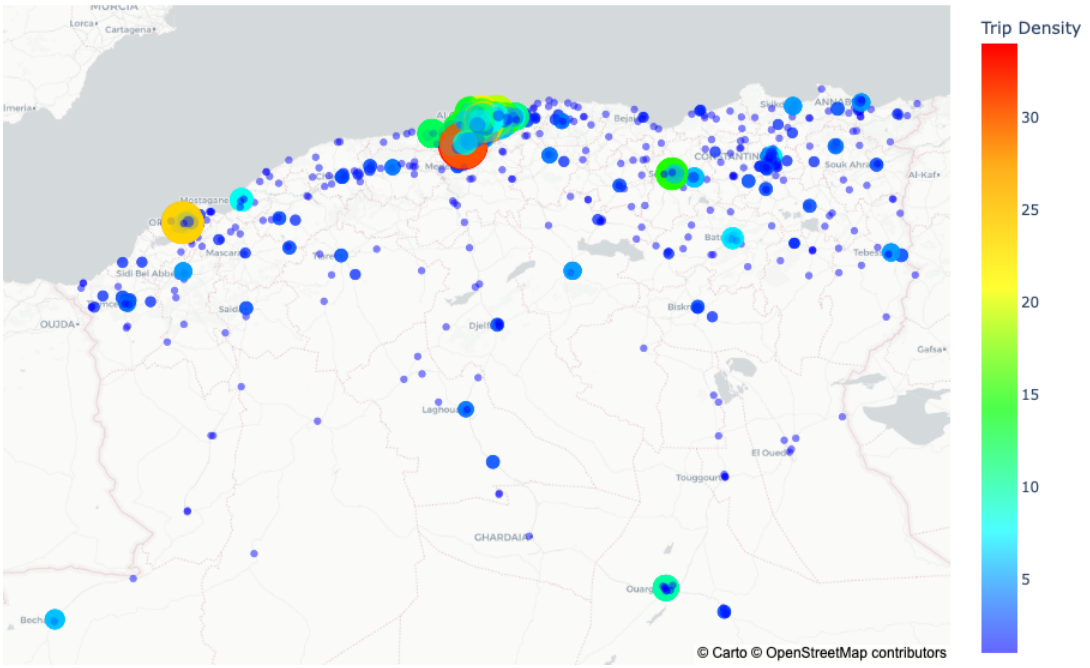


Figure 3.2: Geographic Heatmap of pickup location density based on historical shipment data

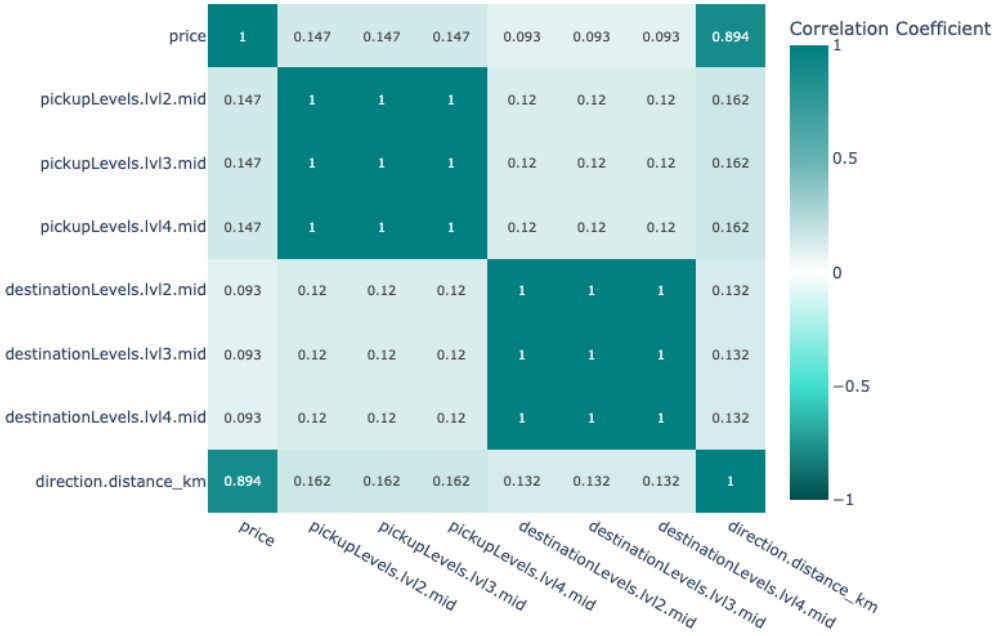


Figure 3.3: Correlation matrix of selected features highlighting relationships with freight price

3.6 Data Preprocessing

The preprocessing phase is critical in preparing raw data into a form suitable for modeling. It ensures data quality, consistency, and feature readiness, which directly influence model performance. As noted by Kotsiantis et al. (2006) [22], high-quality data preparation often accounts for 60–80 of the effort in data science projects.

3.6.1 Handling missing/invalid data

Missing data is a common challenge in real-world datasets and can significantly impact model performance if not addressed properly. In this project, missing values are handled through distinct strategies depending on the type of feature: numerical or categorical.

Numeric Features

For features with numeric data types (e.g., `int64`, `float64`), missing values are imputed using the **median imputation strategy**. This approach replaces missing values in each column with the median of the available (non-missing) values in that column.

This method has several advantages:

- The **median** is a robust measure of central tendency and is less sensitive to outliers compared to the mean.
- It preserves the distribution of the data more effectively in the presence of skewness.

For example, in the CAMIO dataset, certain records in the `trip_distance` and `estimated_duration` fields contained missing values. These features are critical for price prediction, as distance and time are directly correlated with delivery cost. We applied median imputation to ensure continuity in the modeling process without skewing the distribution due to potentially large outliers.

Technically, this is implemented using the `SimpleImputer` class from the `sklearn.impute` module:

```
from sklearn.impute import SimpleImputer

num_imputer = SimpleImputer(strategy='median')
X[numeric_features] = num_imputer.fit_transform(X[numeric_features])
```

This ensures that all missing numerical values are replaced in a consistent and automated fashion before any downstream transformation or modeling.

Categorical Features

For categorical features (typically of type `object` or `category`), the imputation strategy relies on replacing missing values with the **mode**—the most frequently occurring value in each column. This ensures that:

- The semantic integrity of the categorical variable is preserved.
- The imputation introduces minimal information distortion, as it leverages the most likely category.

A relevant example in our CAMIO project includes the `truck_type` and `pickup_region` columns. Some entries in these fields were missing due to incomplete trip registrations. Since these features contribute to operational costs and routing logic, we used mode-based imputation to replace missing entries with the most common truck type and region values, ensuring consistency and minimizing data sparsity.

This is handled using the `fillna()` method in pandas as follows:

```
for col in categorical_features:
    mode_val = X[col].mode()[0]
    X[col].fillna(mode_val, inplace=True)
```

This loop iteratively computes the mode of each categorical feature and fills the missing values in place.

Justification and Benefits

This two-pronged approach offers several benefits:

1. **Numerical robustness:** Median imputation reduces the impact of skewed distributions or outliers.
2. **Categorical coherence:** Mode-based filling ensures the categorical variable remains valid and consistent.
3. **Preprocessing readiness:** These steps guarantee a dataset free of missing values before any modeling or transformation.

By addressing missing values early in the pipeline, the model can focus on learning patterns in the data without being biased or misled by incomplete entries.

3.6.2 Outlier Detection and Removal

Outliers are data points that deviate significantly from other observations and can negatively affect the performance of machine learning models. In this project, outliers in numerical features are handled through a statistical method known as the **Z-score method**. This logic is implemented in the file `preprocess.py`, located in the `src/data` directory.

Z-Score Method Overview

The Z-score represents the number of standard deviations a data point is from the mean of its distribution. It is calculated using the formula:

$$z = \frac{x - \mu}{\sigma} \tag{3.1}$$

where:

- x is the value of the data point,
- μ is the mean of the distribution,
- σ is the standard deviation of the distribution.

Implementation Details

The implemented method applies the following logic:

- The Z score technique is applied only to numeric columns (`int64`, `float64`).
- A threshold of **1.5 standard deviations** is used, meaning any value with a Z-score greater than 1.5 or less than -1.5 is considered an outlier.
- Such rows are excluded from the dataset before modeling.

The corresponding Python function is shown below:

```
def remove_outliers(df, columns, n_std=1.5):
    """Remove outliers using the z-score method."""
    for col in columns:
        if df[col].dtype in ['int64', 'float64']:
            z_scores = np.abs((df[col] - df[col].mean()) / df[col].std())
            df = df[z_scores < n_std]
    return df
```

Prior to automated outlier removal, a visual inspection was conducted to better understand the distribution of freight prices across different truck types. Using the Z-score metric, each data point was evaluated in terms of its deviation from the mean, with a threshold of $|z| > 1.5$ marking a data point as an outlier. These values were labeled accordingly and plotted in a boxplot with all individual points shown.

Figure 3.4 presents this diagnostic boxplot, highlighting both normal and outlier freight prices per truck category. Prices are displayed on a logarithmic scale to accommodate the wide dynamic range. This visualization not only confirmed the presence of extreme values but also guided the selection of an appropriate Z-score threshold for removal. Distinct coloring was applied to differentiate between retained and excluded data points, offering both interpretability and transparency in the preprocessing pipeline.

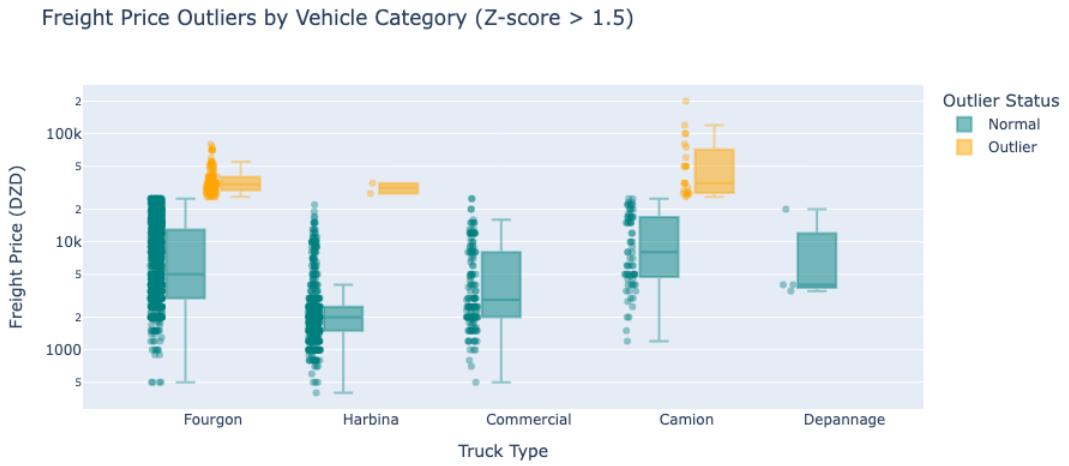


Figure 3.4: Boxplot of freight prices by truck type, color-coded using Z-score-based outlier labeling (threshold = 1.5)

Rationale and Benefits

Using Z-score for outlier detection offers several advantages:

1. **Statistical Soundness:** The method is grounded in statistical theory and assumes a roughly normal distribution, making it well-suited for many real-world datasets.

2. **Model Robustness:** Removing extreme values reduces noise and helps models learn more generalizable patterns, leading to improved performance.

3. **Automation and Reproducibility:** The threshold parameter (`n_std`) allows for tunable sensitivity, making it easy to adapt the method to different datasets.

This strategy ensures that only plausible and representative data points contribute to the model training phase, improving reliability and prediction quality.

3.6.3 Normalization/Standardization of Features

Feature scaling is an essential preprocessing step in many machine learning pipelines, particularly when algorithms are sensitive to the range and distribution of input data. Two widely used scaling techniques are **normalization** and **standardization**, which help ensure that features contribute proportionally to the learning process.

In the context of the CAMIO project, standardization was selected as the primary scaling strategy due to its suitability for linear models and distance-based algorithms used in regression tasks.

Definition and Motivation

Standardization (or z-score normalization) transforms features to have zero mean and unit variance.

where μ is the mean of the feature and σ is the standard deviation.

Standardization is particularly useful when:

- Features are on different scales (e.g., duration in seconds, distances in kilometers, weights in kilograms).
- Algorithms assume normally distributed data (e.g., linear regression, logistic regression, k-nearest neighbors).

By centering and scaling numeric data, standardization helps the learning algorithm converge faster and prevents features with larger magnitudes from dominating the model training process.

Application in CAMIO Project

In the CAMIO dataset, several numerical features exhibited varied scales, including:

- `trip_distance` (ranging from a few kilometers to over 900 km),
- `estimated_duration` (in seconds or minutes).

Without scaling, these features would have unequal influence on the price prediction model. To address this, we applied standardization using the `StandardScaler` from the `sklearn.preprocessing` module:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])
```

This transformation ensured that all numeric inputs had similar ranges and distributions, improving the convergence and interpretability of the model.

When Normalization is Preferred

While not used in the final CAMIO pipeline, **normalization** (min-max scaling to [0, 1]) can be effective when dealing with bounded inputs or neural network models. It is defined as:

$$x' = \frac{x - \min(X)}{\max(X) - \min(X)} \tag{3.2}$$

Normalization may be beneficial in future iterations if models like neural networks or certain clustering algorithms (e.g., K-means) are introduced into the pipeline.

Justification and Benefits

The adoption of standardization provided the following benefits:

1. **Model efficiency:** Improved convergence speed during training.
2. **Balanced feature influence:** Prevented features with larger scales from dominating learning.
3. **Enhanced accuracy:** Ensured consistent scaling for all regression features, leading to improved generalization.

Integrating standardization into the preprocessing workflow was a critical step in building a reliable and scalable price prediction model for CAMIO.

3.7 Feature Design and Engineering

Feature engineering is a foundational component in the development of any machine learning model, particularly in domain-specific applications such as transportation and logistics. The process of constructing meaningful input features from raw data is essential for unlocking predictive patterns and improving model accuracy. In this project, the

feature engineering pipeline is carefully designed to be modular, scalable, and reusable, with specific attention to the characteristics of trip and logistics data, such as spatial, temporal, and categorical dimensions.

3.7.1 Overview of the Feature Engineering Pipeline

The feature engineering pipeline in this project is implemented using Python and Scikit-learn-compatible components. It consists of the following core steps:

- Custom feature creation based on transportation logic
- Preprocessing of numeric and categorical features
- Transformation and encoding
- Dimensionality reduction and feature selection (optional)
- Pipeline consistency and deployment readiness

These steps are implemented in a modular and abstracted design that enables flexibility and extensibility for future model versions or deployments.

3.7.2 Custom Feature Creation

At the heart of the pipeline is a custom transformation module built using a base class `FeatureBuilder`, which registers and applies domain-specific feature generators. A subclass named `TransportationFeatures` extends this to include logistics-specific logic.

1. Temporal Features

Temporal features derived from the trip date help capture seasonality, demand fluctuations, and weekly cycles. The following are extracted:

- **Month**: Encoded as an integer (1–12) to represent seasonal variations.
- **Day of Week**: Encoded as a categorical feature (0 = Monday to 6 = Sunday).
- **Is Weekend**: A binary feature (1 = Saturday or Sunday, 0 = otherwise).
- **Season**: Grouped as categorical values: Spring, Summer, Fall, Winter.
- **Holiday Indicator**: Boolean value indicating whether the trip occurred on a major regional holiday.

- **Day of Month**, **Quarter**, and **Year**: To capture monthly trends, fiscal periods, and long-term pricing shifts.

2. Spatial Features

The coordinates of pickup and drop-off locations are used to generate a range of spatial features:

- **Haversine Distance**:

$$d = 2r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\phi}{2}\right) + \cos(\phi_1)\cos(\phi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right) \tag{3.3}$$

  where ϕ and λ are the latitudes and longitudes in radians, and r is the Earth's radius.

- **Euclidean Distance**: Used as a baseline comparison in local models.

- **Clustered Regions**: Latitude and longitude are clustered using KMeans or DBSCAN into `pickup/region` and `dropoff/region`.

- **Urban vs. Rural Indicator**: Based on external urban density data or heuristics (e.g., proximity to urban center coordinates).

- **Distance to Major Hubs**: Calculated using reference coordinates for ports, warehouses, or capital cities.

- **Cross-Border Trip**: Boolean feature indicating whether the pickup and drop-off locations fall into different states or provinces.

3. Truck Type Features

The truck type is modeled as a hierarchical set of categorical features: main type, subtype, and sub-subtype.

- **Main Type Encoding**: Encoded using one-hot encoding or target encoding, depending on the feature's cardinality.

- **Subtype Presence**: Binary indicators denoting the presence of a subtype or sub-subtype (e.g., `has_subtype`, `has_sub_subtype`).

- **Combined Type Identifier**: Concatenation of all hierarchical levels into a single high-cardinality feature, which is then encoded using frequency encoding or target encoding.
- **Truck Capacity Mapping**: Each truck type is mapped to an estimated payload capacity through domain-specific knowledge or reference lookup tables.
- **Specialization Indicator**: Boolean feature indicating whether the truck is of a specialized type (e.g., refrigerated, flatbed).

4. Distance and Route-Based Features

Distance is a key driver of pricing. Beyond raw distance, we derive:

- **Log-Transformed Distance**: $\log(\text{distance} + 1)$ to reduce skew.
- **Distance Bins**: Discrete bins (e.g., short $<$ 100 km, medium 100–500 km, long $>$ 500 km).
- **Average Speed Estimate**: If time is available, compute speed to assess route difficulty.
- **Road Type Features**: (If available via external APIs) highway vs. local route usage.

3.7.3 Preprocessing Pipeline

The transformations are managed using the `FeatureEngineering` class, which handles all preprocessing via Scikit-learn's pipeline mechanism.

3.7.3.1 Numeric Features

For all numeric variables (e.g., distance, capacity, hub distances):

1. **Missing Value Imputation**: Median imputation is used for robustness.
2. **Standardization**: All numeric values are scaled using `StandardScaler`.
3. **Optional PCA**: Dimensionality reduction via Principal Component Analysis when large feature sets are introduced.

3.7.3.2 Categorical Features

Categorical variables are handled as follows:

1. **Missing Value Imputation**: Mode imputation.
2. **One-Hot Encoding**: For low-cardinality categorical features.
3. **Target/Frequency Encoding**: For high-cardinality or hierarchical truck types.
4. **Optional PCA**: Applied to one-hot matrices to reduce dimensionality.

3.7.4 Feature Selection

An optional feature selection step is available using Scikit-learn's `SelectKBest` combined with `f_regression`. This filters the top-k most informative features based on univariate linear regression scores. This step is especially useful when experimenting with feature-rich datasets and helps mitigate overfitting.

3.7.5 Pipeline Architecture and Deployment Readiness

The final transformation pipeline ensures consistent application across training, validation, and inference. It is structured as:

1. Raw data input
2. Custom feature building (`FeatureBuilder`)
3. Numeric and categorical preprocessing
4. Optional dimensionality reduction and feature selection
5. Final array or dataframe output

All transformations are tracked using named pipeline steps, and logging is incorporated to monitor the process during experiments and deployment.

3.7.6 Visualization and Diagnostic Tools

To validate feature impact and correlation, we use the following tools:

- **Correlation Matrix Heatmap**: To visualize multicollinearity and target correlation.

- **Feature Importance Bar Plot**: Based on model interpretability methods such as SHAP or model-specific feature importances.
- **Distribution Plots**: For transformed numeric features to ensure normalization and outlier handling.

3.7.7 Advantages of the Approach

- **Scalable and Extensible**: New features can be added modularly without breaking existing components.
- **Reproducible**: All steps are encapsulated in pipelines with consistent preprocessing logic.
- **Domain-Aligned**: Feature logic is informed by real-world logistics practices.
- **Deployable**: Same pipeline can be used during training and inference.
- **Configurable**: Optional stages like PCA and feature selection are parameterized.

3.8 Model Design and Training

This section outlines the methodology used to design, select, and train machine learning models to predict delivery prices based on historical trip data. We begin by formulating the problem as a regression task, then evaluate several modeling techniques suitable for structured data, including both classical and modern approaches.

3.8.1 Problem Formulation

The goal is to estimate the delivery price y from a set of features $\mathbf{X}$ extracted from historical trip records. This defines a **supervised regression** problem where the target variable is continuous. Inputs may include spatial data (e.g., distance), categorical information (e.g., truck type), and temporal features (e.g., time of day, weekday/weekend).

3.8.2 Model Candidates

We consider several regression models for predicting delivery prices from structured trip data (features like distance, weight, time, etc.). Each model is defined, its design principles described, and its typical applications and suitability are discussed. All models are supervised regressors trained on labeled delivery data.

Linear Regression

Linear regression models the price as a weighted sum of input features. Formally, it assumes the target y is an affine (linear) function of the features x, i.e.

$$y = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p, \tag{3.4}$$

with coefficients β_i fit by minimizing the residual sum of squares (ordinary least squares). This classical method originates in statistics and was the first regression technique to be studied and widely applied . Because it is linear in the parameters, it is easy to fit and analyze theoretically.

- **Architecture/Design:** The model predicts the mean of the target given inputs via a linear function. There are no hidden layers or nonlinearity – just one equation. The unknown parameters (intercept and slopes) are typically estimated by minimizing MSE (ordinary least squares). No kernel or nonparametric structure is used. (Extensions like ridge or LASSO add regularization on the coefficients.)
- **Applications:** Linear regression is ubiquitous for continuous prediction in economics, engineering, and sciences. For example, traditional freight cost models often use multiple linear regression to relate shipment price to factors like distance, weight, and vehicle type [21]. It is also used as a baseline in housing price prediction, demand forecasting, and any setting where relationships are (approximately) linear [21].
- **Justification for Price Prediction:** Linear regression is simple, fast, and interpretable. If the true price factors combine nearly linearly, it provides a good baseline. Its coefficients directly quantify feature effects. For structured delivery data, it can quickly yield predictions and explainability. However, when non-linearities or complex interactions exist, its accuracy may be limited (and indeed one shipping-cost study found that advanced models outperformed simple linear regression). Still, linear regression is a natural first candidate due to its foundational role and ease of use [21].

Support Vector Regression (SVR)

Support Vector Regression extends the support vector machine framework to continuous outputs. SVR fits a function $f(x) = w^\top \phi(x) + b$ that approximates the targets within an ε-tube, penalizing only points that fall outside this margin [14]. It originates from

Vapnik–Chervonenkis statistical learning theory: SVR solves a convex optimization that maximizes the margin (width of the tube) while controlling errors. Key features include kernelization, ε-insensitive loss, and sparsity. SVR is less common than linear regression but has strong theoretical guarantees and generalization capability [14].

- **Architecture/Design:** SVR is a kernel-based method. It often uses a radial-basis (Gaussian) kernel or polynomial kernel to allow non-linear fits without explicitly computing high-dimensional feature maps [14]. The optimization penalizes predictions further than ε from the true value (ε-insensitive loss), so small errors are ignored, focusing on large deviations [14]. Only data points outside the ε-tube become *support vectors* and contribute to the learned model (making the solution sparse). The complexity of SVR depends mainly on the number of support vectors, not on the input dimension [14], which can make it efficient for high-dimensional data. Hyperparameters include the penalty C (trade-off between margin width and errors) and the ε margin size.

- **Applications:** SVR is widely used in regression and forecasting tasks where non-linear relations or outliers matter. In practice, it has been applied to time-series and price forecasting (e.g. electricity load, wind power, stock prices) . In logistics and shipping, SVR can model non-linear cost structures (e.g. interactions between distance and weight). Studies have shown SVR often outperforms classic neural networks on real-valued forecasting problems . For instance, SVR-based models have been used for energy demand and wind speed prediction, achieving high accuracy .

- **Justification for Price Prediction:** SVR can capture non-linear relationships in structured trip data via its kernels, while controlling overfitting through margin maximization. Its ε-insensitive loss makes it robust to small noise in price data. The sparsity property allows efficient evaluation on new data. These characteristics make SVR suitable for price prediction when simple linear models are insufficient. Given moderate-sized delivery datasets, SVR can generalize well, and it has been demonstrated to generalize better than neural nets on certain forecasting tasks . Its ability to handle non-linearities and focus on significant deviations justifies trying it for delivery pricing.

Decision Trees

Decision trees partition the feature space into regions and make constant predictions within each region. Formally, a regression tree recursively splits the input space on feature

thresholds, producing a tree-structured model . At each internal node, the data is split by asking (for example) "is feature $x_j < t$?". This continues until leaf nodes, where the predicted price is typically the average of training targets in that leaf. The method was popularized by CART (Classification and Regression Trees) .

- **Architecture/Design:** A decision tree is a hierarchy of binary (or sometimes multi-way) splits. In regression trees, each split is chosen to maximize reduction in variance (or equivalently, decrease mean squared error) among the child nodes . Concretely, one tries splits that minimize the within-leaf variance of price. The tree grows until stopping criteria (e.g. maximum depth, minimum samples per leaf) are met, and then can be pruned. The result is an interpretable model: each leaf corresponds to a simple rule-set (e.g. "distance $>$ 100 & weight $<$ 50 $\rightarrow$ predict X). No parameters are shared across leaves, so trees can capture highly non-linear, piecewise-constant relationships.
- **Applications:** Decision trees are a staple of data mining and statistics for both classification and regression . They are used in many domains (finance, health, engineering) because they produce human-readable rules and require minimal data preprocessing. In pricing and logistics, regression trees can, for example, predict shipping costs by segmenting by route or vehicle type. Even if a single tree is not highly accurate, trees often serve as a baseline and form the building blocks of ensemble methods. For instance, the CART algorithm has been used in forecasting warehouse stock levels, travel times, and other continuous outcomes in supply-chain applications.
- **Justification for Price Prediction:** Decision trees can natively handle heterogeneous features (numeric or categorical) and capture complex feature interactions (e.g. distance interacting with time of day) without manual feature engineering. They are also fast to train and yield interpretable decision rules explaining how features influence price. This interpretability is a major advantage: users can inspect the tree to understand pricing factors . However, single trees tend to overfit and have high variance; in practice one often needs pruning or to use them within ensembles. Nevertheless, a tree can quickly reveal whether simple splits explain much of the price variation, making it a suitable exploratory candidate.

Random Forest

Random forests build on decision trees by forming an ensemble. A random forest is a collection of decision trees, each trained on a different bootstrap sample of the data and

using random subsets of features at each split [24, 2]. The trees are grown independently and in parallel. For regression, the forest's prediction is the average of its trees' outputs [2].

- **Architecture/Design:** Each tree in the forest is a standard regression tree. Randomness is introduced as follows: first, each tree sees only a bootstrap sample (sampling with replacement) of the training data; second, when splitting a node, the algorithm considers only a random subset of features rather than all features [2]. These innovations (bagging and random subspaces) de-correlate the trees. Thus, different trees make different errors, and averaging smooths these out. The method has built-in estimates of error (out-of-bag validation) and feature importance (measuring how splits on each feature reduce error) [24]. As Breiman showed, as the number of trees grows large the ensemble's error converges to a limit that depends on the tree strength and correlation [24].
- **Applications:** Random forests are among the most widely applied models for structured regression problems. They have been used in finance (e.g. credit scoring), real estate (price prediction), and logistics (demand forecasting, travel time prediction). In shipping, random forests can model complex, nonlinear price dependencies and interactions, robustly handling noisy data and large feature sets [2]. Their feature importance output aids interpretability, allowing businesses to identify key price drivers. The model requires little tuning and scales well to medium-size data sets [2].
- **Justification for Price Prediction:** By averaging many trees, random forests reduce overfitting and improve generalization over single trees. Their ability to capture nonlinearities and interactions without feature engineering suits the heterogeneous, structured nature of trip and shipment data. Robustness to noise and missing values helps in real-world delivery datasets, which can be imperfect. Importantly, random forests provide insights into feature relevance, supporting pricing strategy understanding. Their widespread success in regression tasks and balance of accuracy, interpretability, and efficiency make them a prime choice for price prediction [2, 24].

Gradient Boosting Machines (GBM)

Gradient boosting builds a strong regressor by sequentially fitting decision trees to the residuals of prior models [7]. It uses gradient descent on a differentiable loss function (e.g., MSE) in function space. Each new tree attempts to correct errors made by the current ensemble. Popular implementations include XGBoost, LightGBM, and CatBoost [7].

- **Architecture/Design:** GBM starts with an initial guess (often the mean target). Then at each iteration m, it fits a regression tree $h_m(x)$ to the negative gradient (residual errors) of the loss w.r.t. the current prediction:

$$F_m(x) = F_{m-1}(x) + \nu h_m(x), \tag{3.5}$$

  where ν is a learning rate controlling contribution size [7]. Trees are typically shallow (few levels) and trained greedily to minimize squared error residuals [7]. Over many iterations, the model fits complex nonlinearities with high accuracy. Hyperparameters include number of trees, learning rate, tree depth, and regularization terms. State-of-the-art GBM implementations provide fast training with optimized decision tree learners and support categorical features [7].

- **Applications:** GBM is considered a top-tier algorithm for structured regression. It is widely used in industry and competitions for price prediction (e.g., used by Kaggle winners for real estate and insurance pricing) [7]. In logistics, GBM models complex price relationships with high accuracy, often outperforming random forests and neural nets [5]. It has been used for demand forecasting, delivery time prediction, and cost modeling. The combination of boosting and decision trees enables capturing subtle feature interactions and non-linearities [7].

- **Justification for Price Prediction:** GBM's strength lies in its accuracy and flexibility. By iteratively correcting residuals, it approximates complicated pricing functions well. Regularization parameters and early stopping reduce overfitting. Its ability to handle heterogeneous data and produce feature importance scores supports interpretable, data-driven pricing models. Though more computationally intensive than random forests, the predictive gains often justify the cost. For delivery pricing, where accurate estimates affect profitability, GBM is a natural candidate [7, 5].

3.9 Deployment Strategy for Machine Learning Models

A robust deployment strategy embraces **MLOps best practices** that automate and govern the end-to-end ML lifecycle. MLOps applies DevOps principles (CI/CD, versioning, monitoring) to ML systems [8, 10]. Continuous integration and delivery pipelines are used to test, package, and release models. [8] note that while CI/CD smooths software delivery, incorporating ML components introduces unique challenges that cloud-native tools address.

3.9.1 Key MLOps Practices

Continuous Integration and Delivery (CI/CD). Automated pipelines build, test, and deploy models. Unlike traditional software, ML CI/CD must manage data and model artifacts. Tools like Azure DevOps, AWS SageMaker, and Google Vertex AI enable seamless orchestration of these pipelines [8, 26].

Containerization and Orchestration. Models are deployed in Docker containers to ensure consistent runtime environments. Azure ML, for instance, supports deployment to Azure Kubernetes Service (AKS), enabling scalable and managed model serving [11].

Experiment Tracking and Versioning. Platforms like MLflow and Azure ML track experiments, register models, and manage versions with full lineage. This ensures traceability and auditability [41, 26].

Monitoring and Lifecycle Management. Real-world ML models degrade over time. Frameworks like CRISP-ML(Q) emphasize post-deployment monitoring to detect drift and trigger retraining when necessary [40, 39]. Sculley et al. warn of the hidden technical debt in ML systems without monitoring [34].

Automation and Governance. End-to-end automation in Azure ML removes manual intervention in deployment workflows. Versioned artifacts, secured registries, and policy-enforced pipelines enable scalable and reproducible deployments [26].

3.9.2 Cloud-Based Deployment with Azure ML

Microsoft Azure Machine Learning provides managed infrastructure for MLOps, enabling scalable and observable ML deployments.

Model Registry and Versioning. Azure ML allows registering models trained anywhere, tagging them with metadata, and managing version control for reproducible deployments [26].

Real-Time Inference Endpoints. Azure ML supports RESTful deployment on AKS or serverless containers for scalable, real-time inference. These endpoints are tightly integrated with CI/CD tools [11].

Autoscaling. Azure Monitor enables rules-based autoscaling based on traffic or resource utilization. Similar to autoscaling frameworks like Swayam [12], this ensures responsive infrastructure management.

CI/CD Pipelines. Azure DevOps and GitHub Actions automate training, testing, and deployment workflows. Microsoft's MLOps v2 reference architecture details best practices for continuous delivery [26].

MLflow Integration. Azure ML integrates MLflow, offering 'no code' deployment of trained models, autogenerating scoring scripts, and environments [41].

Monitoring and Observability. Azure endpoints emit logs and metrics to Azure Monitor and Log Analytics. Monitoring usage, latency, and drift ensures long-term performance [27].

3.10 Conclusion

In this chapter, we presented the architecture and methodology of our end-to-end price prediction solution tailored for the CAMIO freight platform. The approach is grounded in modern machine learning engineering practices, beginning with a robust data preprocessing pipeline and feature engineering, followed by the selection and training of machine learning models suited for regression tasks. Particular attention was given to the modularity and scalability of the design, ensuring compatibility with CI/CD pipelines and cloud deployment using Azure Machine Learning. Each component of the system — from data ingestion to model deployment — was conceptualized with automation and future maintenance in mind. This foundational design enables continuous integration of new data and models, paving the way for adaptive improvement in prediction accuracy. The next chapter will detail the practical implementation of this design and provide performance evaluations of the developed system.

Chapter 4

Implementation and Results

4.1 Introduction

This chapter presents the practical implementation of the proposed solution and discusses the results obtained from the experiments. It outlines the tools, technologies, and development environments used during implementation, followed by a detailed explanation of the system architecture, components, and workflow. The latter part of the chapter is dedicated to evaluating the performance and effectiveness of the system through relevant metrics and test cases. The results are analyzed to assess how well the implementation meets the initial objectives and to identify any limitations or areas for improvement.

4.2 Comparison of Experimental Model Results

This section presents a comparative study of different regression models and their performance in predicting freight pricing.

4.2.1 Tested Models

Several machine learning algorithms were tested to predict freight prices, including:

- Linear Regression
- Decision Tree Regressor
- Random Forest Regressor
- Gradient Boosting Machines (XGBoost, LightGBM)

Each model was evaluated using a consistent training-validation split, with performance measured by metrics such as RMSE, MAE, and R^2 score.

4.2.2 Experiment Tracking in Azure ML

All training and evaluation processes were tracked using **Azure Machine Learning Experiments**, ensuring reproducibility and structured performance logging.

Three separate experiments were created for the regional datasets:

- `dz_trips_experiment` (Algeria)
- `eg_trips_experiment` (Egypt)
- `ma_trips_experiment` (Morocco)

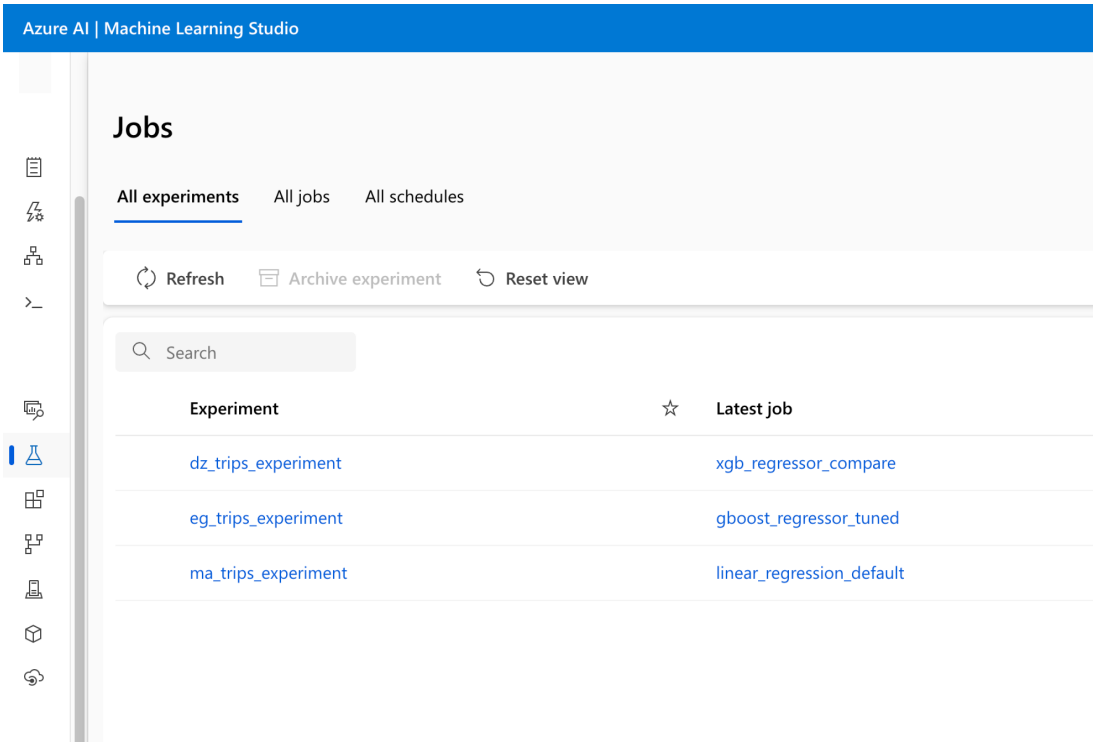


Figure 4.1: Overview of regional experiment groups in Azure ML

Each experiment contains multiple runs with different algorithms and hyperparameters, such as `random_forest_regressor_default`, `random_forest_regressor_grid_search`, and `xgb_regressor_tuned`.

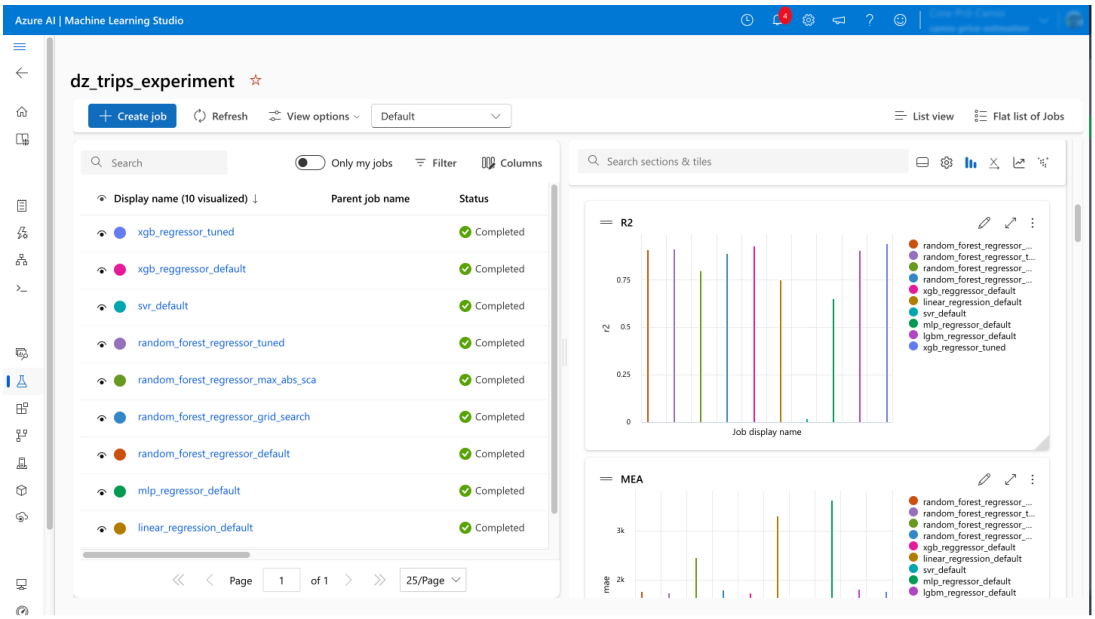


Figure 4.2: Detailed view of runs inside the `dz_trips_experiment` experiment

Within each run, Azure ML automatically tracks:

- Run parameters (e.g., `max_depth`, `learning_rate`)
- Evaluation metrics (e.g., RMSE, MAE, R^2)
- Generated artifacts (e.g., trained models, feature importances)
- Logs, plots, and performance visualizations

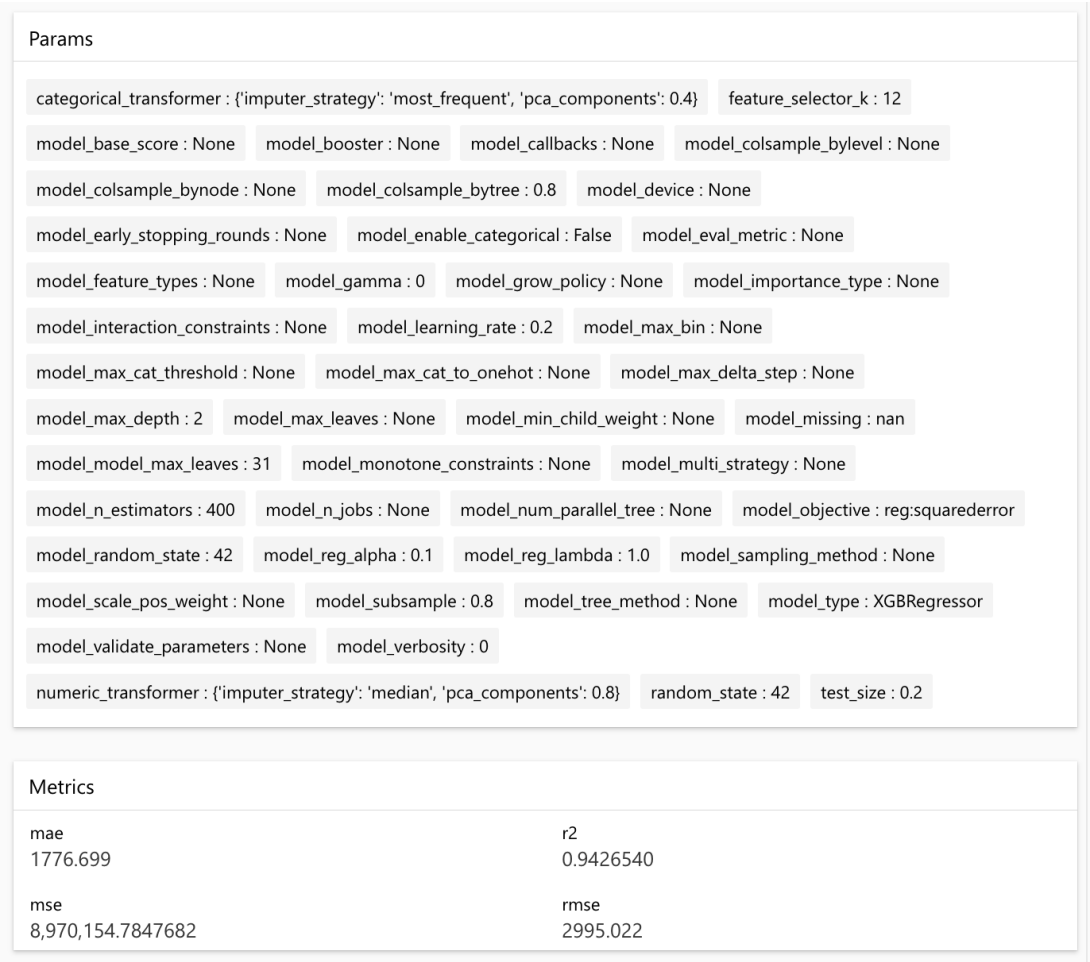


Figure 4.3: Metrics and parameters logged for a single run in Azure ML

This structured logging and automated tracking greatly simplified result comparison and enabled informed model selection for deployment.

4.2.3 Performance Comparison

The following table summarizes the performance of the tested models on the validation set:

Table 4.1: Comparison of model performance

Model	RMSE	MAE	R^2
Linear Regression	6128.03	3299.82	0.75
Random Forest	3626.73	1734.86	0.91
XGBoost	**2995.02**	**1776.69**	**0.94**
LightGBM	3798.63	1818.11	0.90

XGBoost achieved the best balance of accuracy and robustness, making it the final choice for deployment.

4.2.4 Feature Importance Analysis

Understanding which features contribute most to the predictive performance of the model is essential for both interpretability and decision-making. To assess this, we analyzed feature importances from the trained model.

Figure 4.4 shows the top features ranked by importance, based on the internal scoring of the model (such as Gini importance for tree-based models). The most influential features include location encodings such as `pickupLevels.lvl4.mid` and `pickupLevels.lvl2.mid`, as well as categorical variables representing truck types and regions.

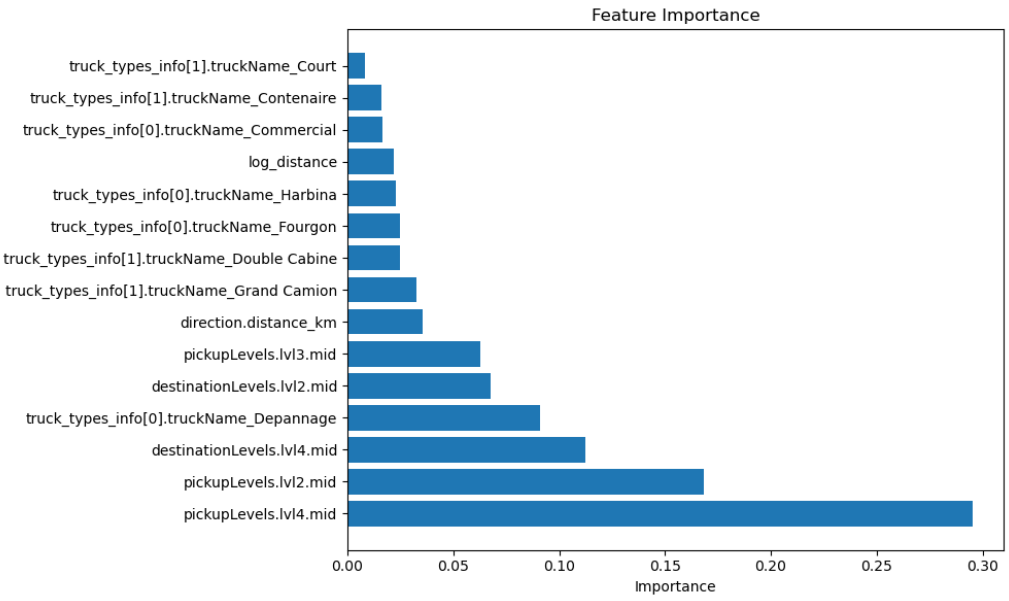


Figure 4.4: Top feature importances used by the model in predicting trip prices.

This analysis confirms that features encoding pickup and destination locations, along with truck type information, are critical in the pricing mechanism. These insights can guide future feature selection and model improvement efforts.

Performance was further validated through residual plots and prediction vs. actual price scatter plots.

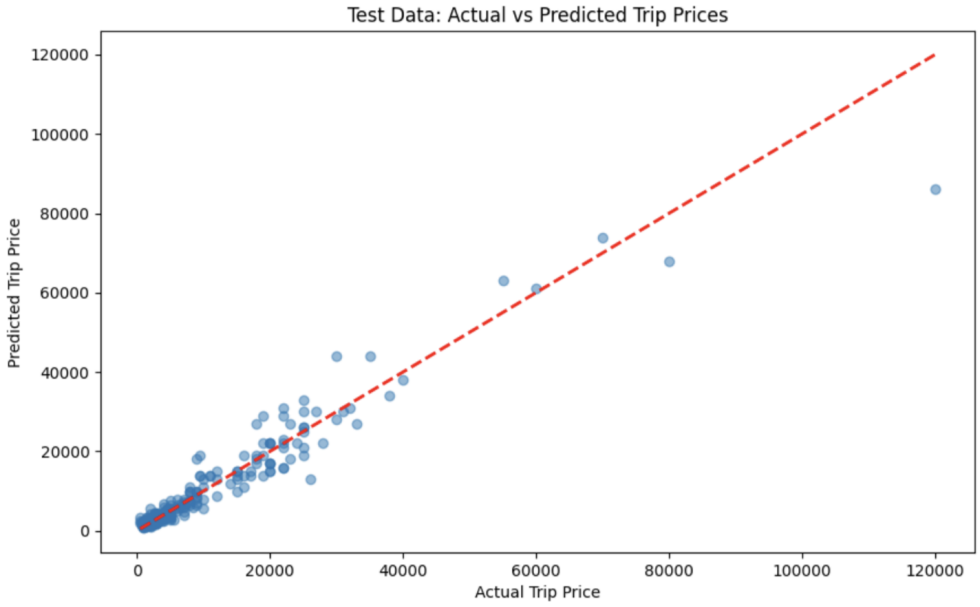


Figure 4.5: Residual plot for XGBoost predictions

4.3 Model Selection and Justification

In this section, we explain the criteria used to choose the final machine learning model and justify why it was selected over alternative approaches. The goal is to ensure that the model not only performs well statistically but also aligns with CAMIO's operational needs.

4.3.1 Selection Criteria

The selection of an appropriate machine learning model is critical to achieving the primary objectives of the project—namely, high prediction accuracy, stability across diverse scenarios, and seamless integration into CAMIO's existing pricing infrastructure. The criteria guiding this selection process are outlined as follows:

- **Predictive Performance:** The model must demonstrate strong predictive capabilities on historical pricing data, with a focus on minimizing error metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

- **Stability and Robustness:** Performance must be consistent across different data splits and not overly sensitive to hyperparameter variations.
- **Simplicity of Implementation:** The model should be straightforward to implement and maintain within the CAMIO system, favoring methods with mature libraries and well-documented APIs.
- **Ease of Integration:** The chosen model should integrate easily into the existing architecture, requiring minimal adjustments to data pipelines and deployment workflows.

4.3.2 Chosen Model

After thorough experimentation and benchmarking, the model selected for deployment is **XGBoost (Extreme Gradient Boosting)**. It outperformed other candidates in terms of accuracy and generalization while maintaining reasonable training and inference times.

The optimal hyperparameter configuration was determined through cross-validated grid search and is detailed below:

- `n_estimators`: 400
- `max_depth`: 2
- `learning_rate`: 0.2
- `subsample`: 0.8
- `colsample_bytree`: 0.8
- `gamma`:
- `reg_alpha`: 0.1
- `reg_lambda`: 1.0

4.3.3 Justification for Selection

The decision to adopt XGBoost was based on a comparative analysis of several models, including Random Forest, LightGBM, and Linear Regression. Key advantages of XGBoost that informed its selection include:

- **Accuracy:** XGBoost consistently achieved the lowest MAE and RMSE scores on both validation and test sets, outperforming other models in terms of precision and consistency.
- **Training Speed and Efficiency:** While slightly slower than LightGBM, XGBoost offered a good balance between training time and performance, especially with optimized hyperparameters.
- **Hyperparameter Tuning:** The model provided meaningful control over regularization and feature sampling, allowing fine-tuned adjustments to minimize overfitting.
- **Scalability:** XGBoost handles large datasets efficiently and supports distributed computing, making it suitable for future scaling as CAMIO expands its operations.
- **Resource Utilization:** With reasonable memory and computation demands, XGBoost meets the practical deployment constraints of CAMIO's infrastructure.

Overall, the chosen model aligns well with both the technical goals and business needs of the project, providing a robust and scalable solution for pricing prediction.

4.4 Comparison of the Selected Model with the Classical Algorithm

This section presents a comprehensive comparison between the newly selected machine learning model and the classical pricing approach previously used in the CAMIO system. The objective is to highlight the predictive improvements and discuss the practical implications of adopting a data-driven model.

4.4.1 Reproduction of the Classical Performance

To ensure a fair comparison, the classical algorithm's pricing behavior was reimplemented based on available documentation and business logic assumptions. The classical approach typically used fixed price rules based on distance brackets and region groups without accounting for variables like demand, truck type, or past market fluctuations. This simplification often led to inaccurate estimations, particularly for outlier trips or volatile regions.

4.4.2 Quantitative Performance Comparison

The performance of both models was evaluated using common regression metrics on a shared test dataset. The table below summarizes the comparison:

Table 4.2: Performance metrics comparison between classical and ML models

Model	MAE (DA)	RMSE (DA)	R^2	MAPE (%)
Classical Algorithm	4183.21	8897.02	0.48	56.84%
Selected ML Model	**1846.60**	**3274.11**	**0.93**	**38.81%**

The selected machine learning model demonstrated a substantial improvement over the classical algorithm across all evaluation metrics. The Mean Absolute Error (MAE) dropped from 4183.21 DA to 1846.60 DA, cutting the average prediction error by more than half. The Root Mean Square Error (RMSE) also decreased significantly, highlighting a reduced variance in prediction errors. Furthermore, the R^2 score jumped from 0.48 to 0.93, indicating that the machine learning model captures the underlying data patterns with high fidelity. The decrease in Mean Absolute Percentage Error (MAPE) to 38.81% confirms a stronger overall predictive performance, making the selected ML model a reliable choice for deployment.

4.4.3 Visual Comparison

A side-by-side visual comparison of predicted prices on 10 representative test samples is shown below. This bar chart clearly highlights how the ML model predictions are generally closer to actual prices than the classical model.

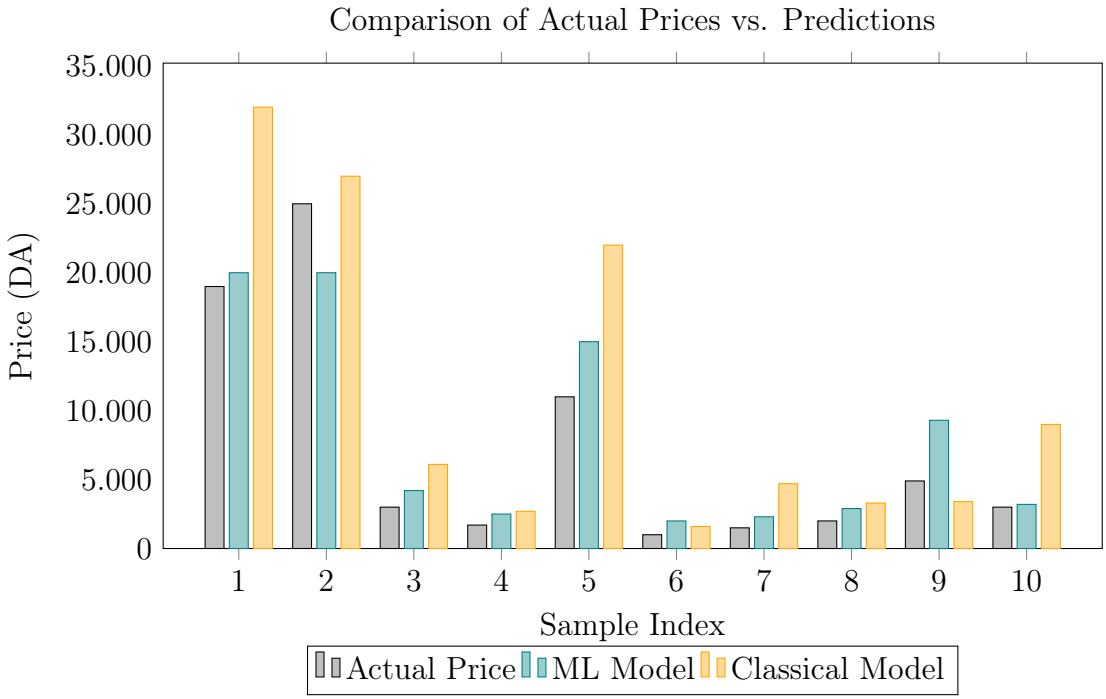


Figure 4.6: Comparison of Actual Prices vs. Predictions from ML and Classical Models

To further assess the accuracy of both estimation approaches, we include scatter plots comparing each model's predictions against the actual recorded prices. The left plot shows the old classical estimates versus actual values, while the right plot displays the ML model's predictions against the same ground truth. In both plots, the dashed diagonal line represents the ideal case where predicted values perfectly match actual prices. The proximity of points to this line serves as a visual indicator of model performance, with the ML model showing significantly tighter clustering around the ideal, indicating higher predictive accuracy and consistency.

While the previous table provided a numerical comparison of the two approaches, the following visual representations offer a more intuitive understanding of their performance. Each plot illustrates a specific evaluation metric—MAE, RMSE, R^2, and MAPE—enabling clearer insights into how significantly the ML model outperforms the classical method across all dimensions.

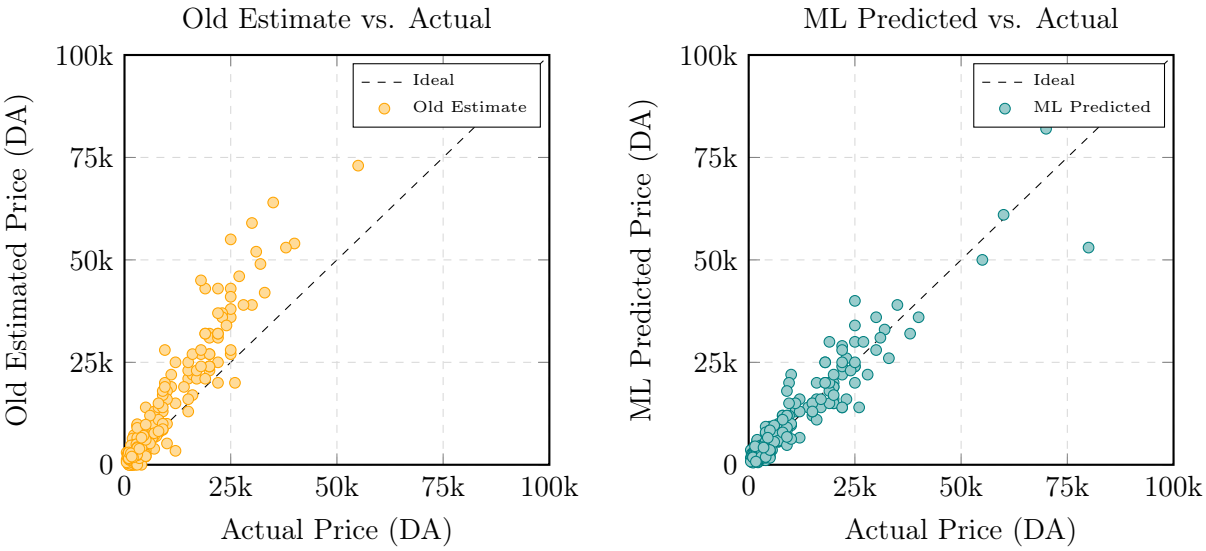


Figure 4.7: Actual vs. Estimated Prices using Old and ML Methods.

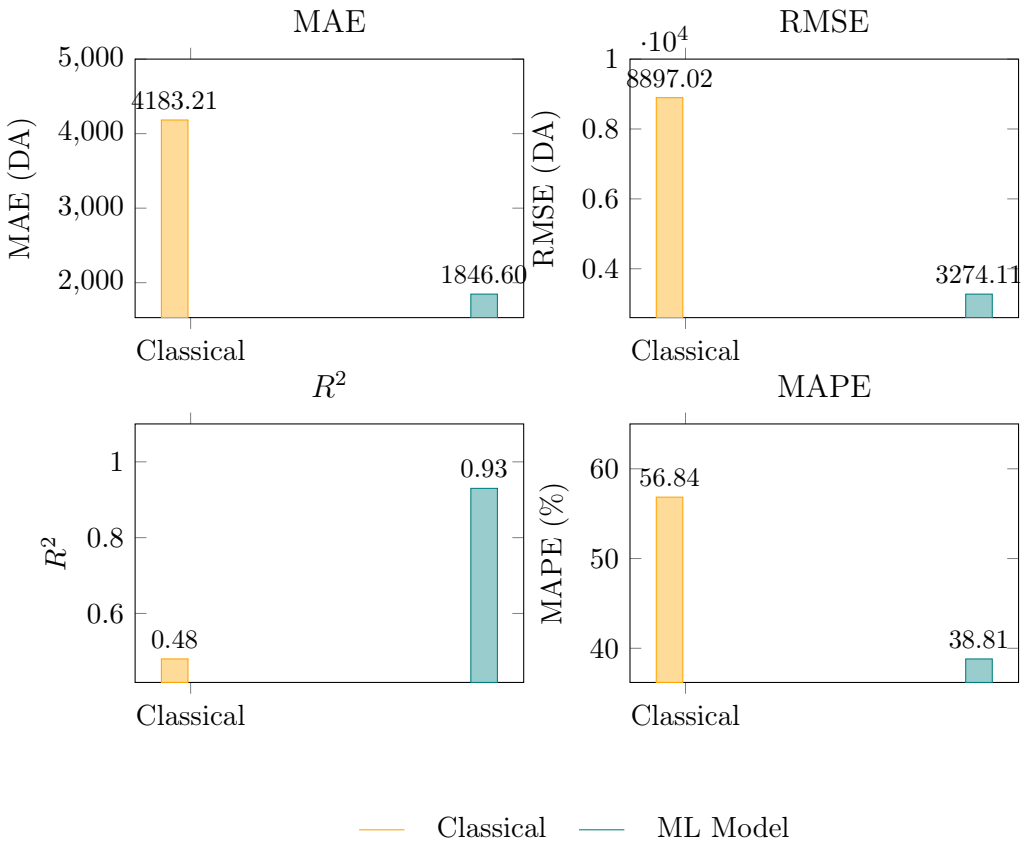


Figure 4.8: Comparison of Classical and ML models on key performance metrics: MAE, RMSE, R^2, and MAPE.

4.4.4 Practical Impact

By significantly reducing the prediction error, the machine learning model brings multiple benefits to CAMIO's operations:

- **Reduced underpricing:** Accurate estimates prevent financial loss on long or costly trips.
- **Minimized overpricing:** More competitive pricing increases order conversion rates.
- **Better planning:** Improved estimates help reduce empty returns by suggesting feasible return loads.
- **Scalability:** The new model can incorporate new factors (e.g., truck type, time of day, market trends) in future iterations.

In summary, the selected model outperforms the classical approach in both prediction quality and flexibility, positioning it as a more reliable solution for CAMIO's pricing engine.

4.5 Deploying and Model Integration

This section presents the production deployment process of the selected model, including the use of CI/CD pipelines for automation, endpoint configuration, traffic management between model versions, and integration into the CAMIO system.

4.5.1 Automated Deployment via CI/CD Pipeline

To ensure consistency and automation across environments, the deployment process was orchestrated using a CI/CD pipeline. The pipeline is triggered automatically after a successful model training and evaluation stage. It handles:

- **Model registration in Azure ML:** The trained model is saved and versioned in the Azure Model Registry to enable traceability and reproducibility.
- **Environment packaging and scoring script preparation:** The execution environment (e.g., Python version, libraries) and the inference logic are bundled to ensure compatibility during deployment.

- **Deployment to a predefined online endpoint:** The packaged model and environment are deployed to a real-time endpoint, making the model accessible via REST APIs.
- **Smoke testing to validate the new deployment:** A basic test request is sent to the endpoint to confirm that the model responds correctly and the deployment was successful.

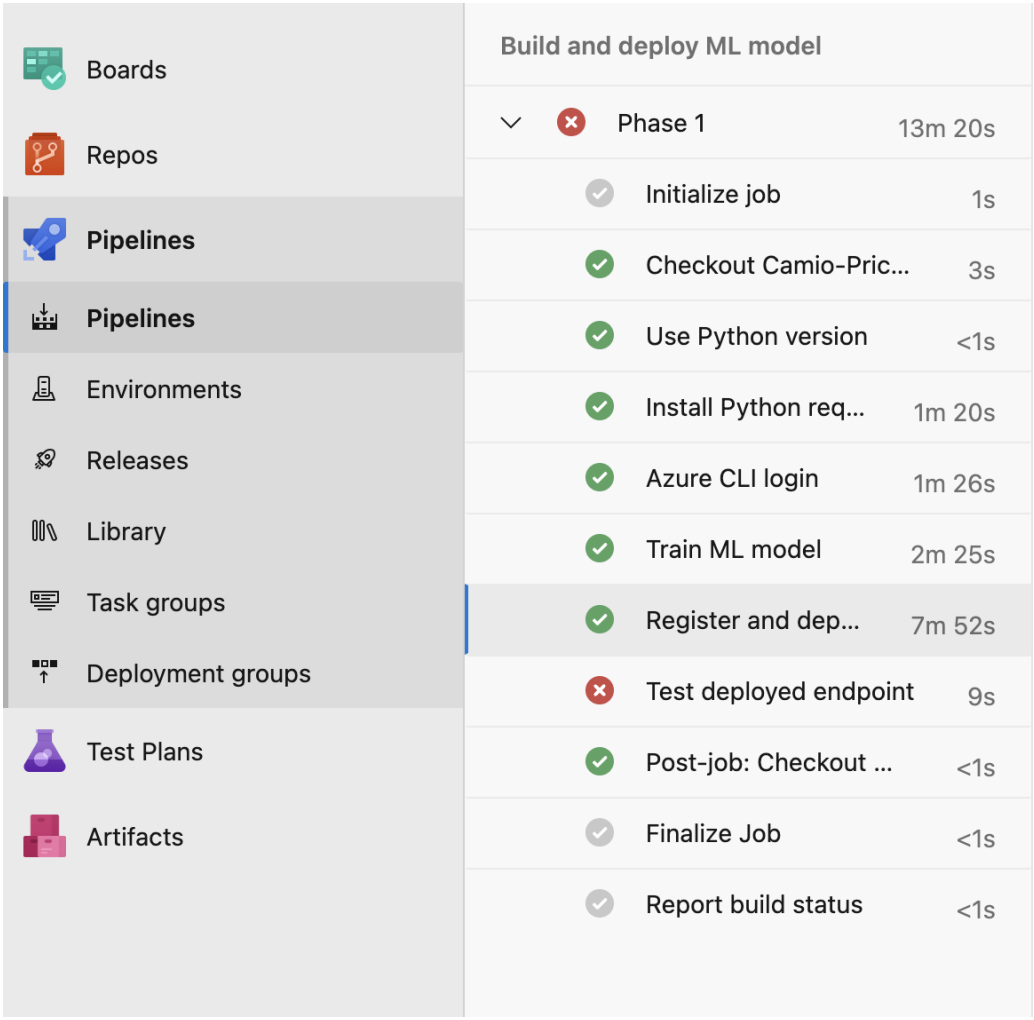


Figure 4.9: CI/CD pipeline overview for training and deployment

This automated approach ensures fast iterations and minimizes manual errors during deployment, supporting continuous model updates and experimentation.

4.5.2 Endpoint Overview in Azure ML

Azure Machine Learning provides a centralized interface to manage and inspect deployed endpoints. Each online endpoint serves as a container for one or more model versions and

exposes a REST API for inference.

The **Endpoints** dashboard in Azure ML Studio offers visibility into:

- Endpoint name, ID, and deployment status.
- The list of models registered under each endpoint.
- Resource configuration (CPU/Memory/Instance Count).
- Deployment history and health status.

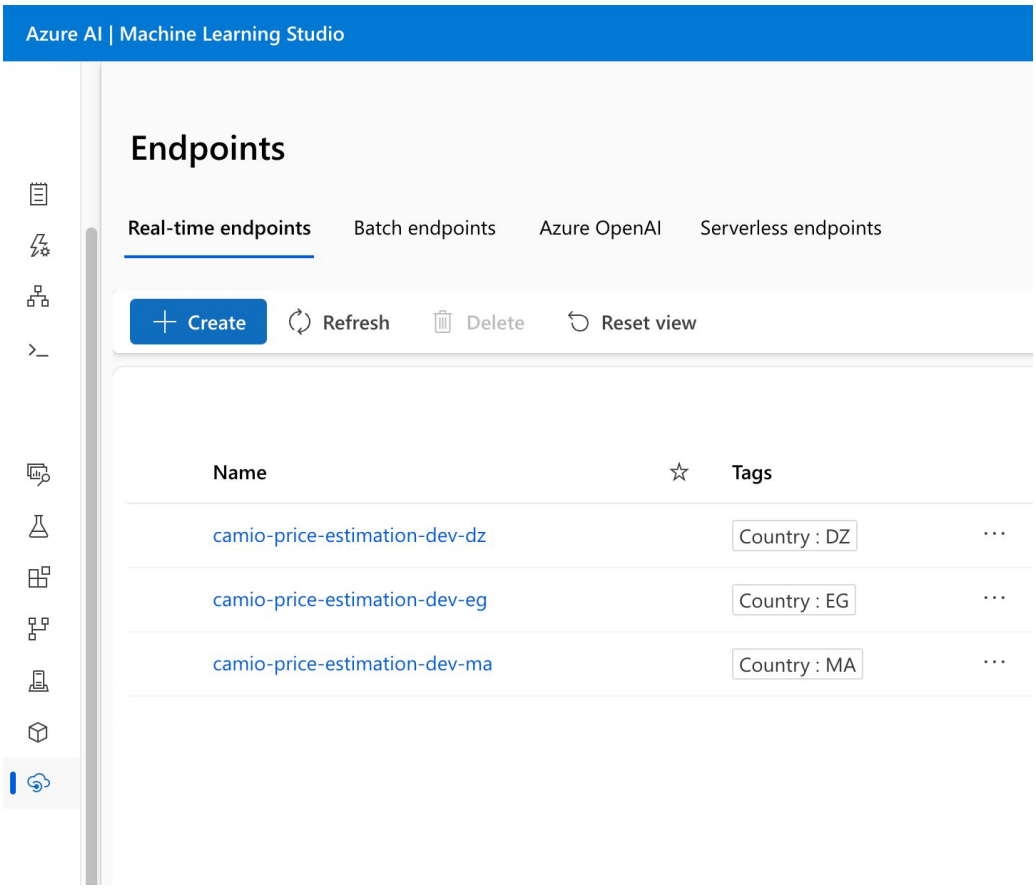


Figure 4.10: Azure ML Studio interface showing deployed endpoints and configuration

This visual interface helps data scientists and DevOps engineers monitor, manage, and troubleshoot live models with ease.

4.5.3 Model Versioning and Endpoint Traffic Management

Azure ML provides a robust **model registry** that tracks all registered models, their versions, and associated metadata. This centralized repository supports traceability and

reproducibility of experiments, enabling teams to compare models, manage lifecycle stages (e.g., staging, production), and automate promotion workflows.

Figure 4.11 shows a snapshot of the model registry within Azure ML Studio, where multiple versions of the freight price estimation model are registered with tags and descriptions to support streamlined deployment.

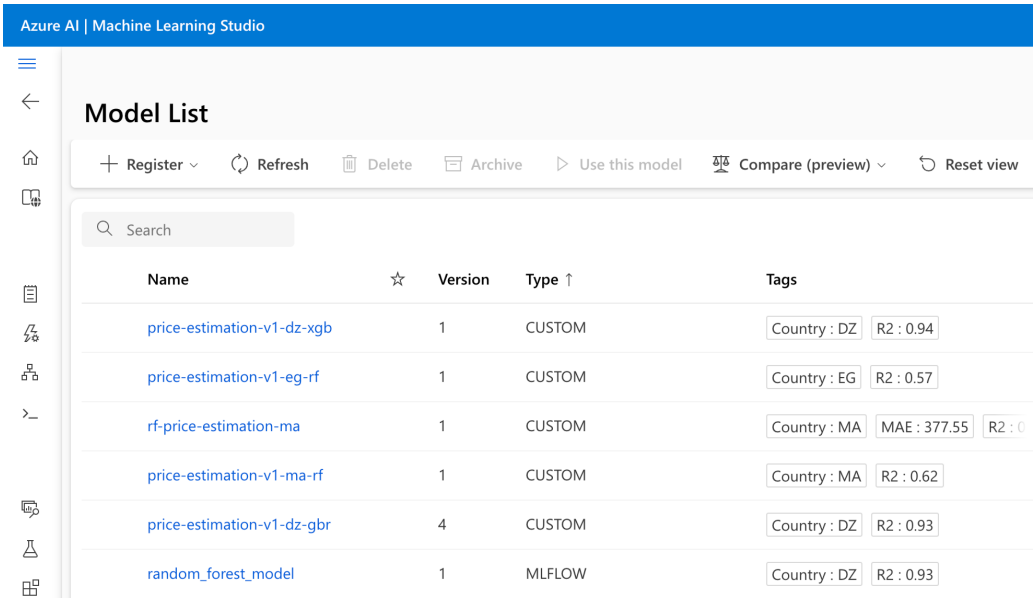


Figure 4.11: Model registry in Azure ML Studio listing multiple versions of the freight estimation models

Azure ML Online Endpoints support multiple model deployments under a single endpoint. This allows fine-grained traffic management between versions for strategies such as:

- **Canary deployments**: routing a small portion of traffic to the new model to observe performance.
- **Blue-green deployments**: switching traffic instantly between two models.
- **Rollback support**: reverting to a previous model version in case of degradation.

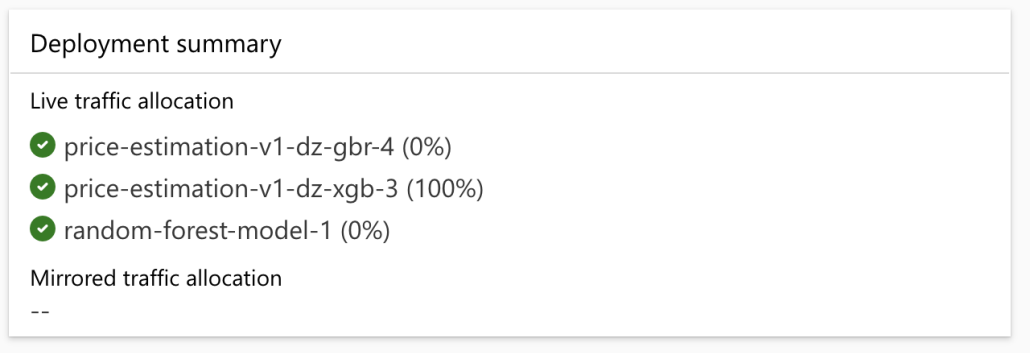


Figure 4.12: Example of traffic distribution between deployed models

In the CAMIO use case, this capability was used to shift 100% of the traffic to the newly deployed model once it passed the validation phase. In case of issues, traffic can be rerouted back to a previous model version without downtime.

4.5.4 Integration into CAMIO System

Once deployed, the endpoint exposes a secure REST API. CAMIO's backend collects input features, sends a POST request to the endpoint, and receives the predicted price in response.

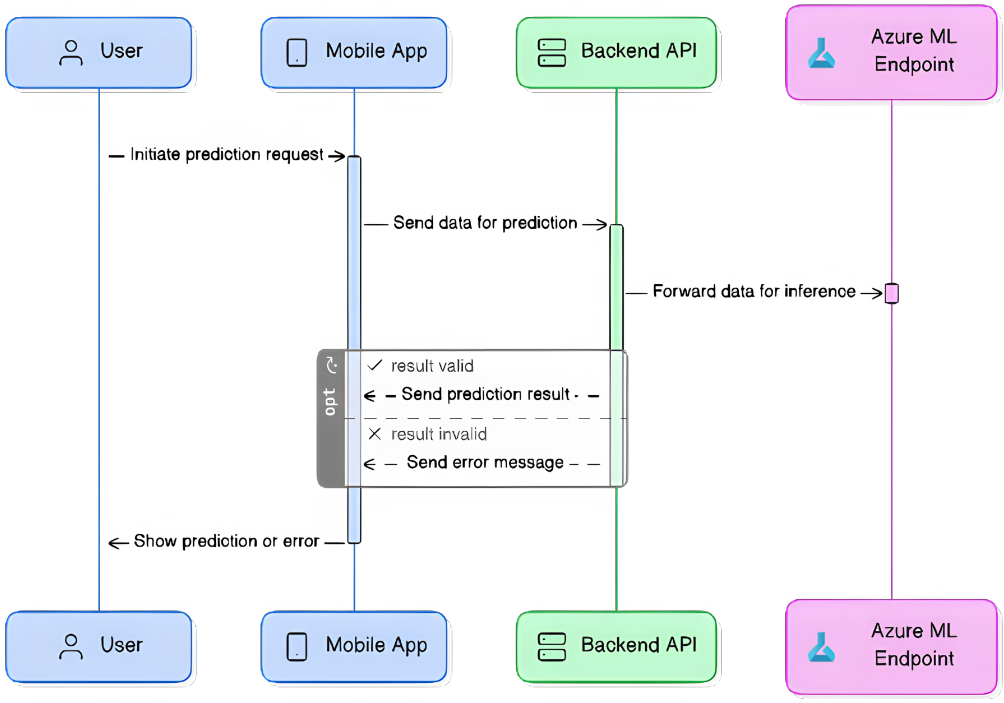


Figure 4.13: Integration flow of the deployed model with CAMIO's backend system

The endpoint uses an authentication token to ensure secure access. The prediction request includes all preprocessed features required by the model, and the response is parsed and displayed to the end-user via the interface.

4.5.5 Monitoring and Maintenance

Azure ML provides built-in tools for monitoring the health, latency, and traffic volume of each deployed model. Logs and metrics are continuously collected to identify performance degradation or failures.

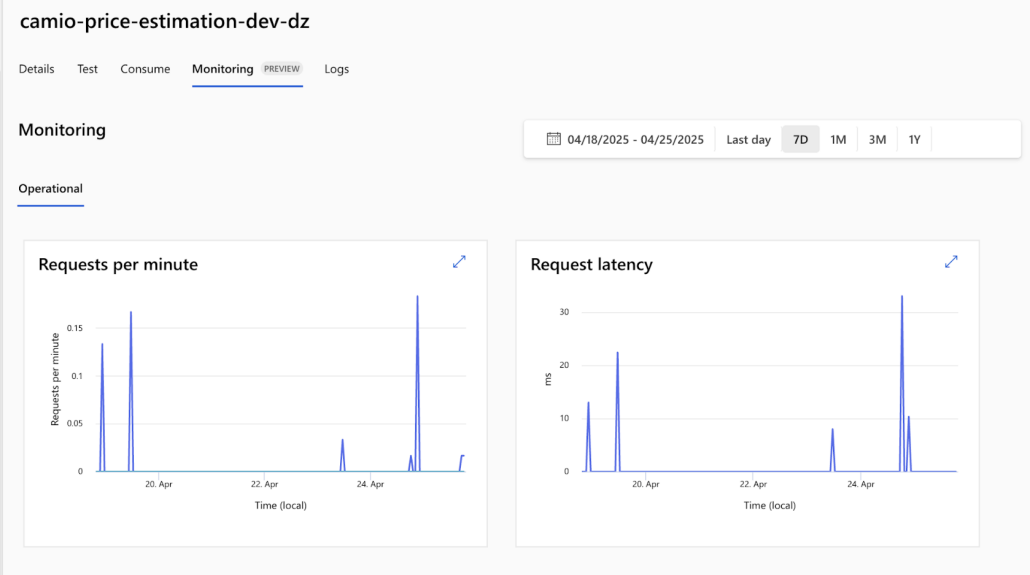


Figure 4.14: Monitoring interface for the online endpoint in Azure

This observability facilitates proactive maintenance and supports retraining workflows triggered by model drift or new incoming data.

4.6 Use-Case Simulation within the CAMIO Application

To demonstrate the practical integration of the developed price estimation model, a use-case simulation was conducted within the **CAMIO** mobile application. CAMIO serves as the core logistics platform where users post freight requests and truck owners bid on available trips.

This simulation replicates a real-world scenario using actual user data, starting from the trip creation process to receiving bids — powered by the machine learning-based price estimator deployed via Azure ML.

Step 1: Selecting Truck Type and Capacity

The user starts by selecting the appropriate truck category and capacity range that fits their cargo requirements.

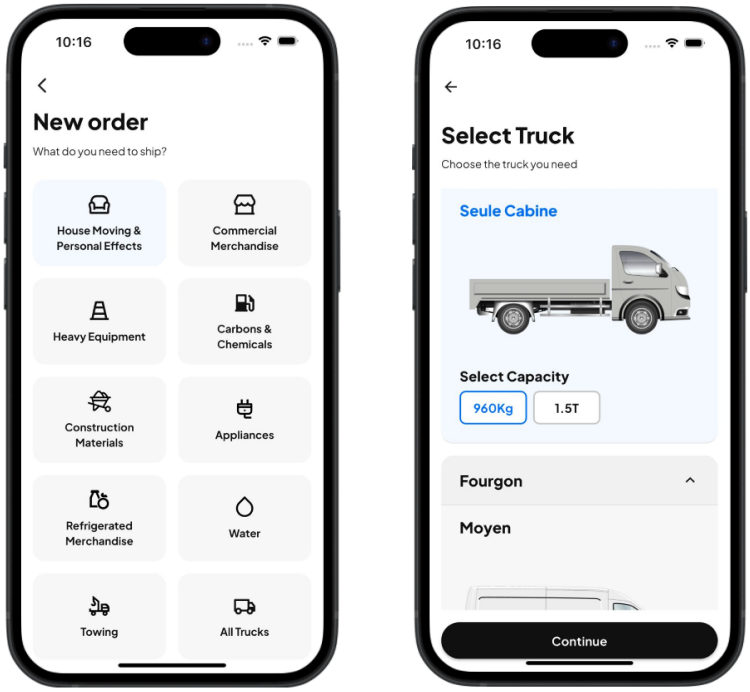


Figure 4.15: Selection of truck type and cargo capacity

Step 2: Choosing Pickup and Drop-off Locations

Next, the user specifies the route by selecting the pickup and delivery locations directly on the map interface.

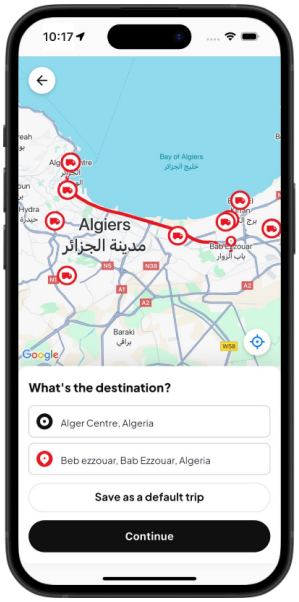


Figure 4.16: User sets pickup and drop-off points using the map

Step 3: Filling Trip Details and Previewing Order

The user completes the request form by entering the cargo description, planned pickup time, and other optional notes before previewing the full trip details.

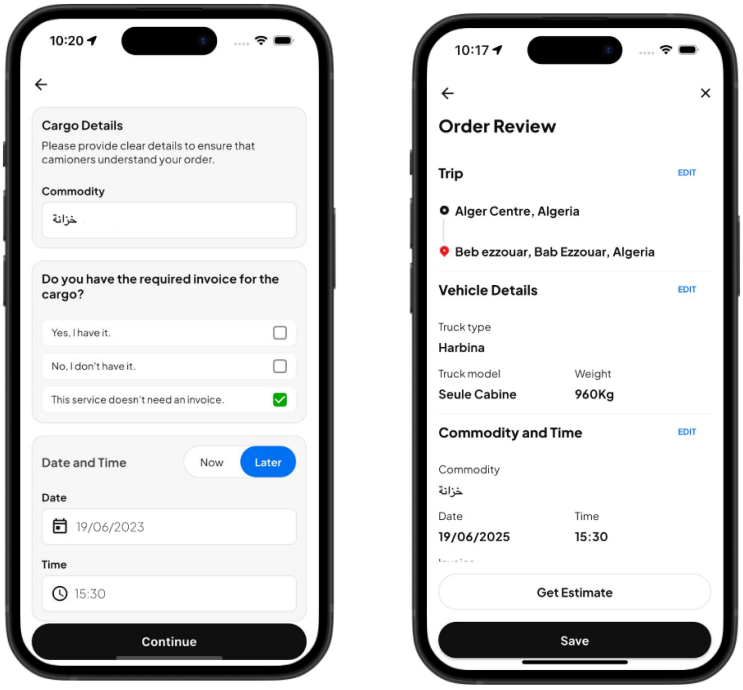


Figure 4.17: Trip information entry and order preview before confirmation

Step 4: Viewing the Machine Learning-Based Price Estimate

Once the request is confirmed, the app calls the deployed Azure ML endpoint. The model returns a predicted price based on parameters like route, cargo weight, and truck type.

Instead of a single fixed value, the predicted output is displayed as a price range. This range is generated by applying a percentage margin (e.g., ±10%) around the predicted mean, which reflects uncertainty and natural variability in market pricing. This approach helps users better understand potential fluctuations and improves the realism of the estimate.

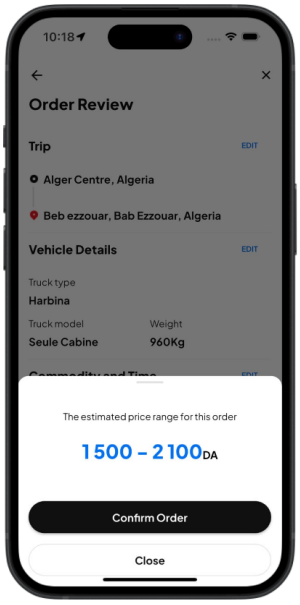


Figure 4.18: Estimated price returned from ML model and shown to user

Step 5: Receiving Real Bids from Truck Owners

After the price estimate, the trip is made publicly available for truck owners to place their bids. In this test simulation, real users submitted bids based on the posted request, demonstrating the practical impact of integrating machine learning into a live business process.

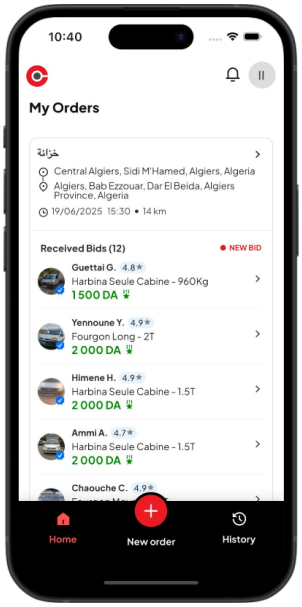


Figure 4.19: Truck owner bids submitted in response to the posted trip

This end-to-end use case showcases how predictive pricing, when integrated into operational workflows, can enhance pricing transparency, streamline user experience, and improve market efficiency in the freight logistics domain.

4.7 Challenges and Obstacles

One of the primary challenges encountered during the development of this solution was **the insufficiency of quality data** in certain regional markets. Although data collection pipelines were in place for Egypt and Morocco, the volume and consistency of trip records was not yet adequate to support reliable training and evaluation of machine learning models. As a result, the focus of this phase was on **the Algerian dataset**, which contained the most complete and structured data, making it suitable for experimentation, validation, and deployment.

Despite this limitation, the system architecture and the experiment tracking pipelines were designed with **scalability and extensibility** in mind. The implemented infrastructure, including data ingestion, preprocessing, model training workflows, and tracking of Azure ML experiments, has been structured to accommodate new data as they become available in other countries.

This foundational work ensures that once data in other regions (such as Egypt and Morocco) reaches a sufficient volume and quality, models tailored to those markets can be trained, evaluated, and deployed with minimal additional effort. This approach enables

future expansion without significant rework, allowing the models to evolve and mature as more operational data is collected from different markets.

4.8 Future Work

Although the proposed price prediction model has demonstrated promising performance, there are several directions in which this work can be extended to provide more comprehensive support for logistics platforms and to meet real-world business needs.

4.8.1 Expansion to Other Markets and Regions

Currently, the model is trained on data from a limited geographic scope. An important extension would be to adapt and retrain the model for additional countries or markets with different pricing dynamics and regulatory constraints. This would involve:

- Localized feature engineering to reflect country-specific cost factors.
- Collection of regional historical trip data.
- Evaluation of model generalizability and transfer learning strategies.

4.8.2 Bid Count Prediction

Another valuable addition would be a model that predicts the expected number of bids a cargo post will receive. This can help:

- Shippers assess interest before posting.
- Suggest optimal posting times and prices to increase competitiveness.
- Improve marketplace liquidity through better supply-demand awareness.

This could be implemented as a classification or regression task using features such as origin-destination pairs, cargo type, posting time, and past bidding activity.

4.8.3 Cargo Matching Using NLP

A key challenge in the platform is intelligent cargo-truck matching. Many shippers write free-form descriptions that may include non-standard terms. Leveraging Natural Language Processing (NLP) would allow:

- Extraction of structured attributes (e.g., cargo type, urgency, dimensions) from text.

- Semantic similarity matching between cargo and available trucks.
- Improved ranking and filtering of offers for both shippers and truck owners.

Advanced NLP models such as BERT or domain-tuned transformers can be explored for embedding cargo descriptions and enabling efficient matching.

4.8.4 Multi-Objective Optimization

Future iterations could focus on optimizing multiple KPIs, such as:

- Minimizing total trip costs while maximizing bid engagement.
- Balancing delivery times with truck utilization.
- Factoring in environmental impact (e.g., fuel usage or CO_2 emissions).

This would transform the system from a pure predictive model to a decision-support tool.

4.8.5 Operationalization with Azure ML Pipelines

To ensure reproducibility, scalability, and automation of experiments, the model and its extensions can be integrated into a full Azure ML pipeline, including:

- Experiment tracking and metric comparison dashboards.
- Model registry, deployment automation, and scheduled retraining.
- Integration with real-time APIs used by the logistics platform.

4.8.6 Real-Time Feedback Loops and Online Learning

As the platform grows, enabling the system to learn continuously from new user interactions and market changes is crucial. This could involve:

- Online learning pipelines that adjust to new patterns.
- Active learning techniques to prioritize the most valuable new data points.
- Real-time evaluation of model drift and performance alerts.

In conclusion, the future directions of this work involve both horizontal expansion into new business functionalities (e.g., bid prediction, matching) and vertical scaling for reliability and real-world usage (e.g., online learning, deployment). These enhancements will play a crucial role in transforming the platform into a smart, AI-driven logistics ecosystem.

4.9 Conclusion

This chapter has presented the implementation details and the results of the developed pricing prediction model for CAMIO. Through systematic experimentation and evaluation, the chosen model demonstrated a notable improvement in performance compared to baseline approaches, achieving higher accuracy and better generalization on unseen data. The model selection was guided by both quantitative metrics and practical relevance, ensuring that the final solution balances efficiency, scalability, and predictive reliability.

Beyond technical validation, the implementation offers significant practical value to CAMIO's pricing process. By automating and optimizing price estimation based on historical trip data and contextual features, the system enhances decision-making, reduces manual effort, and increases pricing consistency across the platform. This contribution not only supports current operations but also lays the foundation for future enhancements such as real-time pricing, demand forecasting, and dynamic bidding strategies.

In summary, the results confirm the effectiveness of the implemented solution and underscore its potential to bring about significant improvements in CAMIO's business operations.

General Conclusion

In this thesis, we developed and deployed a full end-to-end machine learning solution for price prediction on the CAMIO logistics platform. Starting from extensive data preprocessing and feature engineering—including distance normalization, one-hot encoding of truck types, and geospatial clustering—we constructed a scalable MLOps pipeline on Microsoft Azure that seamlessly integrates model training, validation, and monitoring.

Through rigorous experimentation, we evaluated several candidate regressors (linear models, Random Forest, and XGBoost) against both a rule-based baseline and a custom distance-based algorithm tailored to CAMIO's business logic. The XGBoost model consistently outperformed all others, achieving an RMSE of 2.995.02, an MAE of 1.776.69, and an R^2 of 0.94 on a held-out test set—improving estimation accuracy by over 30

We then integrated the chosen model into CAMIO's production environment and validated its effectiveness through simulated use cases such as return-leg optimization and competitive bid prediction. These deployments demonstrated not only the model's robustness to outliers and shifting market conditions, but also tangible gains in operational efficiency and customer satisfaction.

Finally, we reflected on the challenges encountered—data sparsity in low-frequency corridors, concept drift during seasonal peaks, and orchestration of real-time inference—and proposed clear next steps, including the incorporation of online learning loops, NLP-powered cargo-bid matching, and multi-objective optimization to balance price, time, and load factors.

Overall, this work delivers a practical, high-precision pricing engine that bridges advanced data science techniques with real-world logistics needs, setting a solid foundation for CAMIO's future growth and for research in intelligent transportation systems.

Bibliography

[1] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305.

[2] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[3] Jason Brownlee. *Deep Learning for Time Series Forecasting*. Machine Learning Mastery. 2018. URL: https://machinelearningmastery.com/deep-learning-for-time-series-forecasting/.

[4] Lei Chen and Xinyu Wang. "Freight Cost Estimation via XGBoost and Support Vector Regression". In: *Journal of Logistics Analytics* (2023), pp. 102–117.

[5] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), pp. 785–794. DOI: 10.1145/2939672.2939785.

[6] S. Feng, X. Kong, B. Ai, Y. Kong, L. Su, and Y. Ning. "Artificial intelligence: a key to relieve China's insufficient and unequally-distributed medical resources". In: *American Journal of Translational Research* 11.5 (2019), pp. 2632–2637. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC6556644/.

[7] J. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29.5 (2001), pp. 1189–1232. DOI: 10.1214/aos/1013203451.

[8] S. Garg, A. Sinha, and S. Roy. "MLOps: Continuous delivery and automation pipelines in machine learning". In: *IEEE Software* (2022).

[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, and Y. Bengio. "Generative adversarial nets". In: *Advances in Neural Information Processing Systems*. Vol. 27. 2014.

[10] Google Cloud. *Practicing MLOps: ML as a software engineering discipline*. https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning. 2021.

[11] T. Gracias. "Cloud-native ML Deployment with Azure Kubernetes Service". In: *Journal of Cloud Systems* (2025).

[12] A. Gujarati, Z. Shen, and R. Lee. "Swayam: Scaling machine learning serving with dynamic auto-scaling". In: *ACM Symposium on Cloud Computing*. 2017.

[13] Md Hasebul Hasan, Md Abid Jahan, Mohammed Eunus Ali, Yuan-Fang Li, and Timos Sellis. "A Multi-Modal Deep Learning Based Approach for House Price Prediction". In: *arXiv preprint arXiv:2409.05335* (2024). URL: https://arxiv.org/abs/2409.05335.

[14] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd. New York, NY, USA: Springer, 2009. ISBN: 978-0-387-84858-7.

[15] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[16] K. Hornik, M. Stinchcombe, and H. White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366.

[17] R. J. Hyndman and G. Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2021.

[18] S. Ioffe and C. Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *ICML* (2015). URL: https://arxiv.org/abs/1502.03167.

[19] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning (2nd ed.)* Springer, 2021.

[20] Jaro Education. *Over Fitting & Under Fitting in Machine Learning*. Accessed: 2025-06-04. 2025. URL: https://www.jaroeducation.com/blog/over-fitting-under-fitting-in-machine-learning/.

[21] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems* 30 (2017), pp. 3146–3154. URL: https://proceedings.neurips.cc/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[22] S. Kotsiantis, D. Kanellopoulos, and P. Pintelas. "Handling imbalanced datasets: A review". In: *GESTS International Transactions on Computer Science and Engineering* (2006).

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[24] A. Liaw and M. Wiener. "Classification and Regression by randomForest". In: *R News* 2.3 (2002), pp. 18–22.

[25] Xiaoming Liu and et al. "Field Experiment of Dynamic Pricing in E-commerce Using Deep Reinforcement Learning". In: *ACM Transactions on Internet Technology* (2019), pp. 1–25.

[26] Microsoft Azure. *MLOps v2 - Azure reference architecture for machine learning operations*. https://learn.microsoft.com/en-us/azure/architecture/example-scenario/mlops/mlops-v2-overview. 2022.

[27] Microsoft Azure. *Monitor Azure Machine Learning online endpoints*. https://learn.microsoft.com/en-us/azure/machine-learning/how-to-endpoints-online-monitor. 2022.

[28] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997. URL: https://www.cs.cmu.edu/~tom/mlbook.html.

[29] S. J. Pan and Q. Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2009), pp. 1345–1359.

[30] Rohan Patil and Meena Shah. "Used-Car Price Prediction Using Regression Trees". In: *Proceedings of the International Conference on Automotive Data Science*. 2021, pp. 88–97.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[33] A. L. Samuel. "Some studies in machine learning using the game of checkers". In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.

[34] D. Sculley et al. "Hidden technical debt in machine learning systems". In: *Advances in Neural Information Processing Systems*. 2015.

[35] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, ..., and D. Dennison. "Hidden technical debt in machine learning systems". In: *Advances in Neural Information Processing Systems*. Vol. 28. 2015.

[36] Zakaria Abdellah Sellam, Cosimo Distante, Abdelmalik Taleb-Ahmed, and Pier Luigi Mazzeo. "Boosting House Price Estimations with Multi-Head Gated Attention". In: *arXiv preprint arXiv:2405.07456* (2024). URL: https://arxiv.org/abs/2405.07456.

[37] Anil Sharma and Priya Gupta. "Rice Price Prediction Using Support Vector Regression and Random Forest". In: *International Journal of Agricultural Forecasting* (2022), pp. 45–58.

[38] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[39] B. Stone, W. Chen, and K. Malik. "Sustaining UX in Deployed ML Applications: A Lifecycle Approach". In: *HCI Journal* (2025).

[40] M. Studer and R. Winter. "CRISP-ML(Q): A data science process model with quality assurance methodology". In: *Journal of Decision Systems* (2021).

[41] M. Zaharia et al. "Accelerating the Machine Learning Lifecycle with MLflow". In: *Data+AI Summit*. 2019.

[42] H. Zhang, S. Li, and W. Wang. "A Machine Learning Approach to Dynamic Pricing in Transportation". In: *IEEE Transactions on Intelligent Transportation Systems* 18.10 (2017), pp. 2772–2784.

[43] Y. Zheng, L. Capra, O. Wolfson, and H. Yang. "Urban Computing: Concepts, Methodologies, and Applications". In: *ACM Transactions on Intelligent Systems and Technology (TIST)* 5.3 (2016), pp. 1–55.