National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

# Continuous Machine Learning for Cooperative, Connected and Automated Mobility applications

DIPLOMA THESIS

of

**Georgios A. Kyriakopoulos**

**Supervisor:** Panayiotis Tsanakas
Professor NTUA

Athens, October 2025

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Computer Science

# Continuous Machine Learning for Cooperative, Connected and Automated Mobility applications

DIPLOMA THESIS

of

**Georgios A. Kyriakopoulos**

**Supervisor:** Panayiotis Tsanakas
Professor NTUA

Approved by the three-member scientific committee on the 30th of October 2025

| | | |
|---|---|---|
| ..... | ..... | ..... |
| Panayiotis Tsanakas | Andreas-Georgios Stafylopatis | Georgios Alexandridis |
| Professor NTUA | Professor Emeritus NTUA | Assistant Professor NKUA |

Athens, October 2025

.....

**Georgios A. Kyriakopoulos**
Graduate of School of Electrical and Computer Engineering,
National Technical University of Athens

# Περίληψη

Η αστική κινητικότητα είναι εκ φύσεως μη σταθερή: μεταβολές στον καιρό, τη ζήτηση και τις συνθήκες του δικτύου αλλοιώνουν τις κατανομές των δεδομένων και υποβαθμίζουν την ακρίβεια των μοντέλων με την πάροδο του χρόνου. Η παρούσα διπλωματική εργασία σχεδιάζει, υλοποιεί και αξιολογεί μια πλατφόρμα συνεχούς μηχανικής μάθησης με επίγνωση εννοιολογικής μετατόπισης (concept drift) για εφαρμογές Συνεργατικής, Συνδεδεμένης και Αυτοματοποιημένης Κινητικότητας (CCAM). Το σύστημα συνδυάζει μια αρθρωτή (modular) αρχιτεκτονική μικροϋπηρεσιών για εξυπηρέτηση μοντέλων, παρακολούθηση σε πραγματικό χρόνο, συναίνεση πολλαπλών ανιχνευτών concept drift και αυτόματη επαναεκπαίδευση με αντικατάσταση εν λειτουργία (hot-swap). Κατασκευάζεται ένα αναπαράξιμο σύνολο δεδομένων για το κέντρο της Αθήνας με βάση το SUMO, υπό κανονικές και δυσμενείς καιρικές συνθήκες. Το concept drift επάγεται μέσω μείωσης της τριβής ώστε να προσομοιωθεί βροχή, διατηρώντας την τοπολογία του δικτύου και παράγοντας μετρήσιμες μεταβολές (μέση ταχύτητα $-13.68\%$, μέση διάρκεια διαδρομής $+16.74\%$).

Η αξιολόγηση επικεντρώνεται στην πρόβλεψη Χρόνου Άφιξης (ETA) με ένα μοντέλο LightGBM εμπλουτισμένο με ειδικά χαρακτηριστικά για το πεδίο, σε ένα πείραμα χρονικής συμπίεσης (time-lapse) με απότομο (abrupt) concept drift στη μέση, λόγω βροχής. Υπό συνθήκες drift, τα σφάλματα του βασικού μοντέλου αυξάνονται αισθητά (MAE $30.36\,s \rightarrow 56.93\,\mathrm{s}$ και MAPE $13.20\% \rightarrow 19.02\%$). Μετά την ανίχνευση του drift, το μοντέλο επαναεκπαιδεύεται με μία ώρα δεδομένων μετά το drift και αντικαθίσταται εν λειτουργία (hot-swap), ανακτώντας την απόδοση, σε σύγκριση με το βασικό μοντέλο (MAE $-11.00\,\mathrm{s}$, $-25.42\%$ και MAPE $-2.34\,\mathrm{pp.}$, $-13.85\%$). Η ευέλικτη αρχιτεκτονική της πλατφόρμας τεκμηριώνεται περαιτέρω με αξιολόγηση μοντέλων Κατανάλωσης Καυσίμου και Αριθμού Στάσεων, ενώ ο πίνακας ελέγχου και η διεπαφή χρήστη παρέχουν ερμηνευσιμότητα σε πραγματικό χρόνο. Συνολικά, τα αποτελέσματα αναδεικνύουν μια πρακτική προσέγγιση κλειστού βρόχου για ανίχνευση, απόκριση και αντιμετώπιση του concept drift σε ρεαλιστικά αστικά κυκλοφοριακά περιβάλλοντα.

***Λέξεις Κλειδιά:*** Συνεχής Μηχανική Μάθηση, Εννοιολογική Μετατόπιση, Πρόβλεψη Χρόνου Άφιξης, SUMO, MLOps, Δέντρα Ενίσχυσης Κλίσης, Συνεργατική Συνδεδεμένη και Αυτοματοποιημένη Κινητικότητα, Αστική Κινητικότητα

# Abstract

Urban mobility is inherently non-stationary: shifts in weather, demand, and network conditions alter data distributions and degrade model accuracy over time. This diploma thesis designs, implements, and evaluates a drift-aware platform for continuous machine learning in Cooperative, Connected and Automated Mobility (CCAM). The system combines a modular microservice architecture for model serving, online monitoring, multi-detector drift consensus, and automated retraining with hot-swap deployment. A reproducible SUMO-based dataset for central Athens is constructed under normal and adverse-weather scenarios. Concept drift is induced via friction reduction to emulate rain, preserving network topology while yielding measurable shifts (average speed $-13.68\%$, average trip duration $+16.74\%$).

Evaluation centers on Estimated Time of Arrival (ETA) prediction with a LightGBM model enriched by domain-specific features, exercised in a time-lapse experiment with an abrupt drift at the midpoint, due to rain conditions. Under drift, baseline errors increase markedly (MAE $30.36\,s \rightarrow 56.93\,\mathrm{s}$ and MAPE $13.20\% \rightarrow 19.02\%$). After detecting drift, the model is retrained using one hour of post-drift data and hot-swapped, recovering performance, when compared to the baseline model (MAE $-11.00\,\mathrm{s}$, $-25.42\%$ and MAPE $-2.34\,\mathrm{pp}$, $-13.85\%$). The platform's flexible architecture is further evidenced by the evaluation of Fuel Consumption and Number of Stops models, while its dashboard and user interface provide real-time interpretability. Overall, the results demonstrate a practical, closed-loop approach to detecting, responding to, and mitigating concept drift in realistic urban traffic settings.

***Keywords:*** Continuous Machine Learning, Concept Drift, ETA Prediction, SUMO, MLOps, Gradient Boosted Trees, Cooperative Connected and Automated Mobility, Urban Mobility

# Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor, **Professor Panayiotis Tsanakas**, for giving me the opportunity to conduct my thesis in a field that has been of great personal interest to me.

I would also like to thank **Dr. Georgios Drainakis** and **Dr. Panagiotis Pantazopoulos** of the **I-SENSE Research Group of ICCS/NTUA** for their valuable feedback and guidance towards the completion of this work.

I am grateful to my family and friends for their support and encouragement throughout this important chapter of my life, especially **Valentina** and **Artemis**, as well as **Nick**, **Apostolis**, **Alex**, **Christini**, and **Elina**.

Finally, and most importantly, I would like to thank **Serafeim** and **George**, with whom I shared the vision, the challenges, and the excitement of this work. Their collaboration and commitment were essential in shaping this thesis, and I truly appreciate sharing this journey with them.

# Contents

# List of Figures

# List of Tables

# Extended Abstract in Greek

## Εισαγωγή

### Κίνητρο

Τα συστήματα Συνεργατικής, Συνδεδεμένης και Αυτοματοποιημένης Κινητικότητας (CCAM) βασίζονται, πλέον, όλο και περισσότερο στη Μηχανική Μάθηση για κρίσιμες λειτουργίες, όπως η πρόβλεψη κυκλοφορίας και η λήψη αποφάσεων, όπου η ακρίβεια επηρεάζει άμεσα την ασφάλεια και την αποδοτικότητα [1]. Οι πρόσφατες τάσεις στο CCAM τονίζουν την ανάγκη για προσαρμογή σε πραγματικό χρόνο καθώς τα αστικά περιβάλλοντα γίνονται πιο δυναμικά [2]. Ωστόσο, πολλά σημερινά συστήματα δεν επανεκπαιδεύουν συνεχώς τα μοντέλα καθώς φτάνουν νέα δεδομένα [3]. Μία βασική περιοχή μελέτης είναι η πρόβλεψη *Χρόνου Άφιξης (ETA)*.

### Ορισμός Προβλήματος

Τα αστικά συστήματα κινητικότητας είναι εγγενώς μη σταθερά: οι στατιστικές ιδιότητες των δεδομένων αλλάζουν με τον χρόνο, άρα μοντέλα Μηχανικής Μάθησης που εκπαιδεύτηκαν σε στατικά ιστορικά σύνολα δεδομένων υποβαθμίζονται επιχειρησιακά λόγω εννοιολογικής μετατόπισης (concept drift) [4, 5]. Το concept drift προκαλείται από ποικίλους παράγοντες, όπως δομικές αλλαγές στο οδικό δίκτυο, καιρικά φαινόμενα, εποχική ζήτηση, και οδηγεί σε απότομες ή σταδιακές μεταβολές ταχυτήτων ή συμφόρησης. Έτσι, ένα μοντέλο που αποδίδει καλά σε «στεγνές» συνθήκες, υστερεί υπό συνθήκες έντονης βροχής ή σε αιχμές κυκλοφορίας, εφόσον αυτές οι συνθήκες δεν περιλαμβάνονταν στην εκπαίδευση του [6].

Παρά την πλούσια βιβλιογραφία σε ανίχνευση και προσαρμογή drift, οι υπάρχουσες λύσεις είναι συχνά αποσπασματικές: σπάνια ενοποιούν προσομοίωση, συνεχή παρακολούθηση απόδοσης και αυτόματη επαναεκπαίδευση σε έναν «κλειστό βρόχο» [5, 7]. Αυτό που παραμένει ανεπίλυτο είναι η δημιουργία μιας *ολοκληρωμένης πλατφόρμας* ειδικά προσαρμοσμένης για περιβάλλοντα CCAM, που να διαχειρίζεται συστηματικά το concept drift σε κατανεμημένα σενάρια κινητικότητας [8, 9]. Η παρούσα διπλωματική εργασία στοχεύει να καλύψει αυτό το κενό με μια πλατφόρμα μικροϋπηρεσιών που ενσωματώνει προσομοίωση, πίνακες ελέγχου απόδοσης, αλγορίθμους ανίχνευσης και αυτόματη επανεκπαίδευση, διατηρώντας την ακρίβεια των μοντέλων στο δυναμικό αστικό περιβάλλον.

### Στόχοι και Συνεισφορές

Ως κύρια περιοχή μελέτης επιλέγεται η πρόβλεψη Χρόνου Άφιξης (ETA) σε ένα σενάριο βασισμένο στο κέντρο της Αθήνας [2, 3]. Κατασκευάζεται ένα αναπαράξιμο σύνολο δεδομένων μέσω SUMO, όπου το drift επάγεται ελεγχόμενα με μείωση της τριβής στο οδικό δίκτυο (βροχή), διατηρώντας την τοπολογία αλλά μεταβάλλοντας ρεαλιστικά τις ταχύτητες και τους χρόνους άφιξης [10, 11]. Η

πλατφόρμα είναι ανεξάρτητη από το μοντέλο και επεκτάσιμη. Αν και το επίκεντρο είναι το ETA με χρήση μοντέλων ενίσχυσης κλίσης (gradient boosting), υποστηρίζει παράλληλα μοντέλα (π.χ. κατανάλωση καυσίμου, αριθμός στάσεων) [12, 13]. Συνολικά, αποδεικνύεται ότι μια πρακτική, με επίγνωση concept drift, αρχιτεκτονική μπορεί να διατηρεί την ακρίβεια προβλέψεων σε δυναμικά, ρεαλιστικά περιβάλλοντα κινητικότητας.

# Υπόβαθρο

## Μηχανική Μάθηση

Η Μηχανική Μάθηση (Machine Learning) μελετά αλγορίθμους που μαθαίνουν πρότυπα από δεδομένα για βελτίωση της πρόβλεψης/λήψης αποφάσεων με περιορισμένη ανθρώπινη παρέμβαση. Οι βασικές κατηγορίες περιλαμβάνουν επιβλεπόμενη (supervised), μη επιβλεπόμενη (unsupervised), ημι-επιβλεπόμενη (semi-supervised) και ενισχυτική μάθηση (reinforcement learning) [14]. Ενδεικτικά μοντέλα είναι η Γραμμική Παλινδρόμηση (Linear Regression), ως απλό και ερμηνεύσιμο σημείο αναφοράς, τα Δέντρα Αποφάσεων (Decision Trees) και τα Τυχαία Δάση (Random Forests) για ερμηνευσιμότητα και ανθεκτικότητα, καθώς και σύγχρονα μοντέλα ενίσχυσης κλίσης (Gradient Boosting) όπως τα XGBoost, LightGBM και CatBoost, που πετυχαίνουν υψηλή απόδοση σε δεδομένα σε πίνακες [15, 16, 17, 18].

## Βαθιά Μάθηση

Η Βαθιά Μάθηση (Deep Learning) αποτελεί υποπεδίο της ML που αξιοποιεί τεχνητά νευρωνικά δίκτυα με πολλαπλά επίπεδα για εξαγωγή σύνθετων χαρακτηριστικών [19]. Κύριες αρχιτεκτονικές: Πολυεπίπεδοι Αντιληπτές (Multilayer Perceptron) για γενική παλινδρόμηση/ταξινόμηση, Συνελικτικά Δίκτυα (Convolutional Neural Networks) για χωρικά μοτίβα, Επαναληπτικά Δίκτυα (Recurrent Neural Networks) για ακολουθίες και οι Μετασχηματιστές (Transformers) που κυριαρχούν σε γλώσσα και όλο και περισσότερους τομείς, όπως όραση. Τα δίκτυα αυτά είναι ευέλικτα, αλλά απαιτούν περισσότερα δεδομένα και προσεκτική τακτικοποίηση/ρύθμιση.

## Μετρικές Αξιολόγησης Μοντέλων

Η αξιολόγηση μοντέλων παλινδρόμησης στηρίζεται σε μετρικές όπως το μέσο απόλυτο σφάλμα (MAE), το μέσο απόλυτο ποσοστιαίο σφάλμα (MAPE), που είναι χρήσιμο αλλά ασταθές όταν ο στόχος πλησιάζει το μηδέν, η ρίζα μέσου τετραγωνικού σφάλματος (RMSE), η οποία τονίζει τα μεγάλα σφάλματα και το συντελεστή προσδιορισμού $R^2$ [20].

## Κανονικοποίηση, Υπεροπροσαρμογή, Υποπροσαρμογή

Προβλήματα υπερπροσαρμογής/υποπροσαρμογής αντιμετωπίζονται με κανονικοποίηση (L1/Lasso για αραιότητα, L2/Ridge για σταθερότητα), ενώ στα νευρωνικά χρησιμοποιούνται επίσης dropout, early stopping και weight decay.

## Διασταυρωμένη Επικύρωση και Ρύθμιση Υπερπαραμέτρων

Η διασταυρωμένη επικύρωση (k-fold, stratified, leave-one-out, repeated) αποτελεί χρυσό κανόνα για αξιόπιστη εκτίμηση απόδοσης και επιλογή υπερπαραμέτρων. Το μοντέλο και οι ρυθμίσεις του επιλέγονται συγκρίνοντας υποψηφίους μέσω έγκυρης διαδικασίας επικύρωσης. Για τη ρύθμιση υπερ-

παραμέτρων χρησιμοποιούνται grid/random search και πιο προχωρημένες μέθοδοι όπως η Μπεϋζιανή Βελτιστοποίηση.

### Συνεχής Μηχανική Μάθηση

Η συνεχής μηχανική μάθηση εστιάζει στη σταδιακή προσαρμογή μοντέλων καθώς ρέουν νέα δεδομένα, αντί για «εκπαίδευση μία κι έξω». Κεντρική πρόκληση είναι το «καταστροφικό ξεχάσμα». Στρατηγικές όπως incremental/online learning, μεταφορά μάθησης και εμπειρική επανάληψη (experience replay) ισορροπούν σταθερότητα και πλαστικότητα [21].

### MLOps

Το MLOps συνδέει το Machine Learning και τα DevOps για να τυποποιεί/αυτοματοποιεί τον κύκλο ζωής μοντέλων: έκδοση κώδικα/δεδομένων/μοντέλων, CI/CD για εκπαίδευση, αξιολόγηση, διάθεση, και rollback, παρακολούθηση απόδοσης/συμμόρφωσης και συνεργασία ρόλων [22]. Στόχος είναι ταχύτερη και αξιόπιστη μετάβαση από πειραματισμό σε παραγωγή, με ισχυρή ιχνηλασιμότητα.

### Μετατόπιση Έννοιας

Το concept drift, μεταβολή στις υποκείμενες στατιστικές ιδιότητες του στόχου (δεδομένων), μειώνει την ακρίβεια με τον χρόνο [5]. Μορφές: απότομη, βαθμιαία, επαναλαμβανόμενη και εποχική, αυξητική. Οι ανιχνευτές είτε παρακολουθούν σφάλματα είτε κατανομές εισόδων (π.χ. DDM/EDDM, Page-Hinkley, ADWIN, έλεγχοι κατανομών, SPC) [23, 24, 25, 26]. Κρίσιμα διλήμματα: ευαισθησία και ψευδώς θετικά αποτελέσματα, ανάγκη ετικετών, υστέρηση/latency και υπολογιστικό κόστος. Η προσαρμογή γίνεται με συχνό retraining, online ενημέρωση, πρόσφατα παράθυρα δεδομένων ή ensembles που ευνοούν τα πιο πρόσφατα μοντέλα.

### Προσομοιωτής SUMO

Για προσομοίωση κινητικότητας, το SUMO (microscopic, multi-modal) εισάγει δίκτυα από OSM ή εμπορικές πηγές, παράγει ζήτηση (συνθετική με ζευγάρια προέλευσης και προορισμού), υποστηρίζει TraCI για ζωντανό έλεγχο και εξάγει λεπτομερή μεγέθη (τροχιές, στατιστικά διαδρομών, καταστάσεις κυκλοφορίας) για αναλυτικά πειράματα μεγάλης κλίμακας.

### Στοίβα Υλοποίησης

Η στοίβα υλοποίησης αξιοποιεί κοντέινερ (Docker) και ορχήστρωση υπηρεσιών (Docker Compose) για αναπαραξιμότητα/φορητότητα. Οι διεπαφές εκτίθενται ως REST/HTTP, με FastAPI (ισχυρή επικύρωση τύπων, OpenAPI), ενώ οπτικοποιήσεις/πίνακες ελέγχου υλοποιούνται με Dash/Plotly. Η βιβλιοθήκη Pydantic επιβάλλει σχήματα σε δεδομένα εισόδου/εξόδου και το Apache Parquet προσφέρει αποδοτική αποθήκευση/συμπίεση δεδομένων και ταχύτερα analytics.

## Σχετικό Έργο

### Τοπίο των εργαλείων για ανίχνευση/παρακολούθηση drift

Το River είναι ένα ενιαίο πακέτο online μάθησης σε ροές (incremental, δείγμα προς δείγμα), με αλγορίθμους (γραμμικά, Hoeffding trees, σύνολα) και έτοιμους ανιχνευτές drift (ADWIN, DDM, EDDM) [27]. Πλεονεκτεί σε αποδοτικότητα, χαμηλό overhead και ταχεία προσαρμογή, με μειονέκτημα τη μικρότερη αξιοποίηση batch/GPU σε σχέση με αμιγώς batch βιβλιοθήκες.

Το scikit-multiflow είναι πρόγονος του River στο Python οικοσύστημα [28]. Προσφέρει κλασικούς stream αλγορίθμους, generators με ελεγχόμενο drift και ανιχνευτές (ADWIN, DDM, EDDM, Page-Hinkley). Πλέον δεν συντηρείται, πρακτικά έχει «απορροφηθεί» από το River, παραμένει όμως χρήσιμο για αναφορές/benchmarking.

Το Alibi & Alibi Detect είναι δύο εργαλεία, όπου το πρώτο προορίζεται για ερμηνευσιμότητα (counterfactuals, influence), και το δεύτερο για outliers/προσδιορισμό drift σε πίνακα, κείμενο, εικόνα, χρονοσειρές [29]. Υποστηρίζουν στατιστικούς ελέγχους και ανιχνευτές με TensorFlow/PyTorch (π.χ. embedding-based). Έχουν ισχυρή κάλυψη, αλλά απαιτούν ενσωμάτωση σε pipeline (όχι πλήρες σύστημα παρακολούθησης).

Το Evidently είναι μία βιβλιοθήκη ή ένα εργαλείο για αναφορές με πίνακες ελέγχου για data ή target drift, ποιότητα δεδομένων και απόδοση μοντέλων με σύγκριση αναφορών εναντίον τωρινών συνόλων [30]. Έχει εξαιρετική ευχρηστία και οπτικοποίηση για batch ροές. Για λειτουργία σε πραγματικό χρόνο χρειάζεται επιπλέον ενσωμάτωση (κυλιόμενα παράθυρα, προγραμματισμένες εργασίες).

Το NannyML επικεντρώνεται στην εκτίμηση απόδοσης χωρίς άμεσες επισημάνσεις (CBPE), συνδέοντας το drift με πιθανή πτώση ακρίβειας [31]. Είναι πολύ χρήσιμο όταν οι επισημάνσεις καθυστερούν. Στους περιορισμούς, χρησιμοποιείται κυρίως για ταξινομήσεις, και απαιτεί καλή βαθμονόμηση πιθανοτήτων και σωστή ένταξη σε MLOps.

### Βιβλιογραφία για την πρόβλεψη Χρόνου Άφιξης (ETA)

Οι κλασικές μέθοδοι χρονοσειρών (ARIMA) λειτουργούν σε περιορισμένα σενάρια, αλλά υστερούν στη μη γραμμική φύση της κυκλοφορίας. Η Βαθιά Μάθηση (RNN, LSTM, Transformers) αξιοποιεί χρονικές εξαρτήσεις και καθολικές (global) αλληλεπιδράσεις [32, 33]. Τα GNNs ενσωματώνουν ρητά τη δομή του οδικού γράφου και βελτιώνουν την πρόβλεψη υπό δυναμικές συνθήκες. Παράλληλα, στα δεδομένα σε πίνακες (tabular data) με ισχυρά εμπλουτισμένα χαρακτηριστικά, τα gradient boosting δέντρα (XGBoost, LightGBM, CatBoost) παραμένουν ιδιαίτερα ανταγωνιστικά: γρήγορη εκπαίδευση/εξυπηρέτηση, ερμηνευσιμότητα (feature importances) και εύκολη περιοδική επαναπροσαρμογή [34, 35]. Συχνά η πρακτική κατεύθυνση είναι υβριδική: εμπλουτισμένα χαρακτηριστικά με boosting ως ισχυρή βάση αναφοράς, και εξειδικευμένα χωροχρονικά Deep Learning ή GNN μοντέλα όπου τα δεδομένα και η υποδομή το δικαιολογούν.

### Καινοτομία της προτεινόμενης πλατφόρμας

(1) Ελεγχόμενο concept drift με SUMO: κατασκευή σεναρίου time-lapse όπου το drift (π.χ. βροχή μέσω μείωσης τριβής) επάγεται ρεαλιστικά, διατηρώντας την τοπολογία αλλά αλλάζοντας τις ταχύτητες/χρόνους [11]. Αυτό γεφυρώνει το χάσμα μεταξύ συνθετικών γεννητριών και απρόβλεπτων πραγματικών γεγονότων, επιτρέποντας αξιολόγηση σε «γνωστό» συμβάν drift. (2) Ενσωμάτωση και παρακολούθηση πολλαπλών μοντέλων: ταυτόχρονη πρόβλεψη και εποπτεία για ETA, κατανάλωση καυσίμου και αριθμό στάσεων. Σύγκριση συμπεριφορών ανά task και συναίνεση (consensus) ανιχνευτών ανά μοντέλο για μείωση ψευδώς θετικών αποτελεσμάτων. Αντανακλά ρεαλιστική επιχειρησιακή πολυπλοκότητα. (3) Αυτόματη επαναεκπαίδευση & αντικατάσταση εν λειτουργία: κλείσιμο του βρόχου, όταν ανιχνευτεί drift, ενεργοποιείται retraining με πρόσφατο παράθυρο δεδομένων και γίνεται άμεση αντικατάσταση μοντέλου χωρίς downtime. Πέρα από monitoring, δείχνει ένα αυτοδιορθούμενο σύστημα με σαφείς αποφάσεις για το πότε αξίζει το retraining, με σκοπό την αποφυγή περιττών ή πρόορων μέτρων αντιμετώπισης. (4) Πίνακας ελέγχου σε πραγματικό χρόνο: οπτικοποιεί σφάλματα, μετρικές, και συμβάντα (detections, retrains, swaps) και επιτρέπει human-in-the-loop διερεύνηση (π.χ. σταθερή διαδρομή ως δείγμα πριν, κατά, και μετά το drift). Σε αντίθεση

με απλές στατικές αναφορές, προσφέρει διαρκή παρακολούθηση.

# Παραγωγή Δεδομένων και Έρευνα Μηχανικής Μάθησης

## Παραγωγή Δεδομένων

Το κεφάλαιο περιγράφει την κατασκευή ενός συνθετικού, αλλά ρεαλιστικού, συνόλου δεδομένων αστικής κυκλοφορίας για το κέντρο της Αθήνας και τη μεθοδολογία αξιολόγησης ενός μοντέλου ETA, υπό ελεγχόμενο concept drift. Η προσομοίωση υλοποιείται στο SUMO και παράγει τρία σενάρια 10 ωρών με τηλεμετρία FCD ανά δευτερόλεπτο: train, test (βάση) και rain (drift). Το rain εισάγει φυσικά ερμηνεύσιμο concept drift μειώνοντας ομοιόμορφα τον συντελεστή τριβής οδοστρώματος, προκαλώντας χαμηλότερες ταχύτητες και μεγαλύτερες διάρκειες διαδρομής. Το pipeline είναι πλήρως παραμετροποιήσιμο (σταθερά seeds, YAML configs), αρθρωτό (modular) και φορητό σε άλλες περιοχές ή εντάσεις drift.

### Περιοχή Μελέτης

Ορθογώνιο, πυκνό δίκτυο με φωτεινή σηματοδότηση στο κέντρο της Αθήνας, ώστε να αναδύονται ουρές και ρεαλιστική συμφόρηση για ETA. Το τελικό δίκτυο: 1184 ακμές, 689 κόμβοι, 106 σηματοδότες, 68.17 km συνολικού μήκους.

### Κατασκευή Δικτύου

Η δημιουργία γίνεται προγραμματιστικά (scripts με osmGet.py και osmBuild.py που δίνουν πρόσβαση σε netconvert και polyconvert) σε γραμμή εντολών, χωρίς διεπαφή χρήστη. Το βασικό δίκτυο (τριβή=1.0) εξάγεται από δεδομένα OpenStreetMap (OSM), ενώ για το δίκτυο βροχής, κλωνοποιείται και ορίζεται τριβή=0.4 σε όλες τις λωρίδες. Έτσι διατηρείται η τοπολογία, απομονώνοντας το φυσικό αποτέλεσμα της μειωμένης τριβής από τεχνητές αλλαγές δομής.

### Κλάσεις Οχημάτων

Χρησιμοποιούνται μόνο IX, ώστε να περιοριστεί η ετερογένεια (χωρίς δίκυκλα και βαρέα οχήματα) και να απλοποιηθεί η ερμηνεία του drift διατηρώντας τη βασική δυναμική του ETA.

### Μοντελοποίηση Ζήτησης

Ωριαίο μοτίβο 08:00–18:00 με αιχμές πρωί και απόγευμα και κάμψη το μεσημέρι [36]. Οι περίοδοι γένεσης (δευτερόλεπτα/όχημα) δέχονται μικρό Γκαουσιανό θόρυβο (μ=1.0, σ=0.01) ανά ώρα ώστε να διαφοροποιούνται οι ροές χωρίς να αλλοιώνεται το σχήμα. Κάθε σενάριο έχει δικό του seed που ελέγχει θόρυβο περιόδων, τυχαίες διαδρομές, θέσεις εκκίνησης/άφιξης στις λωρίδες και στοχαστικότητα συμπεριφοράς. Τα trips παράγονται με randomTrips.py και ελέγχονται για εφικτότητα διαδρομής.

### Επιλογή Μηχανισμού Drift

Δοκιμάστηκαν εναλλακτικές, όπως κλεισίματα λωρίδων ή ολόκληρων δρόμων και μεταβολές σε συμπεριφορές οδηγών, αλλά απορρίφθηκαν λόγω προσομοιωτικών τεχνικών επιπλοκών, αστάθειας ή δύσκολης ερμηνείας. Επιλέχθηκε βροχή μέσω τριβής για ρεαλιστικό και μετρήσιμο αντίκτυπο,

διατήρηση τοπολογίας και εγγενή υποστήριξη στο SUMO. Η επιλογή τριβή=0.4 είναι σκόπιμα έντονη για καθαρό, ανιχνεύσιμο drift χωρίς κατάρρευση του συστήματος. Παρατηρούνται χαμηλότερες επιταχύνσεις, νωρίτερα ή ηπιότερα φρεναρίσματα και αυξημένοι χρόνοι.

**Pipeline**

(1) Εξαγωγή OSM → (2) Κατασκευή δικτύου → (3) Παραγωγή δικτύου βροχής → (4) Ρυθμίσεις γραφικής διεπάφης χρήστη → (5) Παραμετροποίηση ανά σενάριο → (6) Γένεση διαδρομών → (7) Εκτέλεση SUMO → (8) Μετατροπή από CSV σε Parquet → (9) Διερευνητική ανάλυση δεδομένων. Τα βήματα 1 έως 4 εκτελούνται μία φορά, ενώ τα βήματα 5 έως 9 μία φορά για κάθε σενάριο.

**Αναπαραξιμότητα/Επεκτασιμότητα**

Όλες οι παράμετροι είναι σε ένα αρχείο YAML, κάθε βήμα είναι καθαρή συνάρτηση με inputs και outputs, και υπάρχει εκτεταμένο logging και έλεγχοι επιτυχίας. Εύκολη προσαρμογή περιοχής (bounding box), ζήτησης και θορύβου, seeds, διάρκειας, παραμέτρων drift (τριβή), επιλογών netconvert, πολιτικών rerouting, κλάσεων οχημάτων.

**Μορφή Δεδομένων Εξόδου**

FCD ανά δευτερόλεπτο: timestep, id, x, y, speed, lane, odometer, fuel, waiting. Τα CSV μετατρέπονται σε Parquet (στηλοθετημένο και συμπιεσμένο) για οικονομία χώρου και ταχύτερες συσσωρεύσεις και εξαγωγή χαρακτηριστικών.

**Χαρακτηριστικά Συνόλου**

Και τα τρία σενάρια έχουν διάρκεια 36000 s (10 ώρες). Ενδεικτικά: test μέση ταχύτητα 29.8 km/h και διάρκεια 211.6 s, rain 25.7 km/h και 247.0 s. Οι μεταβολές επιβεβαιώνουν καθαρό drift (μειωμένη ταχύτητα, αυξημένος χρόνος). Οι όγκοι FCD είναι της τάξης δεκάδων εκατομμυρίων εγγραφών με σημαντικό όφελος αποθήκευσης σε Parquet.

**Διαθεσιμότητα Δεδομένων**

Το σύνολο δεδομένων (έκδοση 4) είναι δημόσια διαθέσιμο σε CSV και Parquet για train, test και rain (Zenodo, DOI 10.5281/zenodo.16950674) [37], εξασφαλίζοντας πλήρη αναπαραξιμότητα και επαναχρησιμοποίηση σε μελλοντικά πειράματα.

## Έρευνα Μηχανικής Μάθησης

Το παρόν τμήμα περιγράφει τη διαδικασία ανάπτυξης μοντέλου πρόβλεψης Χρόνου Άφιξης (ETA) στο σύνολο δεδομένων του κέντρου της Αθήνας. Στόχοι: (i) ισχυρές βάσεις αναφοράς ανά οικογένεια μοντέλων, (ii) αξιολόγηση στρατηγικών εξαγωγής χαρακτηριστικών, (iii) επιλογή τελικού μοντέλου με συστηματικό συντονισμό και διασταυρωμένη επικύρωση, (iv) αναφορά απόδοσης και αποδοτικότητας εκπαίδευσης. Κύριο αποτέλεσμα: LightGBM με MAE 26.51,s και MAPE 12.77%, χρόνο εκπαίδευσης 2.95,s. Τα XGBoost και CatBoost πέτυχαν παρόμοια ακρίβεια, αλλά ήταν πιο αργά.

**Από FCD σε Διαδρομές**

Το FCD του SUMO παρέχει ανά-δευτερόλεπτο τηλεμετρία: timestep, id, x, y, speed, lane, odometer, fuel, waiting. Στο επίπεδο διαδρομής, ομαδοποιήθηκαν εγγραφές ανά όχημα και εξήχθησαν:

time_start (αρχή διαδρομής), duration (διάρκεια διαδρομής), source_x, source_y, destination_x, destination_y (συντεταγμένες προέλευσης και προορισμού), distance (απόσταση που διανύθηκε). Εφαρμόστηκε φιλτράρισμα ποιότητας, όπου διαδρομές με διάρκεια μικρότερη από 30 δευτερόλεπτα ή απόσταση μικρότερη από 200 μέτρα απορρίφθηκαν. Το τελικό σύνολο έχει 53229 trips με όλα τα προηγούμενα χαρακτηριστικά, εκτός από τον στόχο (τη διάρκεια).

### Πειραματική Μεθοδολογία

**Ιχνηλάτηση πειραμάτων.** Ενοποιημένη βιβλιοθήκη συναρτήσεων, σταθερά seeds, κοινό αρχείο YAML για παραμετροποίηση, αποθήκευση τεχνουργημάτων (μοντέλα, μετρικές, logs) και αυτόματη συλλογή αποτελεσμάτων.

**Διασταυρωμένη επικύρωση.** 5-fold με χώρισμα που διατηρεί την κατανομή της διάρκειας για σταθεροποίηση εκτιμήσεων. Τα test και rain κρατήθηκαν αποκλειστικά για τελική αξιολόγηση (χωρίς διαρροή δεδομένων).

**Μετρικές.** MAE (s), MAPE (%), και χρόνος εκπαίδευσης (s) ως χρήσιμος δείκτης για συχνές επανεκπαιδεύσεις.

### Βάσεις Αναφοράς

Συγκρίθηκαν η Γραμμική Παλινδρόμηση και τρία boosting μοντέλα (LightGBM, XGBoost, CatBoost) πάνω στο αρχικό σετ χαρακτηριστικών (συντεταγμένες προέλευσης και προορισμού, ώρα εκκίνησης, απόσταση). Τα boosting μοντέλα υπερείχαν καθαρά της Γραμμικής Παλινδρόμησης, επιβεβαιώνοντας τη σημασία των μη γραμμικών σχέσεων για το ETA. Έτσι, τα επόμενα πειράματα επικεντρώθηκαν στα τρία boosting μοντέλα.

### Πειράματα Μετασχηματισμών

Εξετάστηκαν λογαριθμικοί μετασχηματισμοί, κανονικοποιήσεις χαρακτηριστικών, και μετασχηματισμοί στόχου (λογαριθμικός, Box-Cox, ποσοστιαίος). Κανένας δεν έδωσε ουσιαστικό κέρδος, αναμενόμενο για δέντρα boosting που είναι ανθεκτικά σε μονοτονικούς ή κλιμακωτούς μετασχηματισμούς, και απορρίφθηκαν για απλοποίηση.

### Χαρακτηριστικά Πεδίου

Προστέθηκαν ομάδες χαρακτηριστικών για χρονικές και χωρικές σχέσεις (Temporal, Spatial, Fourier, Cell, Cluster, PCA). Κάθε ομάδα βελτίωσε ελαφρά το MAE έναντι της βάσης αναφοράς, ο συνδυασμός όλων των ομάδων όμως είχε υψηλότερο MAE από το άθροισμα των επιμέρους ομαδών, υποδεικνύοντας μερική επικάλυψη πληροφορίας. Για την τελική επιλογή χαρακτηριστικών, συνδυάστηκαν μέθοδοι σπουδαιότητας χαρακτηριστικών, όπως gain, permutation, τιμές SHAP [38] και ανάλυση συσχετίσεων. Διατηρήθηκαν 22/52 χαρακτηριστικά (όλα τα 6 αρχικά, 2 Cluster, 4 PCA και 10 Spatial), με 40% μικρότερο χρόνο εκπαίδευσης και πρακτικά αμελητέα απώλεια MAE/MAPE.

### Ρύθμιση Υπερπαραμέτρων και Τελικό Μοντέλο

Χρησιμοποιήθηκε η βιβλιοθήκη Optuna [39] σε δύο φάσεις: ευρεία (100 πειράματα/μοντέλο) και εστιασμένη (50 πειράματα/μοντέλο). Κριτήριο ήταν η ελαχιστοποίηση MAE με 5-fold. Συνολικά έγιναν 450 πειράματα.

Το LightGBM μοντέλο πέτυχε το καλύτερο MAE και MAPE (26.51 s και 12.77%) με 2.95 s εκπαίδευση, ταχύτερο από XGBoost (περίπου 3 φορές) και CatBoost (περίπου 5 φορές), και επιλέχθηκε ως τελικό μοντέλο.

# Αρχιτεκτονική και Υλοποίηση Πλατφόρμας

## Επισκόπηση

Η πλατφόρμα υλοποιεί μία αρχιτεκτονική μικροϋπηρεσιών που ενορχηστρώνει μία time-lapse προσομοίωση κυκλοφορίας, κάνει πολλαπλές προβλέψεις (ETA/Καύσιμο/Στάσεις), παρακολουθεί ροές σφαλμάτων για concept drift με drift detector consensus και προσαρμόζεται μέσω retraining και hot-swap, εκθέτοντας ένα πίνακα ελέγχου για διαχείριση και μία διεπαφή χρήστη για προβλέψεις.

## Αρχιτεκτονική και Ρόλοι Υπηρεσιών

Αποτελείται από έξι ανεξάρτητες υπηρεσίες σε ένα ιδιωτικό Docker δίκτυο, με HTTP/REST διεπαφές, χρησιμοποιώντας τα FastAPI και Pydantic [40, 41].

Το διάγραμμα Figure 1 δείχνει τη γενική αρχιτεκτονική του συστήματος.

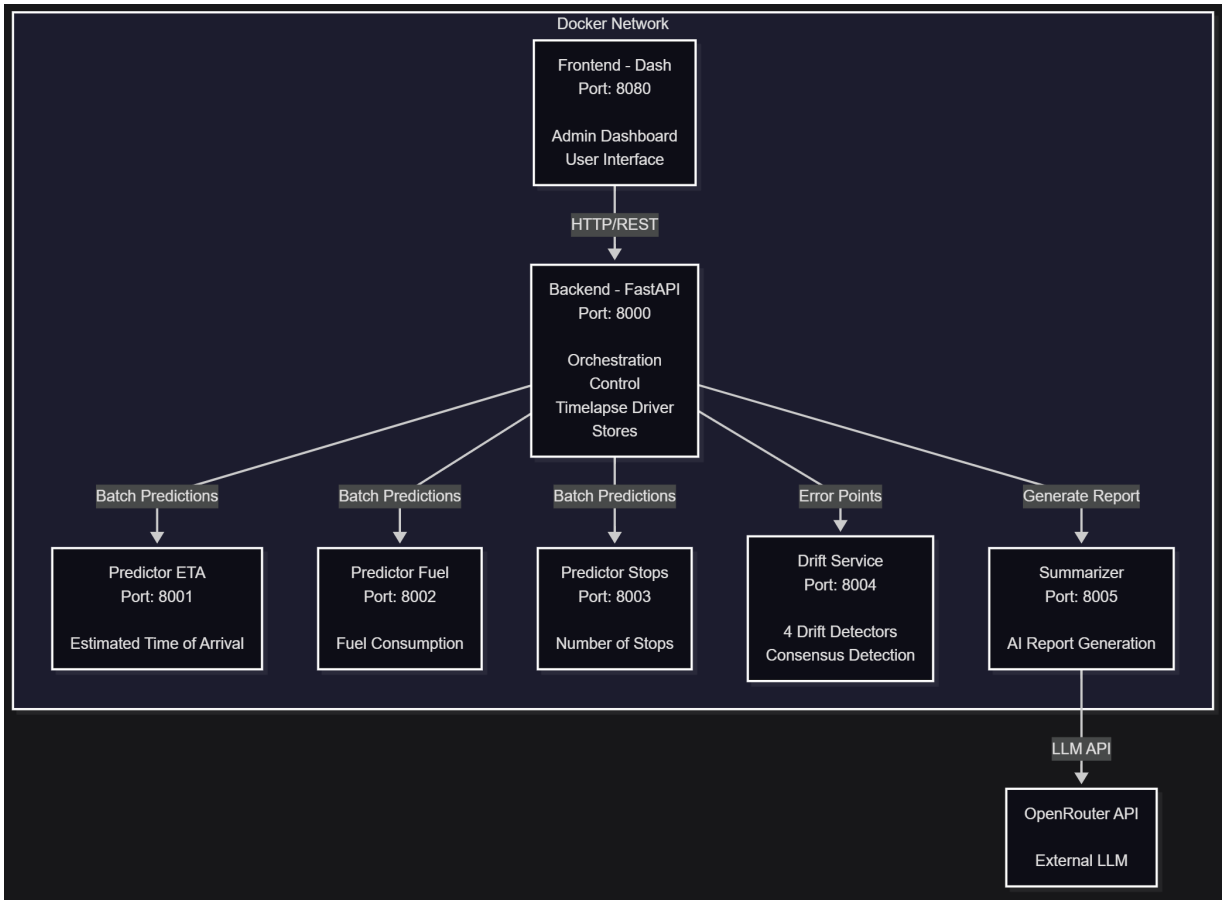


Figure 1: Αρχιτεκτονική της Πλατφόρμας.

### Backend

Ορχηστρώνει την προσομοίωση 20 ωρών σε περίπου 4 έως 5 λεπτά (300 φορές γρηγορότερα). Έχει τον TimelapseDriver που προχωράει το ρολόι 5 λεπτά ανά tick, ζητάει παράλληλα πολλαπλές προβλέψεις από τους predictors, στέλνει ελέγχους για drift detection με τα σφάλματα που επιστρέφουν και ζητάει επανεκπαιδεύσεις, ενώ στο τέλος πυροδοτεί τον Summarizer. Κρατά Metrics/Notification/Report stores σε μνήμη και ανιχνεύει διαθέσιμες υπηρεσίες Predictor.

### Predictors

Διαχειρίζονται πολλαπλές μοντέλα προβλέψεων (ETA/Καύσιμο/Στάσεις) και την επαναεκπαίδευση τους. Κοινός κώδικας ανά task: ModelManager (εκδόσεις μοντέλων), DataLoader (διάβασμα Parquet ανά timestamp), Predictor (πολλαπλές ή μοναδική πρόβλεψη), FeatureCalibrator (online εξαγωγή χαρακτηριστικών), RetrainService (ξεχωριστή διεργασία για επανεκπαιδεύσεις, δυναμικό μέγεθος μοντέλου), SumoService για προεπισκόπηση διαδρομής.

### Drift

Παρακολουθεί ροές σφαλμάτων για concept drift με consensus. Μία εργασία ανά μοντέλο, με ουρές αιτημάτων/απαντήσεων. ADWIN, Page-Hinkley, KSWIN, SPC σε εξομαλυμένες ροές σφαλμάτων, περίοδος ανοχής, consensus 3/4, διαδικασία βαθμονόμησης και επαναβαθμονόμηση μετά το hot-swap.

### Summarizer

Παράγει αναφορές μετά το τέλος της προσομοίωσης με χρήση Μεγάλων Γλωσσικών Μοντέλων. Λαμβάνει μετρικές και ειδοποιήσεις και παράγει Markdown/PDF. Καλεί το OpenAI API και συνδέεται με εξωτερικό Μεγάλο Γλωσσικό Μοντέλο (OpenRouter), με επανάληψη αιτήματος και χρονικά όρια.

### Frontend

Ανακτά δεδομένα από το Backend με περιοδικά αιτήματα. Dash, Plotly, Bootstrap, Leaflet. Πίνακας ελέγχου διαχειριστή με μετρικές, ειδοποιήσεις, κατάσταση του concept drift, προβολή αναφοράς. Διεπαφή χρήστη με επιλογή διαδρομής στον χάρτη και σύγχρονες προβλέψεις.

## Στοίβα Τεχνολογιών

### Υποδομή

Docker, Docker Compose, uv, Python 3.12

### API

FastAPI, Uvicorn, uvloop, httptools, ORJSON, httpx async

### Frontend

Dash/Plotly, Bootstrap, Leaflet, xhtml2pdf

### Μηχανική Μάθηση

LightGBM, XGBoost, NumPy, pandas, River για ανιχνευτές

#### Δεδομένα

pandas, NumPy, PyArrow, Parquet

#### Επικύρωση

Pydantic

### Ροή Προσομοίωσης

Κάθε δευτερόλεπτο το ρολόι προχωράει 5 λεπτά προσομοίωσης. (1) Backend στέλνει παράλληλα batch αιτήματα. (2) Predictors φορτώνουν παράθυρο/κάνουν inference/επιστρέφουν MAE. (3) Backend → Drift για ενημέρωση ανιχνευτών και κατάστασης. (4) Ενημέρωση καταστημάτων και μεταβάσεις εφόσον υπάρχει ανίχνευση concept drift. (5) Συλλογή post-drift παραθύρου, επαναεκπαίδευση στο παρασκήνιο και αντικατάσταση εν λειτουργία. (6) Frontend στέλνει περιοδικά αιτήματα για ζωντανή ενημέρωση. (7) Summarizer παράγει αναφορά στο τέλος.

Διατηρείται καθυστέρηση αρκετά μικρότερη του ενός δευτερολέπτου ανά παράθυρο, ώστε να λειτουργεί ομαλά η προσομοίωση.

### Διάθεση και Ρυθμίσεις

Χρήση Docker Compose πολλαπλών containers με ελέγχους καλής λειτουργίας, απομονωμένο δίκτυο και κοινά volumes. Μόνο το Frontend εκτίθεται εκτός, οι υπόλοιπες υπηρεσίες μένουν εντός εσωτερικού δικτύου. Base image `python:3.12-slim`, και προαιρετικές ομάδες εξαρτήσεων (dependencies) μέσω του εργαλείου uv. Κεντρικό YAML αρχείο για ρυθμίσεις αρχείων καταγραφής, προσομοίωσης, παραμέτρων concept drift, παραμέτρων επαναεκπαίδευσης.

### Επίδοση και Παρατηρησιμότητα

Στόχος: 72000 δευτερόλεπτα μέσα σε λίγα λεπτά με αξιολόγηση δεδομένων σε χρονικά παράθυρα. Επιτυγχάνεται με επιτάχυνση 300 φορών, επεξεργασία περισσότερων από 110000 trips συνολικά, με ανανέωση 12 ανιχνευτών ταυτόχρονα, και συνεχή διαδραστικότητα. Σε αυτό βοηθάνε οι μετρικές/ειδοποιήσεις που διατηρούνται μόνο στη μνήμη και όχι σε κάποια βάση δεδομένων, τα προϋπολογισμένα χαρακτηριστικά σε Parquet, η χρήση ξεχωριστών διεργασιών για επεξεργαστικά βαριές λειτουργίες όπως επαναεκπαίδευση και ανιχνευτές, ώστε να μην μπλοκάρεται το Global Interpreter Lock (GIL), και η χρήση ασύγχρονων κλήσεων εισόδου/εξόδου. Διατηρούνται αρχεία καταγραφών και χρησιμοποιείται ομαλή υποβάθμιση.

### Επεκτασιμότητα και Αρχές Σχεδίασης

#### Backend Ανεξάρτητο από το Μοντέλο

Ορίζει παράθυρα με αρχή και τέλος, συγκεντρώνει σφάλματα, διαχειρίζεται τον συντονισμό της επανεκπαίδευσης χωρίς γνώση χαρακτηριστικών. Προσθήκη νέου task γίνεται με προσθήκη νέου predictor container και εγγραφή στο config.

#### Μοτίβο Χαρακτηριστικών

Για τις πολλαπλές προβλέψεις μέσα στην προσομοίωση, χρησιμοποιούνται προϋπολογισμένα χαρακτηριστικά. Για τις προβλέψεις από τη διεπαφή χρήστη, χρησιμοποιείται ένα FeatureCalibrator που υπολογίζει χαρακτηριστικά σε πραγματικό χρόνο.

#### Μοτίβο Δεδομένων

DataLoader με φιλτράρισμα του Parquet αρχείου με βάση παράθυρο δωσμένο με ένα αρχικό και ένα τελικό timestamp, με δυνατότητα αλλαγής σε κάποια βάση, ή ροή δεδομένων μέσω Kafka ή κάποιας βάσης σε σύννεφο (cloud), χωρίς αλλαγή κλήσεων και χωρίς να χρειάζεται να τροποποιηθεί ο κώδικας.

#### Πολιτική Προσαρμογής και Συναίνεσης

Consensus των drift detectors, μέγεθος αρχικού παραθύρου ανοχής (grace), μέγεθος παραθύρου συλλογής post-drift, όλα ρυθμιζόμενα από παραμέτρους στο κοινό αρχείο YAML.

### Ισχυρά Σημεία και Συμβιβασμοί

#### Ισχυρά Σημεία

Σαφή όρια κάθε υπηρεσίας, HTTP χωρίς κατάσταση, ενιαίος ορχηστρωτής με κεντρική κατάσταση, ασύγχρονα αιτήματα και απομόνωση διεργασιών, απλό μητρώο μοντέλων για προβλέψιμες αντικατάστασεις εν λειτουργία.

#### Συμβιβασμοί

Αποθήκευση μόνο στη μνήμη για ταχύτητα έναντι της διατήρησης ιστορικότητας, προϋπολογισμένα features για ταχύτητα έναντι μίας πλήρως online προσέγγισης με πιθανές καθυστερήσεις, χρήση ενός post-drift παραθύρου για σταθερότητα στην εκπαίδευση έναντι μικρής καθυστέρησης. Όλοι οι συμβιβασμοί μπορούν να βελτιωθούν με λύσεις όπως χρήση βάσης δεδομένων, online παραγωγή χαρακτηριστικών, χρήση σταδιακής εκπαίδευσης, χωρίς μεγάλες αλλαγές και με εύκολη επέκταση του κώδικα.

## Αποτελέσματα

### Τελικό Μοντέλο

Αρχικά παρουσιάζουμε τα αποτελέσματα του τελικού μοντέλου (χωρίς retraining) για κανονικές και βροχερές συνθήκες. Κανονικά: MAE 30.36 s, MAPE 13.20%. Βροχή: MAE 56.93 s, MAPE 19.02%. Η βροχή σχεδόν διπλασιάζει το σφάλμα, ξεκάθαρη ύπαρξη concept drift.

### Προσομοίωση και Προσαρμογή

Στη συνέχεια παρουσιάζουμε τα αποτελέσματα της προσομοίωσης 20 ωρών. Η πρώτη μέρα (10 ώρες) έχει κανονικές συνθήκες, η δεύτερη (10 ώρες) έχει βροχή. Στη μέση ακριβώς εμφανίζεται το απότομο concept drift και το σύστημα το ανιχνεύει, κάνει μία σύντομη επανεκπαίδευση σε 1 ώρα δειγμάτων με βροχή και κάνει hot-swap το μοντέλο. Επομένως, το επανεκπαιδευμένο μοντέλο χρησιμοποιείται για τις προβλέψεις από τη στιγμή που έγινε η αλλαγή και έπειτα. Σε αυτό το παράθυρο, παρατίθεται η παρακάτω σύγκριση των αποτελεσμάτων του αρχικού και του επανεκπαιδευμένου μοντέλου.

Αρχικό: MAE 43.27 s, MAPE 16.89%. Retrained: MAE 32.27 s, MAPE 14.55%. Κέρδος: −11.00 s MAE (−25.4%) και −2.34 pp. MAPE (−13.9%).

Τα παραπάνω αποτυπώνονται και στο γράφημα Figure 2 που ακολουθεί.

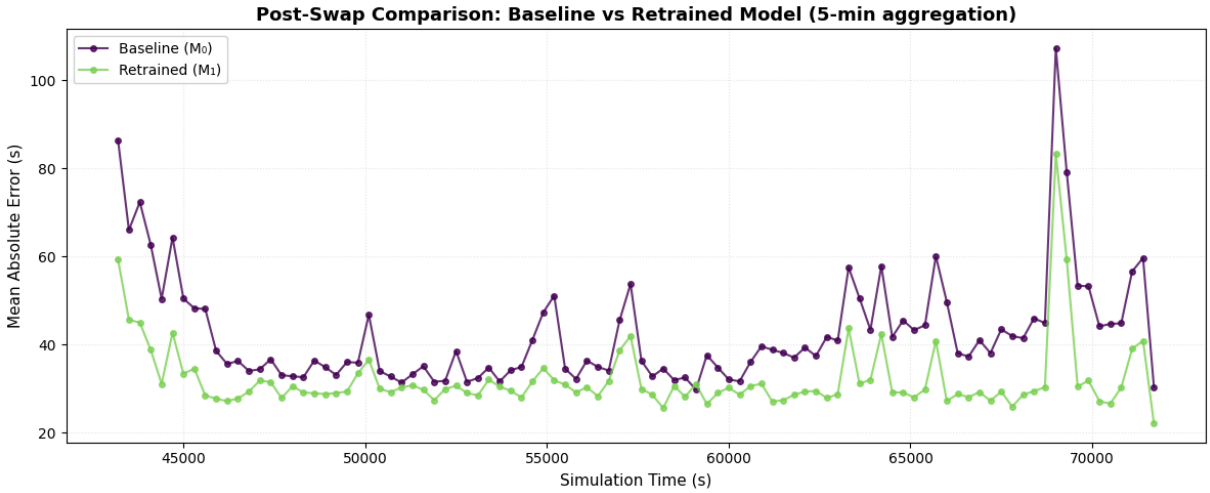


Figure 2: Σύγκριση μετά την Προσαρμογή: Τελικό και Επανεκπαιδευμένο Μοντέλο.

### Συμπεράσματα

Τα συμπεράσματα είναι: (1) Το concept drift ρίχνει την απόδοση του τελικού μοντέλου αισθητά. (2) Στοχευμένο retraining σε μικρό παράθυρο μετά το drift + hot-swap έχουν ως αποτέλεσμα ουσιαστική ανάκτηση ακρίβειας. (3) Μικρό κόστος χρόνου, μεγάλο πρακτικό όφελος από την πλατφόρμα, χωρίς απώλεια λειτουργίας κατά την αντικατάσταση μοντέλου.

### Επίδειξη Πλατφόρμας

Ακολουθούν και δύο στιγμιότυπα οθόνης από την πλατφόρμα στα γραφήματα Figure 3 και Figure 4, όπου φαίνονται ο πίνακας ελέγχου και η διεπαφή χρήστη. Ο πίνακας ελέγχου δείχνει τη ροή: Start → Stable → Drifted (σήμανση και άνοδος σφαλμάτων) → Retraining (αντικατάσταση ανά task) → Retrained (τέλος προσαρμογής). Η διεπαφή χρήστη επιτρέπει επιλογή διαδρομής και on-demand προβλέψεις που αντικατοπτρίζουν την προσαρμογή, ενώ η τελική αναφορά συνοψίζει αυτή την προσομοίωση.

## Συμπέρασμα

### Σύνοψη Συνεισφορών

Η διπλωματική σχεδίασε, υλοποίησε και επικύρωσε μια πλατφόρμα συνεχούς μηχανικής μάθησης με επίγνωση drift για εφαρμογές CCAM, με περιοχή μελέτης το κέντρο της Αθήνας. Παρήχθησαν: (α) αναπαράξιμο pipeline δεδομένων στο SUMO (train, test και ελεγχόμενο drift μέσω μείωσης τριβής για βροχή), (β) μοντέλο ETA με gradient boosting και στοχευμένη εξαγωγή χαρακτηριστικών, (γ) πλατφόρμα μικροϋπηρεσιών «κλειστού βρόχου» που αναπαράγει δεδομένα, παρακολουθεί σφάλματα, ανιχνεύει drift με consensus πολλαπλών ανιχνευτών, επανεκπαιδεύει και αντικαταστεί μοντέλα χωρίς downtime, και (δ) μεθοδολογία αξιολόγησης σε time-lapse που έδειξε ανίχνευση, προσαρμογή και ανάκτηση απόδοσης. Κύρια ευρήματα: (i) το ρεαλιστικό, ελεγχόμενο drift είναι ουσιώδες για μετρήσιμες τακτικές προσαρμογής, (ii) η συναίνεση ανιχνευτών σταθεροποιεί τις αποφάσεις, (iii) η επανεκπαίδευση στο στοχευμένο post-drift παράθυρο αποκαθιστά αποτελεσματικά την ακρίβεια, και (iv) η λειτουργική ορατότητα (πίνακας ελέγχου και ειδοποιήσεις) είναι προϋπόθεση εμπιστοσύνης και γρήγορης διάγνωσης.

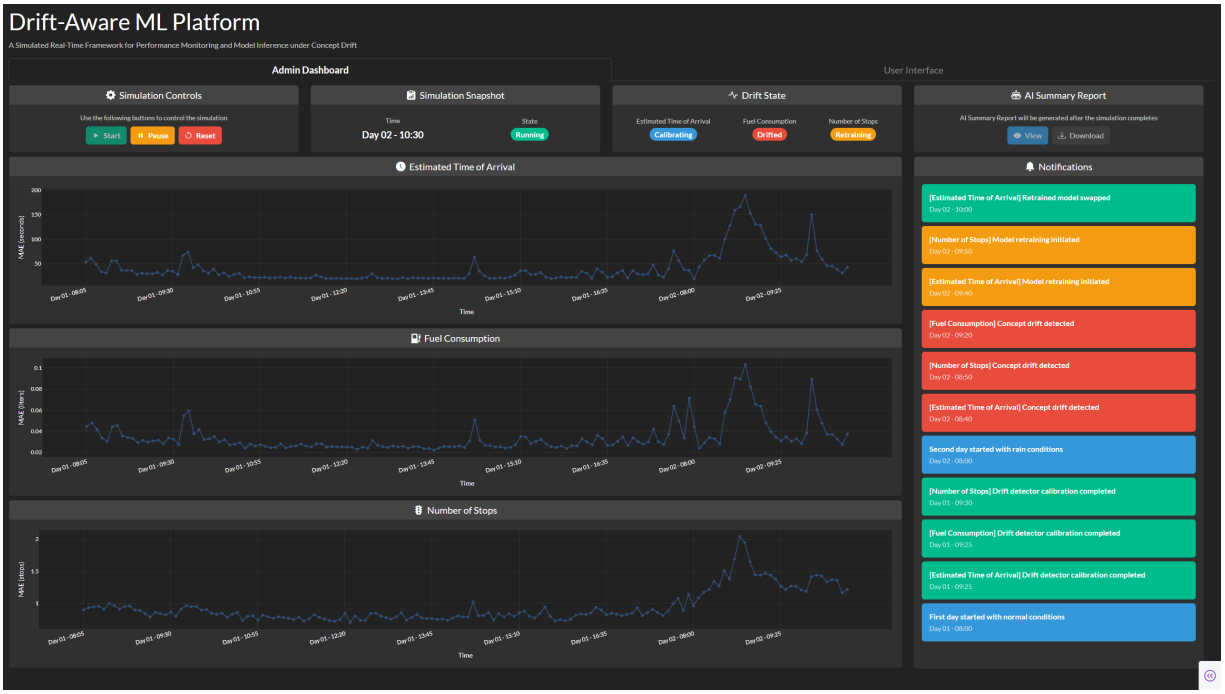


Figure 3: Πίνακας ελέγχου — *Επανεκπαίδευση*. Το ETA μοντέλο έχει επανεκπαιδευτεί και αντικατασταθεί, το μοντέλο Στάσεων επανεκπαιδεύεται ακόμα, ενώ το μοντέλο Καυσίμου δεν έχει ξεκινήσει επανεκπαίδευση ακόμα.

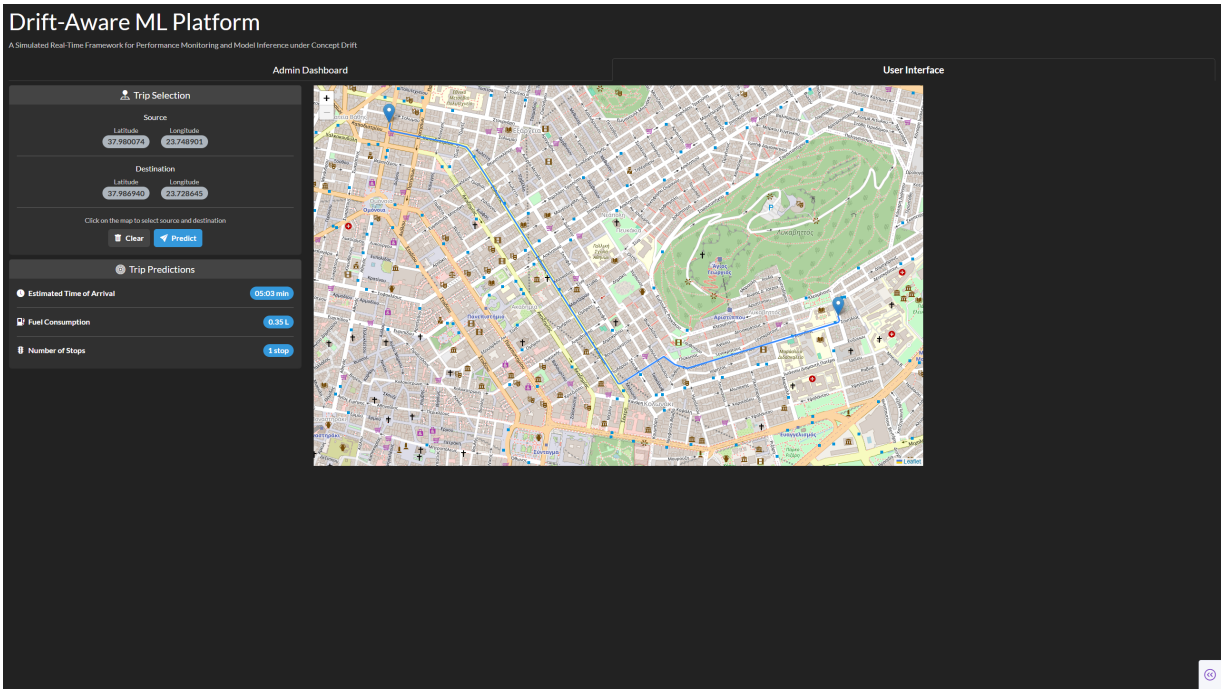


Figure 4: Διεπαφή χρήστη — *Επανεκπαίδευση*. Μία επιλεγμένη διαδρομή δείχνει αυξημένο χρόνο ETA λόγω επανεκπαίδευσης και αντικατάστασης του: `05:03 min`, `0.35 L`, `1 stop`.

## Περιορισμοί και Μελλοντικές Εργασίες

Οι περιορισμοί αφορούν τα συνθετικά δεδομένα (όχι πλήρως ετερογενή ή πολύπηγα), προϋπολογισμένα χαρακτηριστικά (υποτιμούν τις δυσκολίες πραγματικού online serving χαρακτηριστικών) και διάθεση σε ένα κόμβο με Docker Compose. Ως μελλοντική εργασία προτείνονται η εισαγωγή πραγματικών ροών (connectors, καθυστερημένες επισημάνσεις), online/streaming μάθηση ως συμπλήρωμα της επανεκπαίδευσης, διάθεση στο σύννεφο με αυτόματη κλιμάκωση και ανθεκτικότητα, time-series βάση δεδομένων για ιστορικές μετρικές, και επιπλέον σενάρια μετατοπίσεων εννοιών (ατυχήματα, κλεισίματα δρόμων, ξαφνικές αλλαγές ζήτησης) για benchmarking προσαρμογών.

## Επίλογος

Συνολικά, αποδεικνύεται ότι μια πλατφόρμα ML με επίγνωση concept drift για αστική κινητικότητα είναι εφικτή και χρήσιμη: εντόπισε αλλαγές, προσαρμόστηκε έγκαιρα και επανέφερε την ποιότητα προβλέψεων, διατηρώντας παράλληλα διαφάνεια λειτουργίας. Η πρόκληση που ακολουθεί είναι η μετάβαση από το επικυρωμένο «sandbox» σε παραγωγή, με ετερογενή δεδομένα, καθυστερημένες επισημάνσεις και ανάγκη για κλιμάκωση των υπηρεσιών, ώστε το πρωτότυπο να εξελιχθεί σε αξιόπιστη υπηρεσία κινητικότητας.

# Chapter 1

# Introduction

## 1.1 Motivation

The rapid evolution of *Cooperative, Connected and Automated Mobility (CCAM)* is transforming modern transportation systems. CCAM technologies enable vehicles to communicate with each other and with infrastructure, promising safer, more efficient, and more sustainable mobility [1]. As intelligent transportation systems become increasingly reliant on machine learning for critical functions such as route optimization, traffic prediction, and safety-critical decision-making, the accuracy and reliability of these models directly impact both operational efficiency and public safety. Recent trends in CCAM emphasize the need for real-time adaptability and continuous model refinement as urban environments become more complex and data-driven [2].

However, achieving reliable predictions in such dynamic settings presents significant challenges. One key use case is *Estimated Time of Arrival (ETA)* prediction for vehicles, a core service in smart mobility that influences traffic management, logistics, and traveler information [2]. Many current mobility systems do not continuously recalibrate their prediction models as new data arrives [3].

## 1.2 Problem Statement

Urban mobility environments are highly dynamic and *non-stationary*, meaning that the statistical properties of traffic data change over time. Machine learning models trained on static historical data often assume a fixed data distribution, an assumption that rarely holds in real city conditions [4]. When deployed in a dynamic urban setting, a model's performance can gradually degrade as driving patterns shift due to *concept drift*, the evolution of the underlying data relationships over time [5]. This drift is triggered by many factors: for example, road network changes (e.g., construction) can alter traffic flow patterns significantly [42]. Even routine variations like weather and seasonal demand swings can lead to abrupt or gradual shifts in vehicle speeds and congestion levels. A model that performs well on dry, low-traffic days may become less accurate during heavy rain or rush-hour peaks if those conditions were not part of its training distribution [6]. The challenge is that static ML models struggle to maintain accuracy in the face of non-stationarity, as their predictive power deteriorates over time unless the models continually adapt to the evolving urban mobility data [4].

While concept drift detection and adaptation techniques have been well-studied in the litera-

ture [5, 7], what remains largely unaddressed is the lack of a *holistic, integrated platform* specifically tailored for CCAM environments. Existing solutions are often fragmented or confined to laboratory settings, lacking the integration of critical components needed for systematic drift management in distributed mobility scenarios. Current tools may address individual aspects, such as drift detection or model retraining, but they rarely incorporate traffic simulation, continuous monitoring, and adaptive retraining into one seamless, automated loop [8, 9].

This gap motivates the creation of a continuous machine learning platform for mobility: one that integrates traffic simulation to test "what-if" scenarios, live performance dashboards to observe errors as they evolve, drift detection algorithms to raise alerts when model performance degrades, and an automated retraining pipeline to deploy updated models. By monitoring performance through time rather than just offline metrics and triggering adaptation with minimal human intervention, we aim to ensure that mobility prediction models remain accurate and robust despite the ever-changing urban environment, ultimately supporting the safety and reliability requirements of CCAM applications.

Figure 1.1: Cooperative, Connected and Automated Mobility (CCAM).

## 1.3 Thesis Goal

The objective of this thesis is to design, implement, and evaluate a microservice-based platform for *continuous machine learning* in urban mobility applications. In essence, the goal is to create a *drift-aware ML platform* that supports end-to-end monitoring and adaptation of predictive models in a city traffic context.

To achieve this, we first establish a reproducible experimental foundation by developing a synthetic dataset generation pipeline using the SUMO traffic simulator [10]. This addresses a critical gap: the lack of standardized, publicly available drift benchmarks in urban mobility research. By leveraging SUMO's microscopic traffic simulation capabilities, we can generate controlled scenarios that include both stable conditions and deliberate drift events, such as weather-induced traffic changes, providing labeled data for model training and realistic drift testing. This synthetic approach ensures reproducibility, allows for systematic experimentation with different drift patterns, and enables validation of the platform's drift detection and adaptation mechanisms in a controlled yet realistic environment.

Building upon this data foundation, the platform will be built using a modular microservice architecture to ensure scalability and flexibility, where each component (orchestration, model serving, drift detection, etc.) operates as an independent service. Crucially, the platform is intended to be model-agnostic and domain-specific: it can host any machine learning model for mobility prediction tasks, but this thesis will demonstrate and evaluate it primarily through the lens of an Estimated Time of Arrival (ETA) prediction use case. We focus on ETA because of its importance in CCAM scenarios, helping vehicles and travelers coordinate effectively, and because it provides a tangible benchmark to test how well the platform handles concept drift. Upon detecting drift, it will automatically trigger model adaptation, i.e. retraining and model swapping, to self-correct the predictions.

Although ETA is the main case study, the system is designed to accommodate additional models. For example, external predictive modules for Fuel Consumption [12] and Number of Stops [13] will be integrated to showcase that multiple concurrent mobility models can coexist and benefit from the shared drift-aware infrastructure. By the end of this thesis, we aim to show that such a platform can be realized and that it effectively maintains model performance over time in a realistic urban traffic setting.

## 1.4 Contributions

This thesis makes several contributions to the state of the art in continuous machine learning for mobility:

- **Synthetic Dataset Pipeline:** We developed a reproducible, modular data pipeline that generates training and evaluation datasets using synthetic traffic data from Simulation of Urban MObility (SUMO) [10]. Focusing on a case study of central Athens, the pipeline takes an open-source city road network and creates realistic traffic scenarios (vehicles, routes, events) to produce rich labeled data. This approach ensures that experiments are repeatable and tunable, meaning that the pipeline can be re-run for any city region or traffic pattern by adjusting parameters, thus addressing the shortage of standard drift benchmarks in urban mobility.

- **ETA Prediction Model under Drift:** We designed a feature-engineered ETA prediction model using gradient-boosted decision trees (LightGBM) [17]. The model is trained on the above dataset to estimate travel times for vehicles, and we introduce domain-specific features to improve accuracy. More importantly, the model's performance is evaluated under highly-realistic simulation-based drift scenarios, e.g., a sudden shift from normal traffic to heavy rain conditions that reduce vehicle speeds. This allowed us to study how concept drift affects ETA prediction accuracy, and to validate that our model, and platform, can detect and respond to these changes.

- **Time-Lapse Drift Simulation:** We conducted a time-lapse simulation experiment to rigorously assess drift detection and adaptation in an end-to-end fashion. In this setup, the SUMO simulator replays an extended sequence where an initial period of stable conditions is followed by a clear drift event, in this case a transition from dry weather to a storm causing slower traffic. The platform's drift detectors monitor the real-time prediction error and successfully detect the concept drift when the environment changes. We then measure how the system's automatic adaptation, triggering model retraining with new data from the rainy period, helps recover the model's performance. This exploration provides a realistic

validation of the platform's ability to observe, detect, and mitigate drift in a controlled yet lifelike scenario.

- **Drift-Aware Platform Implementation:** We present the design and implementation of a platform for continuous machine learning in mobility, which is a key practical contribution of this work. The platform includes components for simulation replay that feed historical or synthetic data through the model as if in real time, live dashboarding of model performance metrics, so that users can visualize concept drift as it happens, pluggable drift detection algorithms with calibration tools to set detection thresholds and minimize false alarms, and an automatic retraining and deployment pipeline, which seamlessly replaces the old model with an updated one when drift is confirmed. The entire system is built with modern software frameworks, such as containerized microservices, REST APIs for model serving, and event-driven triggers for retraining, to ensure it can be deployed in real operational environments. By open-sourcing the platform and detailing its architecture, we enable other researchers and practitioners to reproduce our approach and extend it to their own cooperative and automated mobility applications.

## 1.5 Thesis Structure

The remainder of this thesis is structured as follows:

- **Chapter 2: Background** - Introduces the foundational concepts and tools underlying our work. We review machine learning basics relevant to continuous model updating, the notion of concept drift, such as types of drift and common detection methods, the SUMO traffic simulator and its functionalities for creating mobility scenarios, and the platform technologies, like microservices, Docker, and APIs, used to build our solution.
- **Chapter 3: Related Work** - Surveys existing literature and systems in areas pertinent to our thesis. We cover drift detection tools and frameworks, examples being streaming ML libraries and MLOps solutions for model monitoring, and review prior research on ETA prediction models in classic and intelligent transportation. Finally, we discuss how our approach compares to and innovates upon the current state of the art.
- **Chapter 4: Dataset and Modeling** - Details our methodology for creating the dataset and developing the ETA prediction model. We describe the data generation process using SUMO for the Athens case study, including how traffic demand patterns and drift scenarios, in this case weather changes, are configured. We then present the ETA model development, including feature engineering, hyperparameter tuning, model selection, and the overall training procedure and the set up of the experiments.
- **Chapter 5: Platform Architecture** - Provides a comprehensive overview of the drift-aware platform's design. We enumerate the microservices and their roles, explain how the system ingests data, either from simulation or future extensions to real streams, how drift detection algorithms are integrated, and how the platform orchestrates retraining and deployment of models. Implementation details of key components, such as the real-time dashboard, are given to illustrate how the system works in practice.
- **Chapter 6: Results** - Presents the results of our evaluation. We report the baseline performance of the ETA model under static conditions and then analyze its behavior under induced drift. We demonstrate the effectiveness of drift detection and quantify the gains

from automatic adaptation. The results validate the platform's ability to maintain model accuracy over time.

- **Chapter 7: Conclusions** - Summarizes the contributions and findings of the thesis. We reflect on the success of the continuous machine learning approach for CCAM applications and any limitations observed. Finally, we outline future research directions, such as improvements to platform scaling and monitoring, or integrating the system with real-time data streams.

# Chapter 2

# Background

## 2.1 Classical Machine Learning

Machine learning (ML) is a field of artificial intelligence devoted to building algorithms that can learn from data patterns, enabling systems to predict and make decisions with minimal human intervention. ML algorithms are typically divided into four broad categories: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning [14].

- **Supervised learning**: involves training models on labeled datasets by learning a function that maps input features to output labels.
- **Unsupervised learning**: finds patterns and structures in unlabeled data, as seen in clustering or dimensionality reduction algorithms.
- **Semi-supervised learning**: combines small amounts of labeled data with larger unlabeled datasets, enhancing efficiency in domains with limited annotation.
- **Reinforcement learning**: guides an agent to optimal behavior through trial and error, rewarding desirable actions.

A few representative machine learning models are [15]:

- **Linear Regression**: Used for predicting continuous variables and modeling the relationship between features and output. It retains mathematical simplicity while remaining a crucial benchmark in both research and forecasting.
- **Decision Trees**: Represent decisions with branching structures, offering interpretability and handling mixed-type features.
- **Random Forests**: Ensembles of decision trees, reducing variance and improving generalization.
- **Support Vector Machines (SVM)**: Classifies data by finding the optimal separating hyperplane, being effective in high-dimensional settings.
- **K-Nearest Neighbors (KNN)**: Classifies points based on the "nearest" labeled examples.
- **Gradient Boosting Frameworks (XGBoost, LightGBM, CatBoost)**: Ensemble methods that extend predictive power, speed, and robustness by aggregating the outputs

of numerous decision trees to make predictions. XGBoost is renowned for its scalability and performance in structured/tabular data [16]. LightGBM excels at speed and memory efficiency [17]. CatBoost handles categorical variables and is competitive in ranking and classification tasks [18].

## 2.2 Deep Learning

Deep learning is a subset of machine learning that uses artificial neural networks to model and solve complex problems, inspired by the structure and function of biological neurons [19]. These networks are composed of multiple layers of neurons, each layer processing the output of the previous layer to extract increasingly complex features. A typical neural network comprises an input layer, one or more hidden layers, and an output layer.

A few representative deep learning models are:

- **Feedforward Neural Networks (Multi-Layer Perceptrons)**: The core architecture for classifying or regressing fixed-size inputs.
- **Convolutional Neural Networks (CNNs)**: Specialized for spatial relationships, such as image data.
- **Recurrent Neural Networks (RNNs)**: Designed for sequential data, such as text or time series.
- **Transformer Models**: Modern architectures excelling in tasks ranging from natural language understanding to vision.

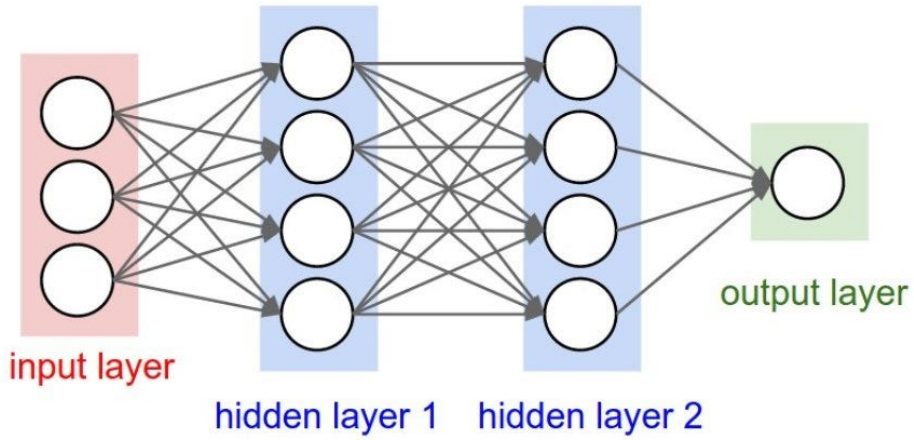


Figure 2.1: Feedforward neural network architecture with input layer, two hidden layers, and output layer.

Neural networks are flexible and scalable but can be data-hungry, prone to overfitting, and require careful regularization and tuning to achieve optimal results.

## 2.3 Model Performance Metrics

Model evaluation is essential for assessing predictive reliability. In the following metrics, $y_i$ denotes the true value, $\hat{y}_i$ represents the predicted value for the $i$-th observation, and $n$ is the total number of samples. Key metrics for regression models include [20]:

- **Mean Absolute Error (MAE)**:

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

  This metric conveys the average absolute prediction error, often valued for its interpretability and resistance to outliers.

- **Mean Absolute Percentage Error (MAPE)**:

$$MAPE = \frac{100}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i|}\,.$$

  MAPE expresses error as a percentage relative to true values, facilitating comparisons across diverse scales. However, MAPE can be unstable when true values approach zero. Other metrics (RMSE, $R^2$) complement these in providing insight into the variance, fit, and reliability of regression and classification models.

- **Root Mean Squared Error (RMSE)**:

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

  RMSE measures the average magnitude of the prediction errors (residuals) between predicted and actual values in regression tasks. It is more sensitive to large errors than MAE due to squaring, penalizing outliers heavily.

- $\mathbf{R^2}$ (coefficient of determination):

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

  $R^2$ measures the proportion of variance in the target variable explained by the model, ranging from 0 to 1. Values near 1 indicate strong fit, while $R^2 \leq 0$ means the model performs worse than a constant baseline.

## 2.4 Regularization, Overfitting, and Underfitting

Models risk overfitting—memorizing the training set at the cost of poor generalization—when they become excessively complex. Underfitting happens when the model is too simple, missing essential patterns.

Regularization combats overfitting by penalizing complexity:

- **L1 Regularization (Lasso)**: Encourages sparsity, zeroing non-critical coefficients.
- **L2 Regularization (Ridge)**: Shrinks coefficients smoothly, preferred for stability.

In neural networks, additional techniques such as dropout, early stopping, and weight decay further mitigate overfitting.

## 2.5 Cross-Validation

Cross-validation is the gold standard for robust model evaluation. By partitioning data into training and validation folds, cross-validation (often k-fold) averages performance over multiple splits, reducing variance and protecting against accidental overfitting to a single dataset segmentation. Model selection and hyperparameter tuning typically leverage cross-validation to ensure real-world generalizability

Some common cross-validation techniques are:

- **K-Fold Cross-Validation**: Partition the data into k equal folds, and then train on k-1 folds and validate on the remaining fold.
- **Stratified K-Fold Cross-Validation**: Partition the data into k equal folds, and then train on k-1 folds and validate on the remaining fold. The folds are stratified, meaning that the distribution of the target variable is kept the same in each fold.
- **Leave-One-Out Cross-Validation**: Partition the data into n equal folds, and then train on n-1 folds and validate on the remaining fold.
- **Repeated K-Fold Cross-Validation**: Repeat K-Fold Cross-Validation n times, and then average the performance over the n repetitions.

## 2.6 Model Selection and Hyperparameter Tuning

Selecting the right model and its hyperparameters is a critical decision in the ML workflow.

- **Model selection** often involves comparing candidate models (such as linear regression, decision tree, or boosting algorithms) using performance metrics evaluated on a validation set or through cross-validation.
- **Hyperparameter tuning** refers to optimizing parameters that govern model behavior but are not learned during training (e.g., learning rate, tree depth, regularization strength, number of layers in a neural network). Grid search systematically explores combinations, while random search samples the setup space. Advanced approaches such as Bayesian optimization build probabilistic models to efficiently home in on good hyperparameter settings.

## 2.7 Continuous Machine Learning

Continuous machine learning, also known as continual or lifelong learning, refers to a paradigm in machine learning where models incrementally update their knowledge by learning from new data streams over time, rather than being trained solely once on a static dataset. In contrast to traditional retrain-from-scratch approaches, continuous machine learning enables models to adapt to evolving environments, integrate recent information, and remain relevant as data changes. Key

strategies include incremental learning (updating parameters gradually), transfer learning, and experience replay.

A major challenge for continuous machine learning is preventing "catastrophic forgetting", the loss of knowledge about previously learned tasks when adapting to new data [21]. Effective approaches balance learning new patterns (flexibility) with retaining established skills (stability).

## 2.8 MLOps

MLOps, short for Machine Learning Operations, is a discipline at the intersection of machine learning and DevOps, focused on standardizing, automating, and streamlining the end-to-end lifecycle of machine learning models in production environments. Its goal is to bridge the gap between data science and IT operations, helping organizations move ML prototypes from research to production efficiently and reliably.

Core MLOps principles include [22]:

- **Versioning**: Tracking code, data, models, and experiment metadata to ensure reproducibility and traceability across iterations and deployments.
- **Automation**: Implementing CI/CD (continuous integration/continuous delivery) pipelines for data ingestion, model training, validation, deployment, and rollback, reducing manual workload and errors.
- **Monitoring**: Continuously assessing live models for data drift, concept drift, and performance degradation, enabling timely retraining, adaptation, and governance.
- **Collaboration**: Facilitating teamwork between data scientists, engineers, and domain experts with standardized workflows, shared environments, and modular architectures.
- **Governance and Security**: Enforcing access controls, audit trails, and compliance with regulatory standards, especially in sensitive domains.

MLOps empowers organizations to deploy ML solutions at scale, react to switching data environments, and maintain high service quality and reliability. Challenges involve cultural change, tool integration, and evolving infrastructure requirements. Prominent MLOps platforms include cloud-native solutions (AWS SageMaker, Google Vertex AI), as well as open-source toolkits like MLflow and Kubeflow.

## 2.9 Concept Drift

Concept drift is the phenomenon where the statistical properties of the target variable change over time, leading to reduced model accuracy [5]. Formally, concept drift occurs when the joint probability distribution changes over time:

$$P_t(X, y) \neq P_{t+k}(X, y)\,,$$

where $P_t(X, y)$ is the distribution at time $t$ and $P_{t+k}(X, y)$ is the distribution at some future time $t + k$. More specifically, drift can affect the conditional distribution $P(y|X)$ (real concept drift) or the input distribution $P(X)$ (virtual drift). This effect is especially critical in streaming and non-stationary environments, such as online monitoring, finance, or cybersecurity.

Concept drift manifests in various forms:

- **Sudden (abrupt)**: rapid changes in data distribution;
- **Gradual**: slow transitions between concepts;
- **Recurring (recurrent)**: cyclic or seasonal patterns;
- **Incremental**: stepwise, small alterations over time.

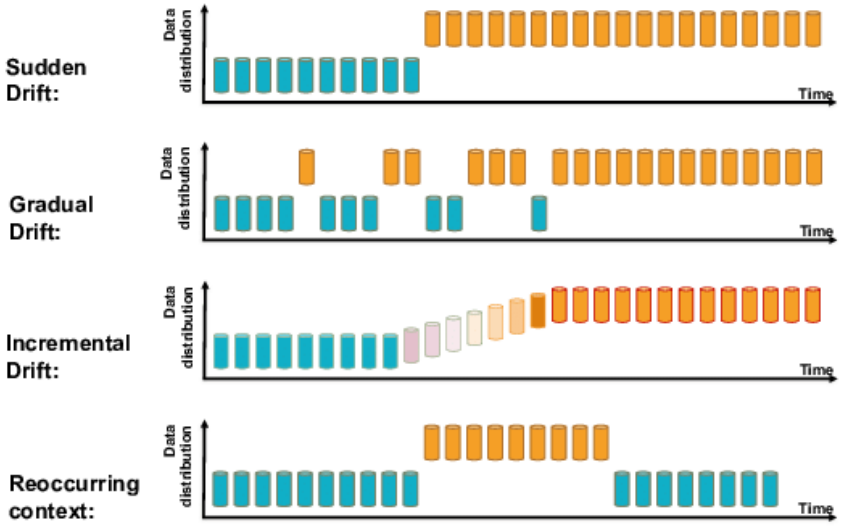


Figure 2.2: Categories of concept drift.

Detection strategies are categorized as error-rate monitoring (e.g., DDM, EDDM, HLFR), statistical tests, and ensembles. Drift detectors can monitor prediction error, data distribution, or model performance metrics. Notable techniques include Hierarchical Hypothesis Testing (HHT), Linear Four Rates (LFR), and ensemble detectors, each with distinct strengths relevant to the type and speed of drift.

Accurate and timely drift detection is paramount for maintaining predictive accuracy and reliability. Systems often employ sliding windows or reference distributions to test for shifts, and performance degradation often triggers retraining and adaptation processes.

Concept drift adaptation strategies help machine learning models stay accurate as data changes over time. The main approaches include frequent retraining (updating the model on new data), online learning (incremental updates with each new sample), window-based methods (using only recent data for training), and ensemble methods (combining several models and favoring those performing best on recent data) [4, 43]. Some strategies rely on drift detectors to trigger adaptation, while others update models continuously regardless of detected drift. The goal is to respond quickly to changes, maintaining reliable predictions in evolving environments.

## 2.10 Key Trade-offs in Drift Detection and Adaptation

Key trade-offs in drift detection and adaptation include [43]:

- **Sensitivity vs. False Alarms**: More sensitive detectors can spot small or early changes but risk frequent false positives. Less sensitive thresholds may delay detection, potentially degrading predictive performance.

- **Label Availability**: Some detectors require true labels to estimate error rates, limiting their use in unsupervised or delayed-label scenarios. Unsupervised methods can work on input distributions alone but often have less precision.
- **Latency**: Maintaining low-latency detection is crucial for real-time systems but may require sacrificing some detection accuracy or threshold tightness to avoid delays.
- **Computational Cost**: More advanced or frequent drift checks, especially those recalculating statistics or performing window comparisons, can significantly raise resource requirements, impacting scalability in high-volume streams.

Carefully balancing these trade-offs is essential when choosing drift detectors and adaptation strategies for different operational contexts, such as real-time monitoring, batch analytics, or resource-constrained environments.

## 2.11 SUMO Simulator

SUMO (Simulation of Urban MObility) is an open-source, microscopic and continuous multi-modal traffic simulation suite designed to model and analyze the movement of individual vehicles, pedestrians, and various transportation modes on large-scale road networks [44]. Each agent, be it a car, bus, bicycle, or pedestrian is simulated explicitly, with unique routes, behaviors, and interactions, allowing for fine-grained analyses of traffic dynamics and management strategies [45].

Core functionalities and capabilities include:

- **Microscopic and Multi-modal Simulation**: Models each vehicle, pedestrian, bicycle, and public transport unit individually, supporting realistic behaviors such as lane changes, right-of-way rules, traffic light interactions, and intermodal trips within the same environment.
- **Flexible Input and Demand Modeling**: Supports importing road networks from OpenStreetMap, commercial formats (VISUM, Vissim, NavTeq), or synthetic networks, and generates vehicle flows using origin-destination matrices, real traffic counts, or virtual population-based demand models.
- **Traffic Management and Policy Testing**: Enables the evaluation of traffic lights, routing policies, eco-aware navigation, and urban policies before real-world deployment.
- **Real-time Control and Interoperability**: Offers APIs like TraCI for live, programmatic control of the simulation and integration with external tools, such as vehicle communication (C2X) simulators.
- **Visualization and Output Analytics**: Provides comprehensive visualization tools and generates detailed simulation outputs including vehicle trajectories, trip statistics, traffic states, and environmental metrics in XML or CSV formats.
- **High Performance and Open Source**: Efficiently manages large networks (10,000+ edges; 100,000+ vehicles) across multiple platforms, and is freely available with an active developer community for extensions and research applications.

These features make SUMO an established tool for transportation planning, intelligent transportation system development, traffic light optimization, and research on smart mobility solutions worldwide.

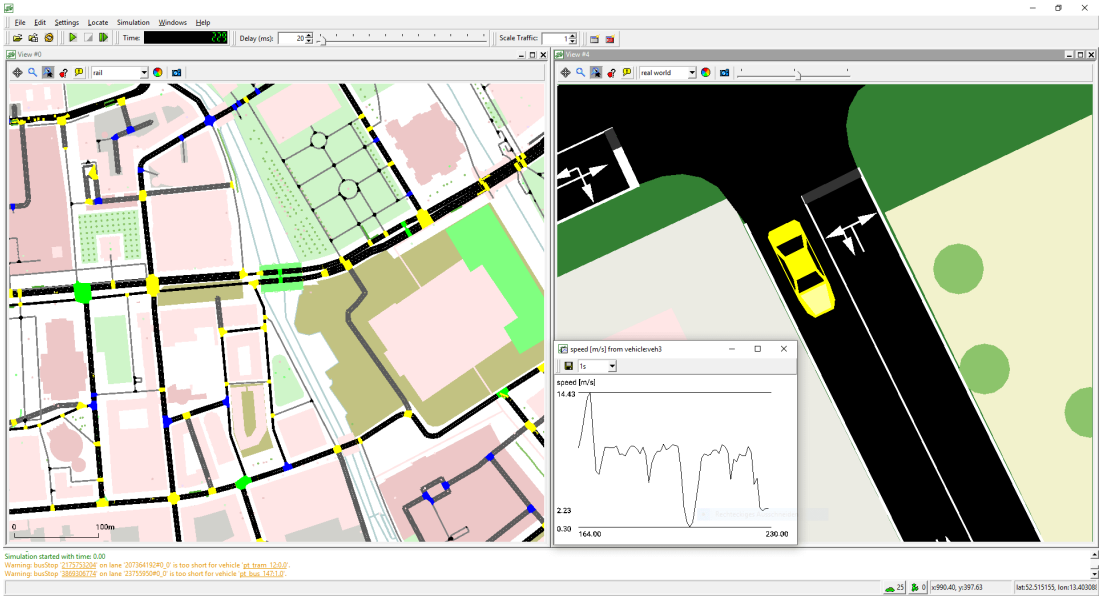

Figure 2.3: SUMO simulation environment showing a microscopic traffic simulation with individual vehicle trajectories, real-time visualization, and speed monitoring capabilities.

## 2.12 Containerization and Orchestration

**Docker** is a widely adopted containerization platform that allows the packaging of software and its dependencies into isolated units called containers [46]. This isolation ensures that code runs the same way regardless of the underlying environment, solving the notorious "it works on my machine" problem. Docker containers are lightweight, highly portable, and enable versioned environments. Academic use cases include reproducible computational research, easy sharing of code and data, and simplifying complex setups for fellow researchers. In large organizations and regulated industries (e.g. healthcare, finance), Docker provides essential security, agility, and compliance by isolating sensitive data and quickly deploying new services.

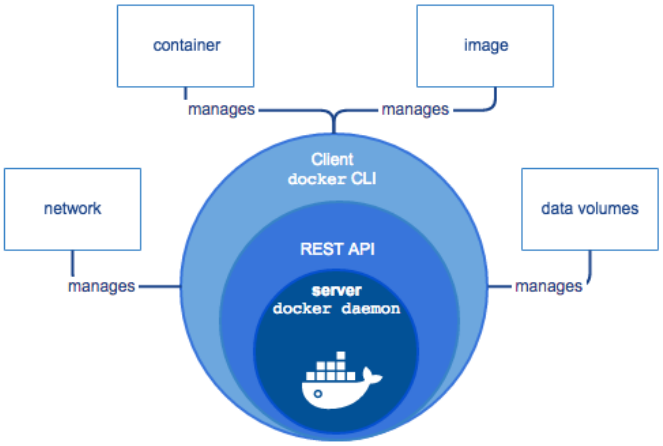


Figure 2.4: Docker components and control flow.

**Docker Compose** builds on Docker by orchestrating multiple containerized services (such as databases, backends, frontends) through simple configuration files [47]. This facilitates local development of microservice architectures, comprehensive integration testing, and staging realistic production environments before deployment.

## 2.13 Web Protocols and API Design

**HTTP** is the foundational protocol for web and API communications, enabling clients and servers to exchange requests and responses in a standardized format, including JSON and XML payloads.

**REST APIs** are the core architectural style in web services for interoperable communication over HTTP [41]. By strictly adhering to stateless client-server principles, REST APIs allow disparate systems, ML pipelines, data stores, external user interfaces, to work together seamlessly.

## 2.14 Backend and Frontend Frameworks

**FastAPI** is a modern web framework for building RESTful APIs with Python [48]. Distinctively, FastAPI leverages Python type hints and asynchronous programming, allowing developers to write concise, robust, and extremely fast endpoints. Its native compatibility with OpenAPI and automatic documentation greatly reduces manual effort and errors. FastAPI is widely used in both prototypes and mission-critical systems due to its speed and strong validation.

**Dash** is a Python framework for creating interactive web dashboards [49]. With minimal Python code, data scientists and engineers can build sophisticated analytics apps that integrate visualizations, user inputs, and RESTful API endpoints. Dash leverages Flask under the hood, but developers can extend with additional Python frameworks as required for interoperable, customized solutions.

**Plotly** is an extensible visualization library available on top of Dash, useful for interactive charts, data exploration, and embedding analytics in dashboards [50].

## 2.15 Data Validation and Storage

**Pydantic** is the leading Python library for data parsing and validation using type hints [51]. By providing strict enforcement of schemas at runtime, Pydantic allows developers to safely handle incoming data—whether from users, APIs, or databases—preventing subtle bugs and security risks. Models inherit from BaseModel and use Python's type annotations to declaratively describe fields, their types, and validation logic. When instantiating a model, Pydantic parses and validates input, raising structured errors for invalid data. Its integration with FastAPI allows for automatic generation of API documentation and validation of request and response data.

**Apache Parquet** is a column-oriented data format optimized for analytical workloads in big data settings [52]. Parquet's design enables highly efficient compression and encoding, dramatically reducing storage costs and query times, especially compared to row-based formats like CSV.

# Chapter 3

# Relevant Work

## 3.1 Tooling Landscape for Drift Detection and Monitoring

### 3.1.1 River

**River** is an open-source Python library for online machine learning on streaming data, born from the merger of the Creme and scikit-multiflow projects [27]. It provides a single unified framework to train models incrementally, one sample at a time, which makes it well-suited to scenarios with continuous data and potential concept drift. River includes a variety of learning algorithms (e.g. linear models, Hoeffding trees, ensemble methods) that can update their parameters on the fly, as well as evaluators and metrics that update incrementally along with the model. Notably, River treats concept drift as a first-class concern: it implements plug-and-play drift detection methods like ADWIN, DDM, and EDDM to signal when the statistical properties of the data or the model's error rate have changed significantly. A major strength of River is its efficiency and low overhead for streaming applications, it avoids expensive retraining by doing continuous machine learning, enabling rapid adaptation to changing data. However, River's focus on single-instance updates means it may not directly leverage batch acceleration or GPU computing as efficiently as some batch frameworks. Overall, River is focused on streaming adaptation and concept drift handling, making it a powerful tool for real-time ML systems.

### 3.1.2 scikit-multiflow

**scikit-multiflow** is an earlier Python framework for stream learning and concept drift, which was one of River's predecessors [28]. Inspired by the Massive Online Analysis (MOA) stream mining software in Java, scikit-multiflow brought popular stream learning algorithms and evaluation tools into the Python ecosystem. It supports a range of tasks including classification, regression, and even multi-output prediction, and it provides several drift detection algorithms (e.g. ADWIN, DDM, EDDM, Page-Hinkley) as part of its toolkit. A key design goal was compatibility with scikit-learn, allowing users to integrate stream learning with familiar APIs. The strengths of scikit-multiflow include its breadth of implemented methods, many inherited from the MOA framework, and the ability to simulate data streams with built-in generators for controlled concept drift, e.g. abrupt or gradual drift in synthetic data. However, scikit-multiflow is no longer actively maintained and its functionality has effectively been subsumed into River, which offers a more consolidated and improved interface. In practice, new projects prefer River for streaming tasks, but scikit-multiflow remains a reference point in the literature and is still useful for benchmarking

new drift detection techniques due to its comprehensive implementation of classical methods.

### 3.1.3 Alibi and Alibi Detect

**Alibi** is an open-source library by Seldon Dev focused on machine learning model inspection and explainability, offering techniques like counterfactual explanations and influence scores for black-box models, for example explaining why a prediction was made. Building on this, the Seldon team also released Alibi Detect, which is a library specifically for outlier detection and concept drift detection in ML applications [29, 53]. **Alibi Detect** provides a collection of both offline and online detectors for various data types: tabular data, text, images, and time series. It includes statistical methods, like Kolmogorov-Smirnov or Chi-square tests for distribution drift in features, as well as learned detectors, like drift detection using neural network embeddings for complex data such as images. A notable focus of Alibi Detect is support for different modalities and integration with modern ML stacks: it can use TensorFlow or PyTorch under the hood, which allows it to implement advanced detectors, for example a PCA-based outlier detector or a classifier-based drift detector, that leverage GPUs. The strength of Alibi/Alibi Detect lies in its broad coverage: it tackles not just drift but also outliers and even adversarial example detection in one package, making it a flexible choice for production monitoring where explainability and data quality are needed alongside drift detection. One limitation is that Alibi Detect is a lower-level library, it provides algorithms but not a full monitoring solution. Users must integrate it into their workflow, for example, through scheduling periodic drift checks or responding to detector alerts in an MLOps pipeline. Additionally, configuring the more complex detectors may require expert knowledge, like setting thresholds or choosing embedding models. In summary, Alibi for explainability and Alibi Detect for drift/outliers offer a robust toolkit focused on model insight and data drift, well-suited for use cases where understanding model behavior and detecting distribution shifts are both important.

### 3.1.4 Evidently

**Evidently** is an open-source Python library and toolset designed to evaluate, test, and monitor machine learning models and data in production [30, 53]. It provides pre-built reports and dashboards to analyze things like data drift, target drift, data quality, and model performance over time. One of Evidently's core features is the ability to generate an interactive HTML dashboard comparing a reference dataset, like the training data or last week's data, to a current dataset. The dashboard includes visualizations and statistical tests that highlight any significant changes in feature distributions or model outputs. It also tracks model performance metrics, if ground truth is available. The strengths of Evidently include its ease of use and rich visualization, as, with minimal code, a user can produce a comprehensive report that would otherwise require manual analysis. This makes it a popular choice for integrating into Jupyter notebooks or automated pipelines to regularly check for data drift and model decay. A current limitation is that Evidently's real-time capabilities are evolving, considering it was initially built for batch analysis and reporting. Using it in a truly live monitoring scenario with continuous streaming data may require additional effort, such as writing a job to compute metrics on rolling windows. Additionally, while it identifies drift and performance issues, it doesn't automatically retrain models or suggest fixes, as it's primarily a monitoring dashboard. In short, Evidently fills the role of ML monitoring and data validation by making drift detection and model quality checks more accessible and transparent.

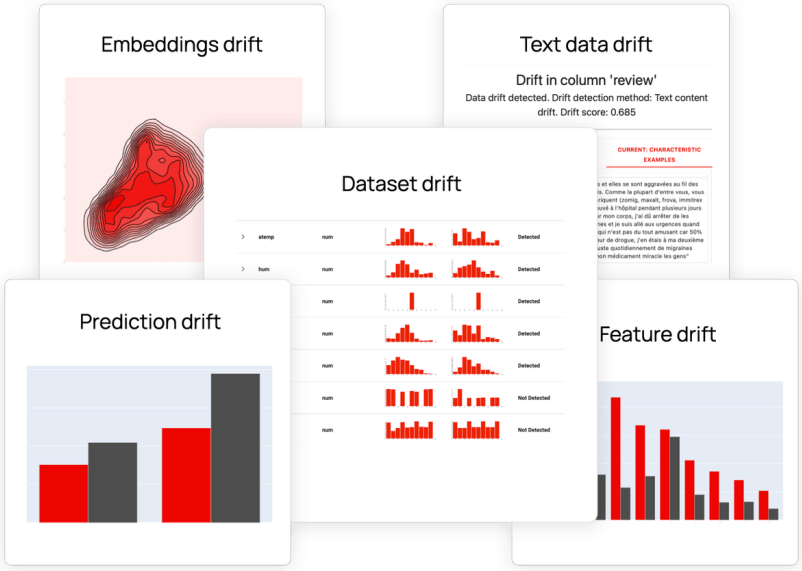


Figure 3.1: Evidently: Data drift evaluation.

### 3.1.5 NannyML

**NannyML** is a newer open-source library specifically focused on post-deployment model monitoring, with an emphasis on detecting concept drift and estimating model performance without immediate ground truth [31, 53]. It aims to answer the question: "How is my model performing now, given that I might not have labels for the latest predictions?". To this end, NannyML provides tools for data drift detection, as well as algorithms to infer performance drop. A centerpiece is its Confidence-Based Performance Estimation (CBPE) method, which uses the model's prediction probabilities and detected data drifts to estimate metrics like accuracy or ROC AUC in production before real labels arrive. In practical terms, NannyML can alert you that "your model's inputs have shifted significantly, and it estimates your model's accuracy has dropped from 0.85 to 0.70, for example", allowing proactive retraining or investigation. Another innovative feature is the "reverse drift" (RCD) concept: assessing how changes in input data would be expected to affect the target distribution and thus the model's error, essentially linking drift to business impact. The strength of NannyML lies in this performance-centric approach to drift: it goes beyond flagging that data has changed, by quantifying the likely effect on model predictions and outcomes. This is particularly useful in industries where obtaining true labels is slow or expensive (e.g., credit risk, where you only know defaults after many months). NannyML's limitations include that it currently supports primarily classification tasks (binary and multiclass) for its performance estimation module. The techniques can be complex to calibrate. For instance, CBPE assumes the model's probability estimates are well-calibrated and that past relationships hold, which might not always perfectly predict actual performance. Additionally, as a specialized library, it may need to be integrated into an existing MLOps setup, as it provides Python APIs and some UI components, but not a full monitoring server on its own. In summary, NannyML represents an advanced approach to concept drift monitoring: instead of just detecting drift, it focuses on what that drift means for model accuracy, helping close the gap between drift signals and model maintenance actions.

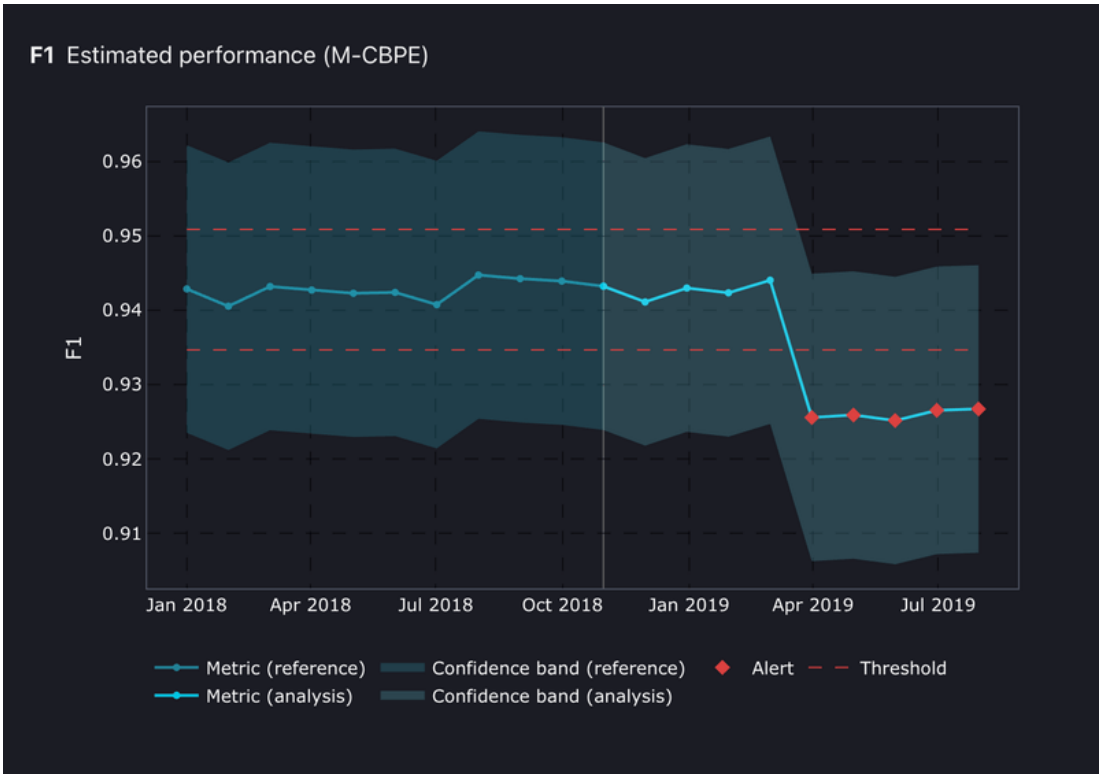


Figure 3.2: NannyML: Monitoring estimated performance with the CBPE method

### 3.1.6 Summary and Limitations

While the libraries surveyed above (River, scikit-multiflow, Alibi Detect, Evidently, and NannyML) offer powerful building blocks for drift detection and online learning, they share a common limitation: they are primarily toolkits rather than complete systems. Each requires manual integration into an MLOps pipeline, often leaving critical gaps such as automated retraining, model hot-swapping, and end-to-end orchestration to the user. Furthermore, none provide a cohesive solution that combines drift detection with automatic adaptation and user-facing interfaces in a single deployable platform. In contrast, this work presents a fully integrated microservices architecture that addresses these gaps by implementing consensus-based drift detection, automatic model retraining and deployment, multi-task prediction, real-time simulation integration, and interactive dashboards for both monitoring and end-user predictions. By demonstrating this end-to-end workflow in a realistic traffic prediction scenario, the platform showcases how drift detection theory translates into production-ready systems with minimal manual intervention, closing the loop from detection to adaptation in a way that standalone libraries cannot.

## 3.2 ETA Prediction: Literature Overview

### 3.2.1 Classical Time-Series Methods

Predicting travel time or estimated time of arrival (ETA) is a well-studied problem in the transportation domain, and a variety of machine learning approaches have been explored. Early works approached ETA prediction with classical time-series models like ARIMA, treating travel time as a time-dependent process [54]. While such statistical models worked in limited settings, they

struggled with the inherent non-linearities and context dependencies of traffic data.

### 3.2.2 Deep Learning Approaches

Over the past decade, the field has seen a surge of deep learning methods that capture complex spatial and temporal patterns in traffic networks. For example, recurrent neural networks (RNNs) and long short-term memory (LSTM) architectures have been used to model vehicle trajectories as sequences of GPS points, successfully learning temporal dependencies such as road segment travel times and rush-hour effects [32, 33]. Convolutional neural networks have been applied to encode spatial information. One notable approach is to treat a path's GPS sequence like an image or matrix (e.g., via a grid or along-route segmentation) and use CNNs to extract features [55]. More recently, attention mechanisms and transformers have been introduced. AttentionTTE [56] employs self-attention to capture global interactions among road segments, combined with a recurrent module for local temporal trends. This achieved state-of-the-art results on a large trajectory dataset, indicating the benefit of modeling both local and long-range dependencies in routes. Similarly, researchers have explored multi-task learning in deep models, for instance, the WDR (wide-deep-recurrent) model [57] and CoDriver system [58] incorporated an auxiliary task (learning driver behavior style) to improve ETA predictions, demonstrating that additional contextual signals can improve accuracy.

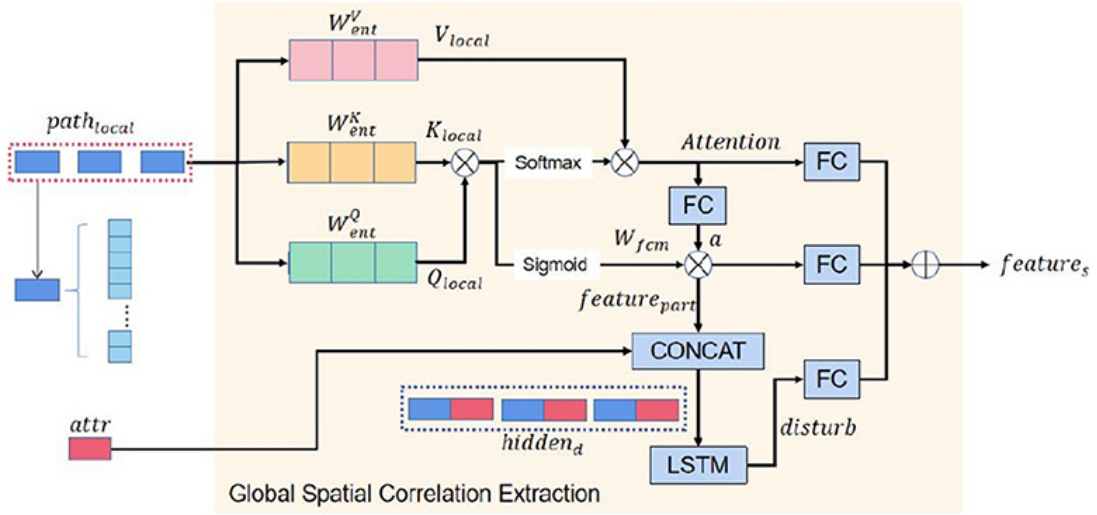


Figure 3.3: AttentionTTE: The self-attention of spatial correlations extraction module.

### 3.2.3 Graph Neural Networks

In parallel to sequence-based methods, graph-based approaches have gained prominence for ETA and traffic prediction. In a road network, intersections and road segments can be naturally represented as nodes and edges in a graph, so graph neural networks (GNNs) can explicitly model the connectivity and traffic flow between road segments. Another notable approach is a GNN-based ETA model deployed in production for Google Maps [59], which learns from massive amounts of traffic data while incorporating road network topology and dynamic conditions (like accidents or rush hour patterns). By using GNN layers to propagate information along the road graph, their model can account for upstream slowdowns or congestion that classical models might miss. This graph-based ETA estimator yielded significant improvements in real-world accuracy,

for example, reducing large ETA errors by over 40% in some cities compared to the previous baseline.

Other studies have followed similar ideas, building spatio-temporal graph models that combine GNNs with temporal sequence modeling (e.g. graph convolution plus LSTM) to predict travel times under varying conditions [60, 61]. These advanced deep learning models represent the current state-of-the-art, especially when rich data is available (historical trajectories, real-time traffic sensors, etc.). They tend to excel at capturing complex patterns, but at the cost of higher complexity and data requirements.

### 3.2.4 Gradient Boosting and Hybrid Models

Despite the success of deep neural approaches, tabular data methods remain highly relevant, especially in industry settings where interpretability, simplicity, or limited data are considerations. Many practical ETA prediction systems reduce the problem to a regression on engineered features: for example, features might include the route distance, expected traffic speed, number of traffic lights, time of day, day of week, weather conditions, and so on. Gradient boosting decision trees (GBDT) have been a popular choice for such tabular formulations [34, 35]. GBDT models, implemented with tools like XGBoost or LightGBM, offer strong performance on structured data and have the advantage of producing feature importance insights, which is valuable for understanding ETA drivers.

Zhang and Haghani (2015) demonstrated that with carefully constructed input features, including real-time traffic indices, a GBDT model can achieve accuracy comparable to more complex models, though it may need frequent updating to handle drift. Another recent work applied XGBoost, CatBoost, and LightGBM to predict bus arrival times in a smart city context, showing that these boosted tree models are competitive for ETA and can be tuned to high accuracy given domain-specific feature engineering [62]. In their case, gradient boosting achieved around 30% mean absolute percentage error for autonomous bus ETAs, providing a solid baseline for comparison.

The continued popularity of gradient boosting in this domain is due to several factors:

1. Tabular features can encode a lot of prior knowledge, for instance, the road speed limit or historical average travel time on a segment is an informative predictor that a tree can easily use.

2. Training and inference are fast and resource-efficient compared to deep sequence models, which is useful for deployment on edge devices or with limited infrastructure.

3. The models are easier to maintain and one can retrain a new boosted tree model on recent data periodically, and examine feature importances if the model starts to degrade.

Of course, a limitation is that these models do not automatically capture sequential or spatial relationships unless those are manually turned into features.

Therefore, the current trend in ETA research and applications is often a hybrid one: use domain knowledge to engineer strong features and/or combine outputs of simpler models, and, where possible, incorporate the structure of the problem (sequential or graph) via specialized models or ensemble approaches. Well-known benchmarks, such as the NYC Taxi trip duration dataset [63] or the Google Maps ETA data [59], often show that gradient boosting with rich features can be very effective, coming close to deep learning methods when the latter are not heavily optimized.

In summary, ETA prediction methods span from deep learning models leveraging spatio-temporal structure to gradient-boosted tree models on tabular features, and the choice often depends on available data and the need for interpretability versus maximum accuracy.

## 3.3 Novelty and Contributions of the Proposed Platform

The platform developed in this thesis brings together ideas from concept drift research, MLOps, and domain-specific traffic simulation to create a closed-loop adaptive learning system for mobility. Here we summarize the key novel contributions of this platform, and how it differentiates itself from existing tools and literature.

### 3.3.1 Controlled Drift Simulation with SUMO

In order to study concept drift in a realistic yet controlled manner, we utilize a traffic simulation, performed with SUMO - Simulation of Urban MObility. SUMO allows us to generate synthetic traffic data under varying conditions with fine control. A novel aspect is the creation of a time-lapse scenario in which we deliberately introduce a sudden change in environment. For example, the platform simulates 10 hours of normal driving conditions followed by 10 hours of heavy rain, where rain is modeled by globally reducing the road friction coefficient in the SUMO network. This causes vehicles to slow down and drive differently, thereby inducing concept drift in the input data and the relationship between features (e.g. speeds, distances) and targets (travel time, etc.). By having this capability to manufacture drift on demand, we can evaluate how well different detectors and adaptation strategies perform on a known "ground truth" drift event. Traditional drift detection research often relies on either real-world events, which are unpredictable, or synthetic data generators for drift. Our use of SUMO bridges that gap by producing realistic, domain-specific drift.

This approach is relatively unique. To our knowledge, few if any published works have demonstrated a full ML workflow where a traffic microsimulator is used to inject controlled drifts and then automatically detect and counteract them in real time. Moreover, the platform continuously tracks the model's performance during the simulation; as the rain scenario unfolds, we log the model's prediction error metrics (e.g. mean absolute error for ETA) to observe how performance degrades due to drift. This provides an authentic demonstration of concept drift's impact on a machine learning model deployed in a dynamic environment.

### 3.3.2 Multi-Task Prediction and Monitoring

The platform is designed to handle multiple related prediction tasks simultaneously, which is a departure from most drift detection case studies that focus on a single prediction output. In our implementation, a set of separate models, produces predictions for ETA, Fuel Consumption, and Number of Stops for a given vehicle trip. This multi-task setup reflects a more complex operational scenario.

From a drift detection perspective, the platform monitors drift on multiple data streams and multiple performance metrics concurrently. This also means that using the platform, one can observe how a drift in the environment can have different manifestations across tasks (e.g., rain might drastically affect ETA and fuel consumption but not the predicted number of stops if stops depend more on route layout than speed). But this option opens another possibility to the end user, which is to compare the performance of similar models, or different flavors of the same

model, on the same data stream, to find the one that has the better behavior, when drift is present. This is novel compared to existing tools like River or NannyML which typically assume a single target.

The above mentioned options required developing a custom monitoring strategy that can aggregate signals from multiple drift detectors and multiple outputs. We implemented a simple consensus mechanism with multiple drift detectors, where each model had its own copy of the drift detectors, running in parallel and independently. This consensus mechanism reduces false alarms in a noisy multi-task setting. Supporting multiple tasks in one platform is a practical contribution because real-world systems often have to monitor dozens of metrics, our work provides a template for how to achieve that in an automated way.

### 3.3.3 Automated Model Retraining and Hot-Swapping

Perhaps the most significant contribution of the platform is the seamless closed-loop adaptation capability. When concept drift is detected, the platform doesn't just raise an alert. It initiates a retraining pipeline and deployment of an updated model, without human intervention. Concretely, the system maintains a buffer of recent data (e.g., the last 1 hour of vehicle trips) and when drift is confirmed, it triggers a retraining job that fine-tunes or re-fits the ML model using this new data distribution. We implement this efficiently for the gradient boosting models, such as the one used for ETA, by retraining the model on the drifted data, based on a warm start on top of the previous one.

Once the new model is trained, the platform performs a hot swap. The running system begins using the new model for all subsequent predictions, with no downtime at all.

This kind of automated retraining and deployment is often discussed in MLOps but not so commonly demonstrated end-to-end in academic literature. The novelty here is showing the entire loop: we not only detect drift, we close the loop by adapting to it immediately. This required solving engineering challenges (e.g., ensuring that the system can retrain while continuing to serve predictions, avoiding a halt or latency spikes) and design challenges (deciding when retraining is worth it to avoid model churn). The outcome is an architecture where model performance degradation is not only observed but automatically countered by an update, moving towards the ideal of a self-healing ML system.

### 3.3.4 Real-Time Dashboard and Visualization

Finally, in order to make the system's operations interpretable and to aid in human-in-the-loop oversight, the platform includes a real-time dashboard that visualizes the current state of the simulation and the ML models. This dashboard (built with Dash/Plotly) displays key information such as charts of performance metrics over time, and a timeline of platform events, with emphasis on drift detections, retraining triggers, and model hot-swaps.

When a drift is detected, a retraining job is triggered and a model swap occurs, the dashboard visibly flags these event separately, for example showing the following notifications: "[ML Task]: Drift detected at 10:30" and "[ML Task]: Model retrained and hot-swapped at 11:00". Subsequent improvements in error can then be observed, showing the effect of the adaptation process.

Beyond passive monitoring, the dashboard exposes a user interface meant to be used in a human-in-the-loop manner. Users can request a specific sample to be predicted (in this traffic simulation, a specific origin-destination trip) and the platform will infer the predictions from the current

models and present them. This enables the user to quantify how a fixed sample's prediction changes before drift, during drift, and after adaptation, enhancing the understanding of the system's behavior.

While a dashboard itself is not a novel research contribution, integrating it end-to-end with the platform and coupling it with the user interface closes the feedback loop by allowing users to watch the ML system adapt in real time. Many existing drift tools like Evidently produce static reports or require the user to inspect logs; in contrast, our system's UI shows a continuously updating view, which is especially important in a streaming context. This also helps communicate the results, effectively illustrating the benefit of the adaptive approach.

### 3.3.5 Conclusion

In summary, the platform distinguishes itself by combining drift generation, detection, *and* adaptation in one unified environment and by making their effects inspectable through an interactive dashboard. Unlike existing monitoring libraries that stop at detection, our system covers the full lifecycle of a realistic mobility scenario. It manufactures a drift (rainy weather) in a traffic simulation, observes the model's decline, detects it via multiple detectors, and then remedies it through automated retraining, all while delivering insights through a real-time interface. This end-to-end closed-loop approach, applied in the context of cooperative, connected and automated mobility, is a novel contribution demonstrating how advanced MLOps techniques, such as continuous monitoring and automatic adaptation, can be applied to maintain model performance in non-stationary environments.

# Chapter 4

# Dataset Generation and Machine Learning Research

## 4.1 Dataset Generation

This chapter details the construction of a synthetic, yet faithful, urban-traffic dataset for central Athens and the methodology used to train and evaluate an Estimated Time of Arrival (ETA) model under controlled concept drift.

The data were generated with the SUMO microscopic simulator and released in three complementary scenarios: *train*, *test*, and *rain*, each spanning 10 hours of simulated time with per-second Floating Car Data (FCD) telemetry (vehicle position, speed, lane, odometer, fuel, waiting).

The *rain* scenario introduces a physically interpretable distributional shift by uniformly reducing road-surface friction across the network, yielding a measurable degradation in speed and a significant increase in trip durations.

The pipeline emphasizes reproducibility (fixed seeds, parameterized configs), modularity (clean separation of network preparation and per-scenario simulation), and portability to other regions and drift magnitudes.

### 4.1.1 Study Area

The study area is a dense, signal-controlled rectangle in central Athens, selected for its non-trivial junction structure and representative city-center congestion. The final network comprises 1,184 edges and 689 junctions, covering 68.17 km of roadway, a scale large enough to expose complex queueing and dynamics relevant to ETA modeling.

The study area can be seen in Figure 4.1.

Network bounds and more detailed summary metrics are reported in Table 4.1.

### 4.1.2 Network Construction

Rather than relying on SUMO's interactive tooling like the `osmWebWizard` GUI, the network build process invokes `osmGet.py` and `osmBuild.py` directly from scripts, with specific options specified. These scripts are responsible for extracting the OSM data [64] for the specified bounding box

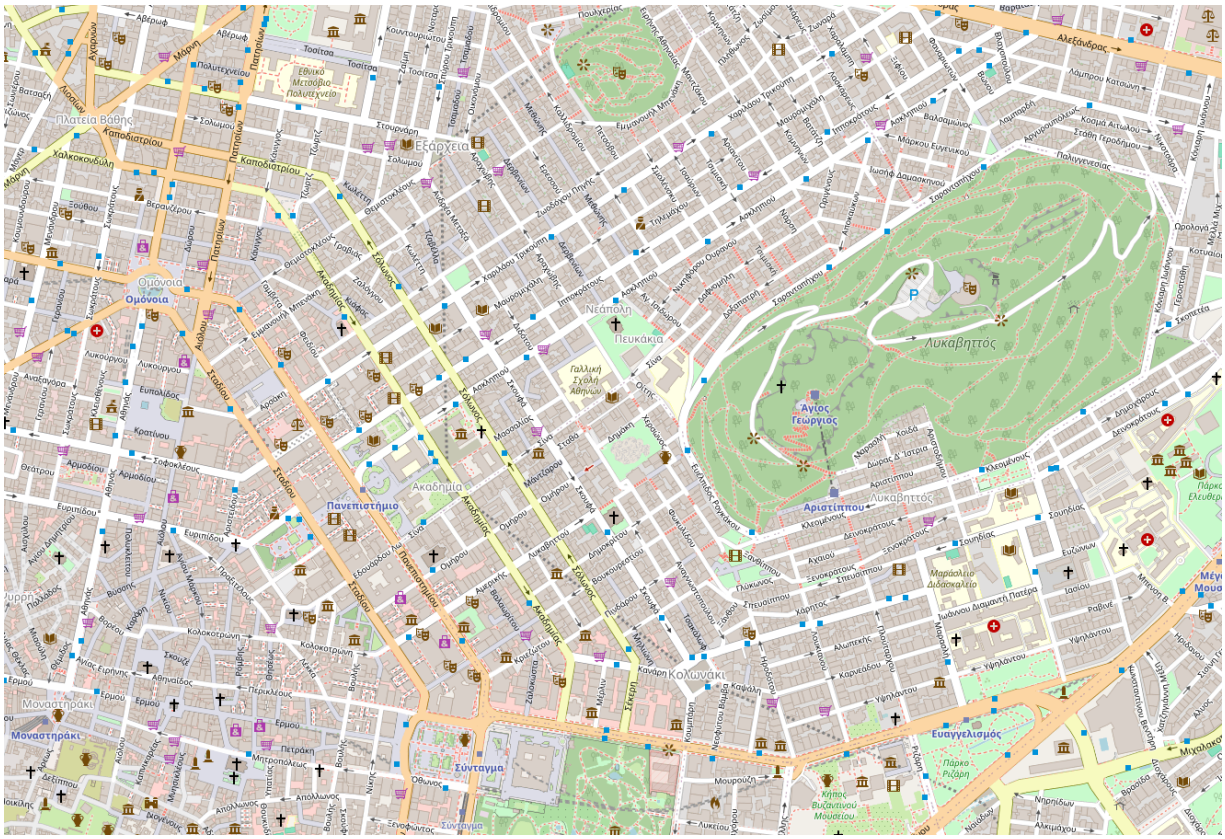


Figure 4.1: Study area in central Athens.

Table 4.1: Network characteristics of the central Athens study area.

| **Metric** | **Value** |
|---|---|
| North-West Boundary | (37.974745936977456, 23.725252771719436) |
| South-East Boundary | (37.988290142332225, 23.752735758169127) |
| Size | 2.42 km × 1.48 km |
| Area | 3.58 km$^2$ |
| Edges | 1184 |
| Junctions | 689 |
| Traffic Lights | 106 |
| Total Road Length | 68.17 km |
| Average Edge Length | 57.57 m |

and building the SUMO network by converting the OSM data into a SUMO-compatible network format. This choice affords programmatic reproducibility, gaining full access to the internal `netconvert` and `polyconvert` options that are not exposed by the GUI, and headless, automated runs without manual interaction. The base network is generated once from OpenStreetMap extracts for the specified bounding box, with traffic-light inference and junction geometry refinement. A second variant is then created by cloning the base network and uniformly applying a reduced lane friction coefficient of 0.4 to all lanes for the *rain* scenario. This design preserves topology and routing feasibility across scenarios, isolating the physical effect of reduced friction from confounds induced by network edits.

### 4.1.3 Vehicle Classes

Vehicle classes are limited to passenger cars to avoid heterogeneity that would affect drift analysis and feature learning. Excluding motorcycles and heavy vehicles reduces behavioral variance (e.g., lane splitting or heavy acceleration/deceleration profiles) and simplifies interpretation while retaining the primary dynamics needed for ETA modeling. These vehicle classes would account for a small percentage of the total vehicle fleet, with the motorcycles holding a higher percentage, and being rather difficult to model accurately due to their heterogeneous driving behavior [65]. By focusing on the dominant vehicle class, passenger cars, the dataset maintains methodological simplicity while capturing the primary traffic dynamics relevant to the prediction tasks and research objectives.

### 4.1.4 Traffic Demand Modeling

Traffic demand follows a realistic hourly pattern designed to mimic real-world Athens traffic patterns observed in sources such as the Athens Mobility Observatory [36] and similar traffic monitoring platforms. Considering that we opted for 10 hour simulation, we chose a pattern that would capture characteristic morning and evening rush hours with a midday decline typical of urban traffic, in between the hours of 08:00 and 18:00. The base generation periods, in seconds between vehicle departures, are shown in Table 4.2.

Table 4.2: Hourly demand schedule for 08:00–18:00. Period denotes the generation interval (seconds per vehicle). Trips/hour are approximated as 3600/period.

| **Hour (local)** | **Period (s/veh)** | **Approx. Trips/hour** |
|---|---|---|
| 08:00–09:00 | 0.50 | ∼7200 |
| 09:00–10:00 | 0.55 | ∼6545 |
| 10:00–11:00 | 0.65 | ∼5538 |
| 11:00–12:00 | 0.75 | ∼4800 |
| 12:00–13:00 | 0.80 | ∼4500 |
| 13:00–14:00 | 0.80 | ∼4500 |
| 14:00–15:00 | 0.75 | ∼4800 |
| 15:00–16:00 | 0.65 | ∼5538 |
| 16:00–17:00 | 0.65 | ∼5538 |
| 17:00–18:00 | 0.60 | ∼6000 |

To introduce natural variability between simulations and avoid identical traffic patterns, each scenario applies Gaussian noise to the base traffic generation periods. Each base period is multi-

plied by a random value drawn from a normal distribution centered at 1.0 with standard deviation 0.01. These values are chosen empirically to introduce some *stochastic variability* in traffic volumes while preserving the overall hourly pattern shape, and to avoid introducing too much noise.

Each scenario uses a distinct random seed that controls the following:

- Gaussian noise applied to traffic generation periods
- Random trip generation (origin-destination pairs)
- Departure and arrival positions on edges
- Vehicle behavior stochasticity in SUMO

Based on the above, the trip generation is performed using SUMO's `randomTrips.py` tool. Random origin-destination pairs are generated for each scenario. Trips are distributed according to the hourly traffic generation periods, after Gaussian noise application, validated for route feasibility, and assigned random departure/arrival positions on edges. This process is performed separately for each scenario (*train*, *test*, *rain*) using the respective random seeds.

### 4.1.5 Concept Drift Scenario

Several candidate drift mechanisms were evaluated prior to finalization.

**Lane Closure Approach** This approach used SUMO's `closingLaneReroute` mechanism to mark specific lanes as closed, with vehicles calculating routes at insertion time. The critical issue was that vehicles would stop at green traffic lights when approaching closed lanes, remaining stationary until SUMO's automatic teleportation mechanism activated after 300 seconds—a clear simulation failure.

The root cause was the absence of rerouting devices. Adding them would allow dynamic recalculation around closures, but this also meant that all vehicles would reroute based on real-time conditions, fundamentally altering traffic behavior compared to the base scenario. As a result, isolating the effect of closures from dynamic rerouting became impossible, and the scenario was abandoned.

**Network Topology Modification** This approach used SUMO's `netedit` tool to physically remove closed lanes or edges, with junctions automatically recalculated. Removing edges reduced network capacity, causing vehicles to be inserted at different times due to congestion delays—breaking temporal comparability between scenarios.

The sensitivity was highly non-linear and unpredictable: closing major roads like Panepistimiou, a main thoroughfare, sometimes produced minimal impact, while closing minor edges would completely bottleneck the network. Network analysis metrics, such as betweenness centrality and edge importance, were used to identify strategic closures, but finding combinations that were realistic (e.g., metro construction, roadworks) while producing detectable but not catastrophic drift proved extremely difficult.

**Vehicle Behavior Modification** This approach altered Krauss car-following model parameters (tau and sigma) [66], default vehicle type attributes such as acceleration and deceleration, following gaps, and speed factors to simulate aggressive or cautious driving patterns.

Table 4.3: Distributional shifts between baseline test scenario and rain scenario with reduced friction.

| Metric | Test (Baseline) | Rain (Drift) | Change |
|---|---|---|---|
| Average Speed | 29.84 km/h | 25.76 km/h | **-13.68%** |
| Trip Duration | 211.58 s | 247.00 s | **+16.74%** |
| Trip Distance | 1677.53 m | 1685.18 m | **+0.46%** |
| Waiting Time | 19.78 s | 22.02 s | **+11.33%** |
| Fuel Consumption | 216300.76 mg | 218780.58 mg | **+1.15%** |

Behavior changes produced only subtle effects on aggregate traffic patterns, didn't correspond to clear real-world events, unlike rain or construction, and lacked ground truth for validation of "realistic" parameter ranges. This approach was also deemed unsuitable.

**Conclusion - Rain Scenario** Based on the above, the final drift mechanism uses friction-based rain simulation. The rain scenario introduces concept drift by simulating adverse weather conditions through reduced road surface friction. This approach was chosen because:

- **Physical realism**: Rain directly reduces tire-road friction, a well-understood phenomenon
- **Interpretability**: Clear causal relationship between friction and vehicle behavior
- **Measurable impact**: Reduced friction increases braking distances, decreases acceleration, and lowers average speeds
- **Network preservation**: No topology changes—all routes remain valid
- **SUMO support**: Native friction parameter in lane definitions

As mentioned earlier, a modified network is created by parsing the base network XML and applying a global friction reduction to all lanes. The friction parameter is set to 0.4, down from the default 1.0, uniformly across all lanes in the network. This modified network is saved separately and used for the rain scenario simulation.

The chosen value of 0.4 technically falls within the snow/ice range according to road surface friction coefficients, rather than the wet road range ($\mu \approx 0.5$-$0.8$) [11]. This decision was intentional: the objective was to produce a significant and detectable concept drift for model evaluation, rather than to precisely simulate realistic rain conditions. The 0.4 coefficient ensures measurable performance degradation while maintaining simulation stability and avoiding extreme scenarios that would lead to complete traffic collapse.

This decreased friction affects vehicle dynamics by altering the maximum acceleration and deceleration, cornering speed, and emergency braking, and more. The result is that vehicles in the rain scenario experience slower acceleration from stops, earlier and gentler braking, longer trip completion times, and different congestion patterns.

The rain scenario introduces measurable distributional shifts compared to the baseline test scenario, as seen in the following Table 4.3.

The most significant impacts are on average speed (reduced by 13.68%) and trip duration (increased by 16.74%), demonstrating clear concept drift that challenges machine learning models

trained on baseline conditions.

### 4.1.6 End-to-End Generation Pipeline

The complete pipeline consists of 9 steps:

1. **OSM Data Extraction** → Download OpenStreetMap data for Athens bounding box
2. **Network Building** → Convert OSM data to SUMO network format
3. **Rain Network Creation** → Generate friction-modified network for drift scenario
4. **GUI Settings** → Write SUMO-GUI visualization settings
5. **Configuration Files** → Generate simulation configuration files per scenario
6. **Trip Generation** → Create random origin-destination pairs per scenario
7. **Simulation Execution** → Run SUMO simulation per scenario
8. **Format Conversion** → Convert CSV output to Parquet format
9. **Exploratory Analysis** → Generate statistics and plots

The first 4 steps are performed only once, while the remaining 5 steps are performed once for each scenario (*train*, *test*, *rain*).

### 4.1.7 Reproducibility and Extensibility

The pipeline is designed for full reproducibility and extensibility with all configuration parameters centralized in a YAML configuration file. Each step is an independent function with well-defined inputs and outputs. All parameters are stored in a YAML configuration file with structured dataclass access. Random seeds control all stochastic elements (traffic noise, trip generation, vehicle behavior). All commands, outputs, and errors are logged with timestamps. Path existence and command success are validated at each step.

The pipeline can be easily adapted to generate new datasets by modifying configuration parameters. Key customization points include:

- **Geographic Area:** Change the bounding box to simulate any OSM-covered region
- **Traffic Demand:** Modify hourly traffic patterns
- **Traffic Volume Noise:** Adjust the mean and standard deviation of the Gaussian noise
- **Random Seeds:** Use different seeds to generate alternative traffic patterns
- **Simulation Duration:** Extend or shorten the simulation timespan
- **Drift Parameters:** Adjust network friction to vary rain severity
- **Network Processing:** Modify netconvert options to change junction detection and traffic light logic
- **Rerouting Behavior:** Adjust adaptation steps and intervals to change vehicle re-routing aggressiveness
- **Vehicle Classes:** Include buses, trucks, or other vehicle types

Table 4.4: Comprehensive dataset characteristics across all three scenarios.

| Metric | Train | Test | Rain |
|---|---|---|---|
| **Purpose** | Model training | Model evaluation | Concept drift testing |
| **Network** | Base (friction=1.0) | Base (friction=1.0) | Rain (friction=0.4) |
| **Random Seed** | 13 | 2025 | 314159 |
| **Simulation Duration** | 36000 s (10 hours) | 36000 s (10 hours) | 36000 s (10 hours) |
| **Total Trips** | 53,978 | 56,212 | 55,366 |
| **Average Trip Duration** | 205.22 s | 211.58 s | 247.00 s |
| **Average Trip Distance** | 1675.81 m | 1676.36 m | 1684.88 m |
| **Average Speed** | 30.24 km/h | 29.81 km/h | 25.74 km/h |
| **Total FCD Records** | 11,130,801 | 11,948,843 | 13,728,897 |
| **CSV Size** | 770.1 MB | 826.8 MB | 946.9 MB |
| **Parquet Size** | 233.3 MB | 247.4 MB | 282.3 MB |

This modular design allows researchers to reproduce the exact datasets described in this thesis or generate new variants for different experimental conditions.

### 4.1.8 Output Format

The output format is Floating Car Data (FCD) with per-timestep vehicle telemetry at 1-second resolution:

- **timestep**: Simulation time (seconds)
- **id**: Vehicle identifier
- **x**, **y**: Position coordinates on the network (meters)
- **speed**: Instantaneous speed (m/s)
- **lane**: Current lane ID
- **odometer**: Cumulative distance (m)
- **fuel**: Fuel consumption rate (mg/s)
- **waiting**: Time spent waiting since last stop (s)

To optimize downstream analytics, the raw CSV is converted to Parquet (columnar, compressed), which substantially reduces storage and accelerates feature extraction and aggregation.

### 4.1.9 Dataset Characteristics

The final dataset, containing the three scenarios (*train*, *test*, *rain*), has the following characteristics, summarized in Table 4.4.

### 4.1.10 Data Availability

The dataset generated and used in this thesis is publicly available as version 4, comprising three scenarios (*train*, *test*, *rain*) and dual formats (CSV and Parquet). It is archived on Zenodo under the DOI 10.5281/zenodo.16950674 [37].

## 4.2 Machine Learning Research

### 4.2.1 Overview

This section presents the research process for building a machine learning model for Estimated Time of Arrival (ETA) prediction on the generated Athens dataset. Objectives were to (i) establish strong baselines across model families, (ii) evaluate feature engineering strategies, (iii) select a final model through systematic tuning and cross-validation, and (iv) report performance and training efficiency.

Key outcomes: the final selected model is LightGBM [17], achieving an MAE of 26.51 s and a MAPE of 12.77% with a training time of 2.95 s. Competing gradient-boosting methods, XGBoost [16] and CatBoost [18] performed almost the same in accuracy but were slower to train under comparable configurations.

### 4.2.2 From FCD to Trips

The SUMO FCD output provides per-second telemetry for each vehicle: `timestep`, `id`, `x`, `y`, `speed`, `lane`, `odometer`, `fuel`, `waiting`.

To obtain a trip-level dataset suitable for supervised learning, records were grouped by vehicle identifier and aggregated to derive the following features:

- **time_start:** first observed timestep (seconds)
- **duration:** last minus first timestep (seconds)
- **source_x, source_y:** origin coordinates (meters)
- **destination_x, destination_y:** destination coordinates (meters)
- **distance:** final odometer reading (meters)

To ensure data quality and model reliability, the generated trips were filtered based on the following criteria:

- **Minimum Duration:** duration $\geq$ 30 seconds (exclude extremely short trips)
- **Minimum Distance:** distance $\geq$ 200 meters (exclude stationary or negligible movement)

These thresholds remove edge cases such as vehicles that failed to complete routes, experienced immediate insertion errors, or represented non-meaningful trips that would introduce noise into the training process.

The transformation yields a trip-level dataset with 53,229 samples, where each row represents a complete vehicle trip characterized by:

- **Origin-Destination Pair:** Source and destination coordinates
- **Temporal Context:** Departure timestamp
- **Route Characteristics:** Distance traveled
- **Trip Duration:** Target variable

This trip-level representation forms the foundation for feature engineering and model training, enabling prediction of trip duration based on spatial, temporal, and route characteristics.

### 4.2.3 Experimental Methodology

**Experiment tracking.** A systematic experiment tracking framework was implemented to manage the iterative research process and ensure reproducibility across all experiments. Experiments were organized as scripted runs with consistent I/O structure, saved artifacts (models, metrics, logs), and fixed random seeds. This ensured reproducibility, reduced boilerplate, and enabled reliable comparisons. For this to happen, utilizing a shared library of code for common functions and tasks was necessary. A centralized configuration YAML file was also used to store all shared settings and parameters, ensuring consistency across all experiments. This approach also enabled the use of automated result collection scripts. These scripts would aggregate metrics across all experiments and have quick and easy access to comparative visualizations and CSV exports for further analysis.

**Cross-validation.** Model selection used 5-fold cross-validation with stratification over the target (trip duration) to stabilize fold distributions and reduce variance in estimates. An important aspect of this is preventing the model from overfitting to specific ranges of trip durations, if only such ranges are present in the training data. The test and rain datasets were kept strictly for final evaluation and were never used for training or model selection to avoid data leakage.

**Metrics.** Primary metrics were Mean Absolute Error (MAE, seconds) and Mean Absolute Percentage Error (MAPE, %). Training time (seconds) was tracked to assess computational efficiency, since this would be an important factor for prospective re-training in real-time scenarios, where resources would be limited.

### 4.2.4 Baseline Models

Four model families were benchmarked using the initial feature set (origin/destination coordinates, start time, distance): linear regression and three gradient boosting libraries, namely LightGBM, XGBoost, and CatBoost. This decision was made because of previous work showing that gradient boosting methods are well-suited for ETA prediction tasks [62] and tabular data in general.

As seen in Table 4.5, gradient boosting methods outperformed the linear baseline, as was expected, confirming the value of non-linear interactions for ETA.

Table 4.5: Baseline results (5-fold CV on train).

| Model | MAE (s) | MAPE (%) | Train time (s) |
|---|---|---|---|
| Linear Regression | 31.46 | 16.11 | 0.003 |
| LightGBM | 27.78 | 13.39 | 0.40 |
| XGBoost | 27.91 | 13.59 | 0.57 |
| CatBoost | 28.23 | 13.57 | 0.58 |

All subsequent experiments focused on the three gradient boosting models due to their superior performance.

### 4.2.5 Transformation Experiments

Prior to feature engineering, standard transformations were evaluated to test whether distributional normalization could improve accuracy or stability [67]. These transformations were:

- **Feature log-transform:** applying $\log(1 + x)$ to the distance feature.
- **Standard scaling:** applying standardization (zero mean, unit variance) to all features.
- **Target log transform:** applying $\log(1 + y)$ to the duration target variable.
- **Target Box-Cox transform:** applying Box-Cox transformation to the duration target variable [68].
- **Target quantile transform:** applying quantile normal transformation to the duration target variable [69].

All transformation experiments used the original 6 features (baseline configuration) with 5-fold stratified cross-validation. The results of these experiments are summarized in Table 4.6.

Table 4.6: Transformation experiments (5-fold CV on train).

| Transformation | Average MAE (s) | Improvement | Best Model |
|---|---|---|---|
| None | 27.97 | — | LightGBM (27.78s) |
| Log (distance) | 27.97 | 0.00s (0.00%) | LightGBM (27.78s) |
| Standardization | 27.98 | 0.01s (0.04%) | LightGBM (27.80s) |
| Log (duration) | 27.92 | 0.05s (0.18%) | LightGBM (27.73s) |
| Box-Cox (duration) | 27.93 | 0.04s (0.14%) | LightGBM (27.72s) |
| Quantile Normal (duration) | 27.90 | 0.07s (0.25%) | LightGBM (27.71s) |

None of the transformations resulted in a significant improvement in model performance. This is expected for tree-based gradient boosting models, which are inherently invariant to monotonic transformations and feature scaling. Therefore, since the added complexity did not justify the performance gains, no transformations were applied in subsequent experiments.

### 4.2.6 Feature Engineering

Feature engineering was conducted systematically to capture spatial separations, temporal context, and route length in model-friendly forms [70, 71]. Initially, some feature groups were tested independently to assess their individual contribution to model performance. These feature groups were:

1. **Original Features (6 features):** Base features extracted directly from the simulation FCD data. These features serve as the baseline for all experiments.
2. **Temporal Features (5 features):** Time-based patterns extracted from trip start time. These features encode temporal patterns.
3. **Spatial Features (17 features):** Geometric and geographic relationships derived from coordinates and distances. These features capture urban structure and geometric relationships that influence travel time.
4. **Fourier Features (16 features):** Sinusoidal positional encoding of spatial coordinates at 2 frequency scales. These features capture spatial patterns in the coordinate space.
5. **Cell Features (4 features):** Spatial discretization into fixed-size grid cells (100 meters). These features capture spatial patterns in the grid space.

6. **Cluster Features (2 features):** K-Means clustering (K=20) on coordinates to discretize spatial regions. These features capture spatial patterns in the cluster space.
7. **PCA Features (4 features):** Principal Component Analysis for dimensionality reduction (2 components). These features capture dominant spatial variance in a reduced dimensional space.

Each feature group was evaluated through dedicated experiments to assess its contribution to prediction performance, and then once all together to assess their collective impact. The results of these experiments are summarized in Table 4.7.

Table 4.7: Feature engineering experiments results.

| Feature Configuration | Average MAE (s) | Improvement | Best Model |
|---|---|---|---|
| Original only | 27.97 | — | LightGBM (27.78s) |
| Original + Temporal | 27.84 | 0.13s (0.5%) | XGBoost (27.57s) |
| Original + Spatial | 27.77 | 0.20s (0.7%) | XGBoost (27.42s) |
| Original + Fourier | 27.69 | 0.28s (1.0%) | XGBoost (27.48s) |
| Original + Cell | 27.76 | 0.21s (0.8%) | XGBoost (27.39s) |
| Original + Cluster | 27.87 | 0.10s (0.4%) | XGBoost (27.51s) |
| Original + PCA | 27.73 | 0.24s (0.9%) | XGBoost (27.38s) |
| All features | 27.63 | 0.34s (1.2%) | XGBoost (27.33s) |

All feature groups provided improvement over the original features, ranging from 0.10s to 0.34s MAE reduction. The combined features performed best, achieving the lowest MAE of 27.63 s, but the improvement from combining all features is less than the sum of individual improvements, suggesting some overlap in captured information.

After evaluating the combined feature set (52 features total), systematic feature selection was performed to identify the most valuable features while reducing dimensionality, training time, and model complexity. The selection methods were:

- **Gain-Based Importance:** Used LightGBM's built-in split gain importance to measure feature contribution to loss reduction across all tree splits.
- **Permutation Importance:** Model-agnostic approach measuring feature impact by computing increase in MAE when each feature is randomly shuffled [72].
- **SHAP Values:** Shapley Additive Explanations from game theory to quantify each feature's contribution to individual predictions [38].
- **Correlation Analysis:** Computed Pearson correlation between all feature pairs to identify highly correlated features ($|r| > 0.95$) [73].

The three importance-based methods (gain, permutation, SHAP) were aggregated into a combined ranking with normalization, aggregation and a weighted combination of normalized importance and rank as a final score. The top 20 features by combined score are listed in Table 4.8.

Distance based features dominate the top 10 poisitions, and all 4 pca features appear in the top 20, confirming their value for dimensionality reduction. Cluster features rank moderately high (12th), capturing spatial zone patterns. Original coordinate features rank lower (20+), but are retained due to PCA high correlations.

Table 4.8: Top 20 features by combined importance score.

| Rank | Feature | Score | Category |
|---|---|---|---|
| 1 | distance | 0.938 | Original |
| 2 | is_short_distance | 0.567 | Spatial |
| 3 | euclidean_distance | 0.520 | Spatial |
| 4 | x_center | 0.497 | Spatial |
| 5 | is_long_distance | 0.476 | Spatial |
| 6 | detour_length | 0.473 | Spatial |
| 7 | time_start | 0.428 | Original |
| 8 | y_center | 0.424 | Spatial |
| 9 | destination_distance_from_city_center | 0.412 | Spatial |
| 10 | source_distance_from_city_center | 0.396 | Spatial |
| 11 | is_medium_distance | 0.395 | Spatial |
| 12 | source_cluster | 0.384 | Cluster |
| 13 | is_noon | 0.378 | Temporal |
| 14 | trip_centrality | 0.368 | Spatial |
| 15 | destination_pca_1 | 0.367 | PCA |
| 16 | source_pca_2 | 0.359 | PCA |
| 17 | destination_y_sin_1 | 0.358 | Fourier |
| 18 | source_pca_1 | 0.335 | PCA |
| 19 | destination_pca_2 | 0.308 | PCA |
| 20 | destination_y | 0.289 | Original |

Based on the combined ranking and the correlation analysis, we decided to eliminated all 5 temporal features, all 16 fourier features, all 4 cell features and 7 of the 17 spatial features. The retained features are all 6 original features, all 2 cluster features, all 4 pca features and 10 of the 17 spatial features:

The feature selection was validated through a dedicated experiment comparing against the full feature set:

Table 4.9: Feature selection validation results.

| Metric | All features | Selected features | Change |
|---|---|---|---|
| **Features** | 52 | 22 | **-58%** |
| **Average MAE (s)** | 27.63 | 27.66 | **+0.03s** |
| **Average MAPE (%)** | 13.26 | 13.27 | **+0.01%** |
| **Average Training Time (s)** | 0.81 | 0.49 | **-40%** |
| **Best Model** | XGBoost (27.33s) | XGBoost (27.33s) | **Identical** |

With only 0.03s MAE increase, 0.01% MAPE increase, and a 40% reduction in training time, the feature selection was successful in retaining the most informative features while eliminating redundancy and reducing computational cost.

### 4.2.7 Hyperparameter Tuning

Systematic hyperparameter optimization was performed for all three gradient boosting models using Optuna [39] with the Tree-structured Parzen Estimator (TPE) sampler [74]. The optimization objective was to minimize Mean Absolute Error (MAE) across 5-fold stratified cross-validation. A two-phase approach was employed to balance exploration and exploitation. Phase 1 was a broad search with 100 trials per model, while Phase 2 was a more focused search on the best performing regions with 50 trials per model. The optimization process totaled 450 trials (150 per model) without early stopping, leveraging the fast training times enabled by the compact dataset.

The results of the hyperparameter tuning are summarized in Table 4.10.

Table 4.10: Hyperparameter tuning results.

| **Model** | **Phase** | **Best MAE (s)** | **Best MAPE (%)** | **Training Time (s)** |
|---|---|---|---|---|
| CatBoost | 1 | 26.72 | 12.80 | 11.56 |
| CatBoost | 2 | 26.70 | 12.81 | 14.70 |
| LightGBM | 1 | 26.58 | 12.78 | 2.60 |
| LightGBM | 2 | 26.51 | 12.77 | 2.95 |
| XGBoost | 1 | 26.69 | 12.82 | 7.47 |
| XGBoost | 2 | 26.63 | 12.83 | 9.44 |

LightGBM achieved the lowest MAE of 26.51 s and MAPE of 12.77% on cross-validation with a training time of 2.95 s, and was therefore selected for downstream evaluation on the held-out scenarios. Its training efficiency was superior to both XGBoost and CatBoost, being 3x and 5x faster respectively, and its performance was the best and most consistent across both phases of the tuning process.

### 4.2.8 Final Model

The selected model is LightGBM with the following hyperparameters seen in Table 4.11, chosen via cross-validated tuning for minimum MAE:

Table 4.11: Final LightGBM hyperparameters.

| **Parameter** | **Value** |
|---|---|
| `max_depth` | 14 |
| `n_estimators` | 1100 |
| `num_leaves` | 104 |
| `learning_rate` | 0.043 |
| `subsample` | 0.958 |
| `colsample_bytree` | 0.692 |
| `min_child_samples` | 38 |
| `min_split_gain` | $1.2 \times 10^{-5}$ |
| `reg_alpha` | $6.7 \times 10^{-6}$ |
| `reg_lambda` | $3.5 \times 10^{-5}$ |

# Chapter 5

# Platform Architecture and Implementation

## 5.1 High-Level Overview

The platform embodies a microservice design that orchestrates a time-lapse traffic simulation, performs multi-task prediction, monitors error streams for concept drift via consensus, and adapts by retraining and hot-swapping models, while exposing a web dashboard for both administration and user-facing predictions.

At a high level, the platform comprises six independent services (Backend, three Predictors for ETA/Fuel/Stops, Drift, Frontend, and Summarizer) communicating over HTTP/REST on a private Docker network. The Backend acts as the source of truth for simulation state and orchestrates a 20-hour scenario compressed to about 4-5 minutes (300x speedup factor), coordinating batch predictions, drift detection, and adaptation, when needed. The Frontend offers an admin dashboard (controls, metrics, notifications, post-simulation report) and a user interface for map-based route selection and on-demand multi-task predictions.

## 5.2 System Architecture

The platform follows a microservices architecture with strict separation of concerns [40]. Each service exposes a small, well-typed FastAPI with Pydantic schemas, is independently deployable, and can be evolved or scaled without impacting others. A minimal service inventory and ports are:

- **Backend** (Port 8000): orchestration, timelapse control, state stores, simulation snapshot API.
- **Predictor-ETA/Fuel/Stops** (Port 8001/8002/8003): inference, feature calibration, batch or single predictions, background retraining.
- **Drift** (Port 8004): per-task workers, four detectors, consensus and online calibration.
- **Summarizer** (Port 8005): post-simulation AI report generation from metrics and notifications.

- **Frontend** (Port 8080): admin dashboard and user interface, interacts only with the Backend API.

Figure 5.1 showcases the architecture as a diagram:

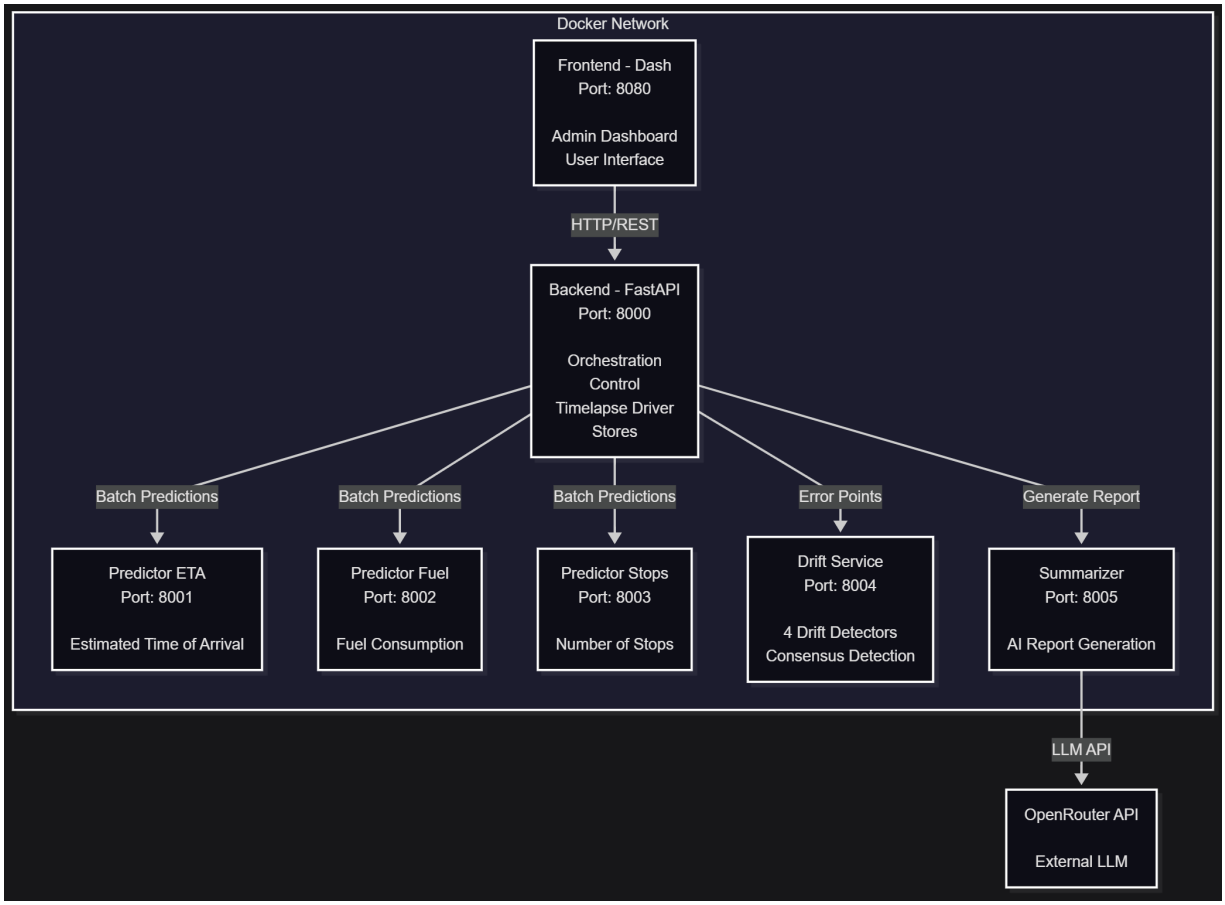


Figure 5.1: Platform Architecture.

**Inter-service links.** Backend dispatches batch windows to predictors and error streams to Drift and it also triggers the Summarizer post-simulation. Frontend polls Backend for snapshots, metrics, notifications, and report content.

## 5.3 Service Responsibilities

**Backend.** The Backend owns orchestration and state. *TimelapseDriver* advances the clock in one-second real-time steps at 300× simulated speed (5 simulated minutes per tick), parallelizes windowed prediction requests, triggers drift checks, and manages the adaptation lifecycle. State is maintained in in-memory circular buffers (*MetricsStore*, *NotificationStore*) and a *ReportStore*. Startup includes dynamic discovery of available predictors via parallel health checks to support partial deployments. The Backend remains task-agnostic by design: it requests predictions for time windows and stores absolute errors without needing task-specific feature knowledge.

**Predictors (ETA, Fuel, Stops).** Each predictor shares a common codebase parameterized by task identity. Core components include: *ModelManager* (versioned registry under the shared simulation appdata directory), *DataLoader* (fast Parquet windowing by timestamp), *Predictor* (batch/single inference and error computation), *FeatureCalibrator* (task-specific engineered features for single predictions), a *RetrainService* running in a dedicated process with retries and dynamic model sizing according to the size of the model and the retraining data, and a *SumoService* as an interface with to the SUMO network for route preview and feature extraction.

**Drift Detection.** The Drift service runs one worker process per ML task with request and response queues,and implements four complementary detectors (ADWIN [23], Page-Hinkley [24], KSWIN [25], SPC [26]). Detectors operate over smoothed error streams with constant time computation, and a grace period before activation. Drift is declared when at least 3/4 detectors agree (configurable). A calibration phase collects baseline errors, tunes detector hyperparameters against false positives, then transitions to online monitoring. After adaptation, the detectors need to be recalibrated.

**Frontend.** The Frontend (Dash + Bootstrap + Leaflet) exposes two tabs. The Admin Dashboard provides simulation controls (Start/Pause/Resume/Reset), MAE charts with drift window highlights, per-task status, a chronological notification feed, and a report viewer/downloader. The User Interface supports map-based source/destination selection, SUMO-based route preview, and synchronous ETA/Fuel/Stops prediction. Frontend state is maintained client-side using stores, interval polling of the Backend API, and properly set up callbacks to ensure that the UI is updated in real time.

**Summarizer.** On completion, the Backend packages a timeline of notifications and time-series metrics and asks the Summarizer to generate a markdown report (and PDF). The Summarizer formats and structures the data and then uses the OpenRouter API to access the LLM models. The service is configured to use free models, so no charges are incurred. There are robust retries and timeouts in place to handle transient failures.

## 5.4 Technology Stack

The following core technologies are used in the various platform services:

- **Infrastructure:** Docker, Docker Compose, uv dependency manager [75], Python 3.12
- **APIs:** FastAPI with Uvicorn, uvloop, httptools and ORJSONResponse, and async httpx clients [76] for inter-service calls
- **Frontend:** Dash with Plotly components, Bootstrap theming, Leaflet for interactive maps, xhtml2pdf for report pdf generation
- **ML:** LightGBM, XGBoost, NumPy, pandas, and River for drift detectors
- **Data:** Pandas [77], NumPy [78] and PyArrow [79]/Parquet for fast filtered reads by timestamp
- **LLMs:** OpenAI client for the LLM API (OpenRouter API)
- **Validation:** Pydantic models shared across services for type safety and OpenAPI docs

## 5.5 Simulation Control Flow

### 5.5.1 Timelapse Tick

Each tick advances the simulated clock by five minutes, then:

1. Backend sends concurrent batch prediction requests to all available predictors.
2. Predictors load the window via *DataLoader*, apply precomputed features, run inference, and return absolute errors and aggregates (e.g., MAE).
3. Backend forwards error points to the Drift service, which updates its per-task detectors and returns the state of each task (Calibrating/Stable/Drifted/Retraining).
4. Backend updates the stores, performs state transitions, and if drifted, starts collecting a fixed post-drift window for retraining.
5. When the window completes, Backend submits a background retraining job; on completion, the predictor hot-swaps to a new version and the Drift service recalibrates.
6. Frontend polling refreshes snapshot/metrics/notifications, keeping the UI responsive in real time.
7. Summarizer generates a report from the notifications and metrics if the simulation is completed.

This design maintains sub-second processing for each batch window and preserves UI interactivity throughout the simulation.

Figure 5.2 shows the various drift states of a model, as the simulation progresses, in a diagram:

### 5.5.2 User Prediction Flow

For on-demand queries, the user first picks endpoints on the map available on the User Interface tab. Then, a predictor returns a route preview (via SUMO network access) and the Backend dispatches parallel single-prediction requests to the predictors. Each predictor uses its task-specific *FeatureCalibrator* to compute identical features to training, then serves the predictions values to the user.

## 5.6 Deployment Architecture

### 5.6.1 Compose-Based Orchestration

The system ships as a multi-container deployment with Docker Compose. Docker Compose ensures startup ordering via health checks, isolates the network, and mounts shared volumes under `appdata/` (common SUMO data, task datasets, model registry, feature artifacts, and per-service logs). In production, only the Frontend requires external exposure, internal ports can remain bridged on the Docker network. Environment variables identify the running service, set the app data root, and toggle environment mode.

### 5.6.2 Images, Footprint, and Resources

All services start from `python:3.12-slim`, install only their optional dependency group (via *uv*), and include *curl* for health checks. Predictor images are larger due to gradient-boosting stacks

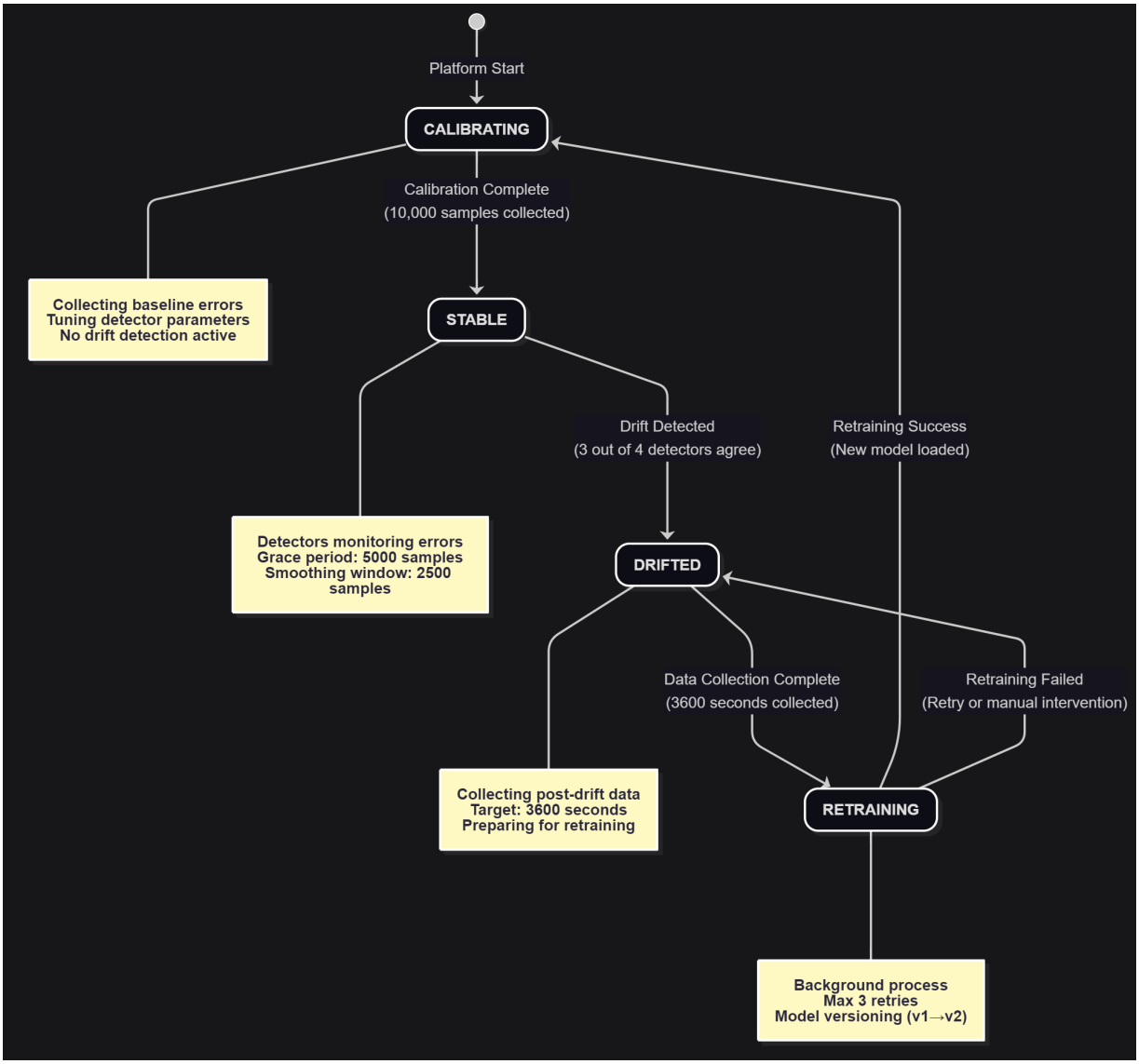


Figure 5.2: Drift States of a Model.

(around 2 GB), while other services remain comparatively light (around 500 MB). Dependencies are managed with a lockfile-based workflow. This includes dependency groups in `pyproject.toml` that install only what each service needs (e.g., `thesis[drift]` vs. `thesis[predictor]`), keeping images lean and builds fast.

### 5.6.3 Configuration and Dependencies

Configuration is centralized in a single YAML file, covering logging and rotation, event loop/server choices, timelapse parameters, drift detection parameters and consensus threshold, retraining policies and resource allocation settings, and summarizer options, such as the system prompt, and retry settings.

## 5.7 Performance Goals and Realization

A central goal was to simulate 72,000 s (20 h) of traffic in a few minutes of wall-clock time while preserving the fidelity of per-window evaluation and monitoring. To that end, the implementation combines asynchronous request orchestration with lightweight data paths that avoid blocking operations in the hot loop. Batch predictions read precomputed feature matrices from Parquet, which eliminates per-tick feature construction and reduces I/O overhead. Metrics and notifications are maintained in memory during the run, thereby removing database round-trips when the system is most active. Model retraining is delegated to background worker processes so that inference remains responsive even when adaptation is underway. Finally, per-task drift detectors run in separate processes to avoid blocking the main thread and interfering with the communication with the backend.

In practice, the platform achieves the 300x multiplier, processes 110,000+ trips across tasks, runs a total of 12 drift detectors concurrently, and sustains an interactive dashboard throughout, without any performance degradation.

## 5.8 Logging, Error Handling, and Observability

All services initialize a common logger with sensible defaults and log rotation, so that operational traces remain useful across long runs. Critical paths are guarded with graceful degradation on downstream failures, so a localized issue does not cascade into systemic failure. Looking ahead to production deployment, the same hooks can be routed to centralized log aggregation and tracing, enabling correlation of events across services. Looking ahead to production deployment, the same hooks can be routed to centralized log aggregation and tracing, enabling correlation of events across services. Health checks already standardize liveness and readiness; extending them with lightweight self-diagnostics (e.g., queue backlog sizes, last successful tick) would make on-call diagnostics faster without altering the public API. In addition to the above, comprehensive automated tests (unit, integration, performance, end-to-end) would further harden reliability and provide a safety net for the platform.

## 5.9 Concurrency and Isolation

The architecture uses async I/O for parallel calls per tick (`httpx.AsyncClient`, `asyncio.gather`), while process-based parallelism isolates heavy/long-running work. CPU-bound activities, most notably retraining and detector updates, run in separate processes so that Python's GIL does not throttle throughput and so that faults remain contained. Drift workers run per task in separate processes, while retraining uses a dedicated `ProcessPoolExecutor` per predictor, so training never blocks inference. This division of labor yields robust concurrency on multi-core hosts and prevents cross-task interference, while preserving responsiveness under load.

## 5.10 Extensibility and Design Principles

### 5.10.1 Backend Abstraction and Task Agnosticism

The Backend purposefully avoids task-specific feature and model logic. It orchestrates time windows, aggregates errors, and manages adaptation state without knowing how a predictor computes its outputs. Adding a new task involves preparing its model and calibrator artifacts, registering

the task in configuration, and deploying an additional predictor container. The Backend discovers it at startup and incorporates it into orchestration automatically. This separation of concerns reduces coupling and simplifies future extensions.

### 5.10.2 Feature Engineering Pattern

Two complementary paths support inference. Batch predictions rely on precomputed features, enabling sustained high throughput during the simulation. Conversely, on-demand user queries construct features on the fly through the *FeatureCalibrator* artifacts, ensuring consistency with the training pipeline while accommodating arbitrary origins and destinations. This pattern balances efficiency and flexibility. If future scenarios require true online features, the same interfaces can be preserved while swapping the feature source.

### 5.10.3 Data Access Pattern

The platform encapsulates batch data access behind a simple *DataLoader* interface that accepts a start and end timestamp and returns a time-windowed frame for inference and evaluation. In the current implementation, windows are served from local Parquet files with fast, vectorized timestamp filtering. However, the same interface can back onto alternative sources, like a relational or time-series database, a message bus (Kafka/RabbitMQ), cloud object storage (S3/GCS), or a live stream, without requiring changes to callers. This design keeps the Backend and predictors free of storage concerns and allows deployments to evolve from file-based artifacts during development to production-grade data services by swapping only the loader implementation while preserving contracts.

### 5.10.4 Adaptation Policy and Consensus

Drift detection uses multiple detectors with a configurable consensus threshold (default 3/4) to reduce false positives. The service begins with a calibration phase that characterizes baseline error behavior. Only after calibration does it enter steady monitoring. A short grace period after startup further suppresses transient alerts. When drift is declared, the system collects a fixed post-drift window that is sufficient for stable retraining, then performs a model hot-swap and returns to recalibration. Externalizing these parameters in configuration allows tuning of sensitivity and adaptation latency for different operating contexts without code changes.

## 5.11 Architectural Strengths and Trade-offs

**Strengths.** The platform benefits from clear module boundaries, stateless HTTP interfaces, and a single orchestrator that maintains system-wide state. Asynchronous requests keep ticks short, process isolation prevents heavy tasks from monopolizing resources, and a simple, versioned model registry makes hot-swaps predictable and reversible. Together, these choices create a system that is easy to reason about and robust under typical faults.

**Trade-offs.** Several pragmatic trade-offs accompany the design. In-memory stores provide speed at the expense of long-term persistence and historical analytics. Precomputed features accelerate batch evaluation but are less representative of a fully online setup. The fixed post-drift collection window introduces some adaptation delay in exchange for training stability. Each of these decisions was calibrated for the thesis setting and all can be replaced with production-grade

counterparts, such as durable storage, a streaming feature store, or incremental learners, without changing the external service contracts.

## 5.12 API Endpoints Reference

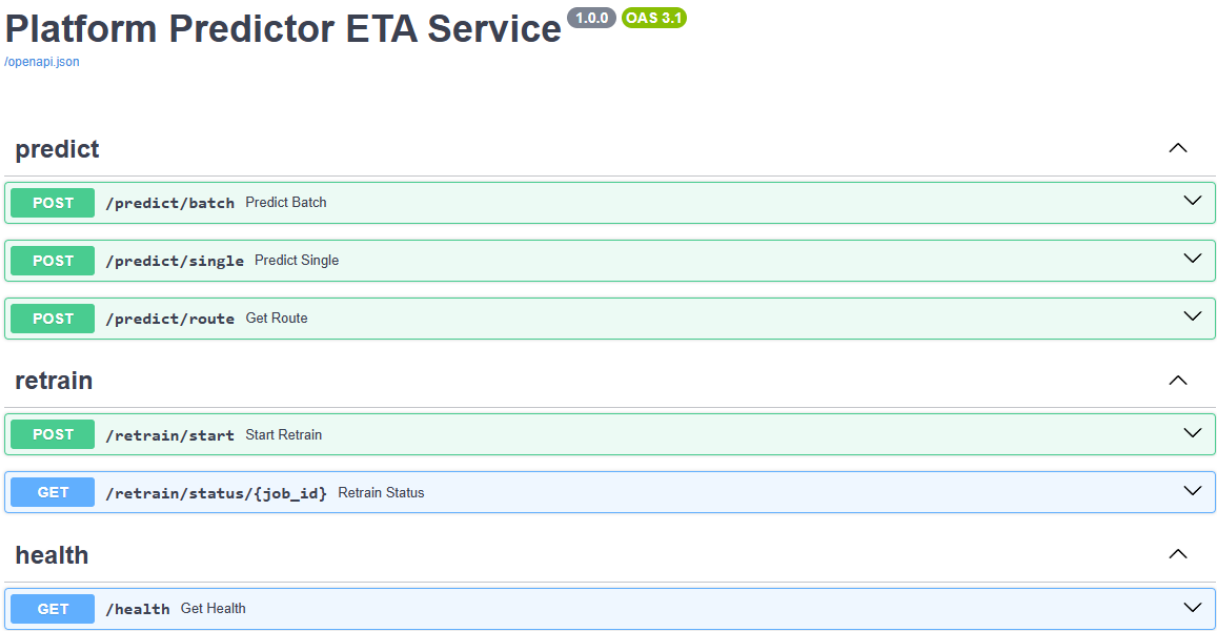


Figure 5.3: Backend API Endpoints Reference, Swagger UI.

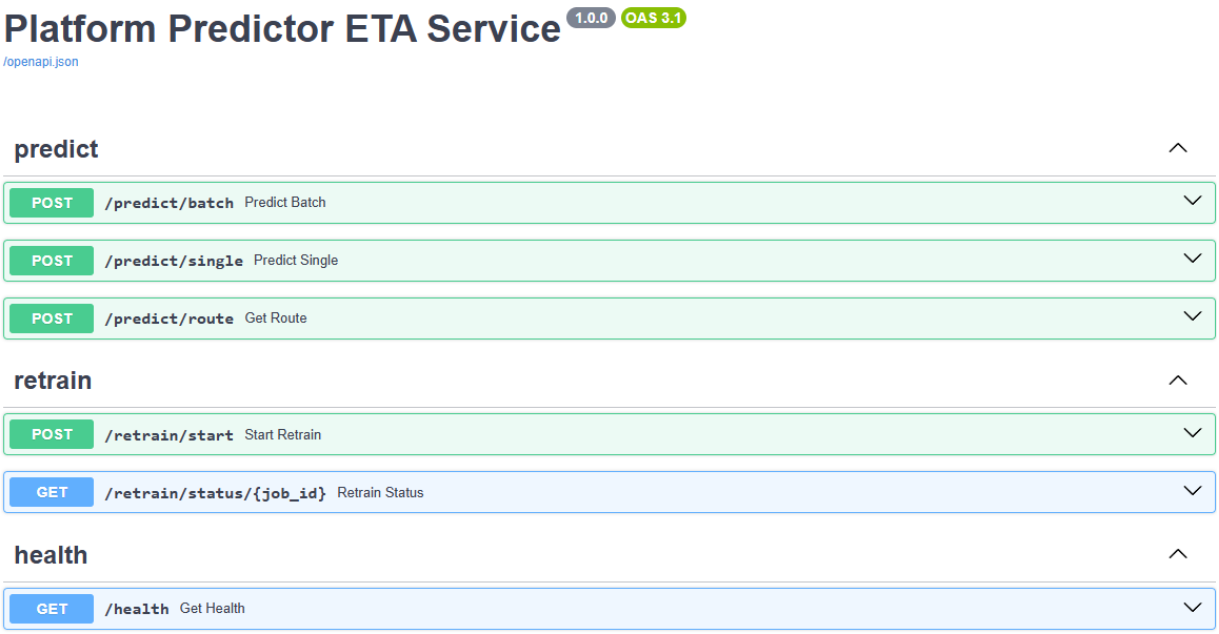


Figure 5.4: Predictor API Endpoints Reference, Swagger UI.


Figure 5.5: Drift API Endpoints Reference, Swagger UI.

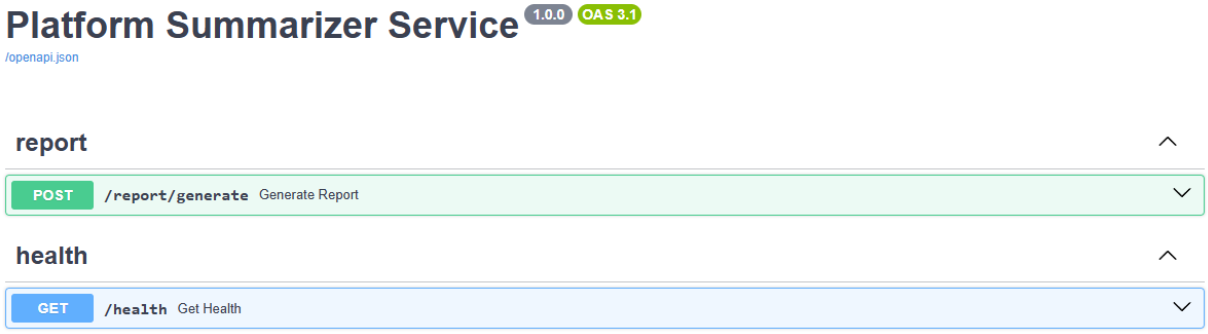


Figure 5.6: Summarizer API Endpoints Reference, Swagger UI.

**Frontend Service** The frontend provides a web interface with Admin Dashboard and User Interface tabs. It communicates with the backend exclusively through the backend's REST API endpoints. It does not expose any REST API endpoints itself. The interface is accessible at `http://localhost:8080`.

## 5.13 Summary

The platform operationalizes continuous ML for urban mobility through a principled microservice architecture. It demonstrates that a time-lapse, consensus-driven monitoring loop with targeted retraining can maintain prediction quality under controlled distribution shifts, all while sustaining real-time interactivity and a practical developer experience.

# Chapter 6

# Results

## 6.1 Overview

This chapter presents (i) baseline performance of the ETA model across normal and rain scenarios, (ii) adaptation via incremental retraining after detected drift, and (iii) a head-to-head comparison of the initial vs. retrained models on the same post-swap evaluation window. We conclude with a concise demonstration of the operational platform, meaning the dashboards, notifications, and reports.

## 6.2 Notation and Setup

Let $t \in [0, 72{,}000]$ denote simulation time in seconds. We define the following time windows:

$$\begin{aligned} W_{\text{test}} &= [0, 36{,}000), \\ t_{\text{rain}} &= 36{,}000, \\ W_{\text{rain}} &= [36{,}000, 72{,}000], \\ t_{\text{drift}} &= 38{,}400, \\ W_{\text{retrain}} &= [38{,}400, 42{,}000], \\ t_{\text{swap}} &= 43{,}200, \\ W_{\text{eval}} &= [43{,}200, 72{,}000]. \end{aligned}$$

## 6.3 Models and Metrics

**Models.** We compare two models:

- **Initial model** $M_0$: trained on 10 hours of normal data (initial training set size 53,229).
- **Retrained model** $M_1$: incrementally fine-tuned on $W_{\text{retrain}}$ using 1 hour of samples or 6,755 samples ($\approx 12.69\%$ of the initial training volume) and deployed at $t_{\text{swap}} = 43{,}200$.

**Metrics.** We use Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) to evaluate the performance of the models:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{y}_i\right|, \qquad \text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \%.$$

For relative improvements from model $A$ to $B$ we use

$$\Delta_{\%}(A \to B) = 100 \times \frac{A - B}{A},$$

and we denote absolute differences in percentage points as "pp".

## 6.4 Baseline Model Performance

Table 6.1 summarizes the performance of the initial model $M_0$ on the first day (normal conditions) and on the second day (rain conditions), before any adaptation.

Table 6.1: Baseline results for $M_0$ (no adaptation), by scenario.

| Scenario / Window | MAE | MAPE |
|---|---|---|
| First day $W_{\text{test}}$ | **30.36** | **13.20%** |
| Second day $W_{\text{rain}}$ | **56.93** | **19.02%** |

Moving from normal to rain nearly doubles MAE (from 30.36 to 56.93) and raises MAPE from 13.20% to 19.02%, indicating a substantial scenario shift, a concept drift.

The evolution of the baseline model errors over time can be seen in the following Figure 6.1:

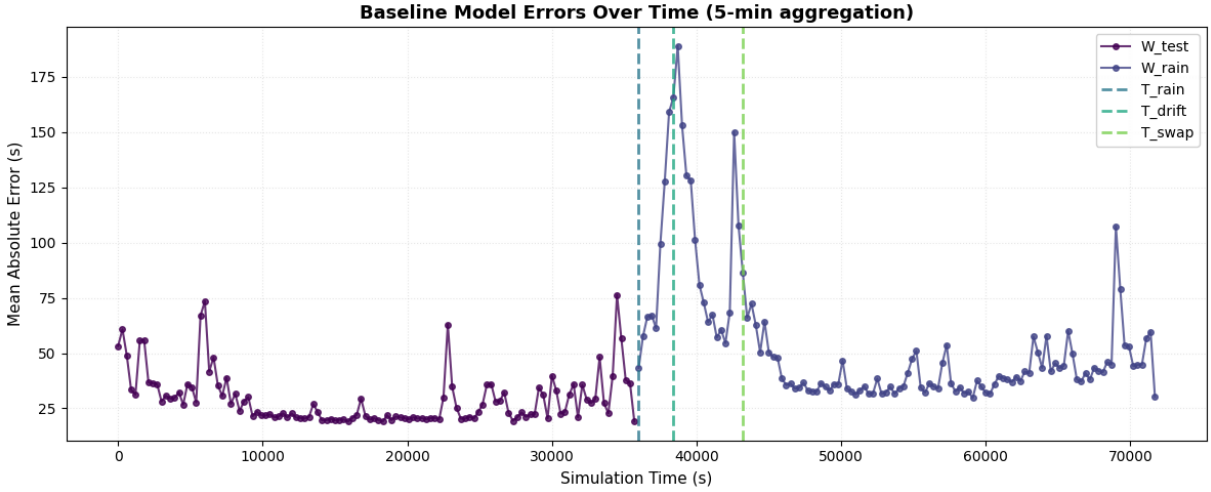


Figure 6.1: Baseline Model Errors Over Time.

## 6.5 Retrained Model: Training Window and Rationale

A drift event is detected at $t_{\text{drift}} = 38{,}400$ by the platform's drift detectors, running for this specific model. To adapt, the platform fine-tunes $M_0$ on the first hour of post-drift rain data, i.e., $W_{\text{retrain}} = [38{,}400, 42{,}000]$ (6 755 samples, $\approx 12.69\%$ of the initial training volume). The updated model $M_1$ is then hot-swapped at $t_{\text{swap}} = 43{,}200$.

This choice targets the specific post-drift distribution and with only an hour of data and a quick retraining time, it allows for a seamless adaptation, producing an adapted model while the

simulation continues. We evaluate the effect of this adaptation on a clean hold-out: the remainder of the rain scenario after the recorded swap.

## 6.6 Post-Swap Evaluation on an Identical Window

To ensure a fair, apples-to-apples comparison, we evaluate both the non-adapted baseline ($M_0$) and the retrained model ($M_1$) on the *same* post-swap window $W_{\text{eval}} = [43{,}200, 72{,}000]$.

Table 6.2: Head-to-head results on the post-swap window $W_{\text{eval}} = [43{,}200, 72{,}000]$.

| Model (Window) | MAE | MAPE |
|---|---|---|
| $M_0$ (initial; $W_{\text{eval}}$) | **43.27** | **16.89%** |
| $M_1$ (retrained; $W_{\text{eval}}$) | **32.27** | **14.55%** |
| Absolute difference | **11.00** s MAE | **2.34** pp MAPE |
| Relative change $\Delta_{\%}$ | **25.42%** ↓ | **13.85%** ↓ |

The retrained model reduces MAE by 11.00 s (25.42%) and MAPE by 2.34 pp (13.85%) versus the baseline on the post-swap window, evidencing effective adaptation to rain.

The post-swap comparison of the baseline and retrained models can be seen in the following Figure 6.2:

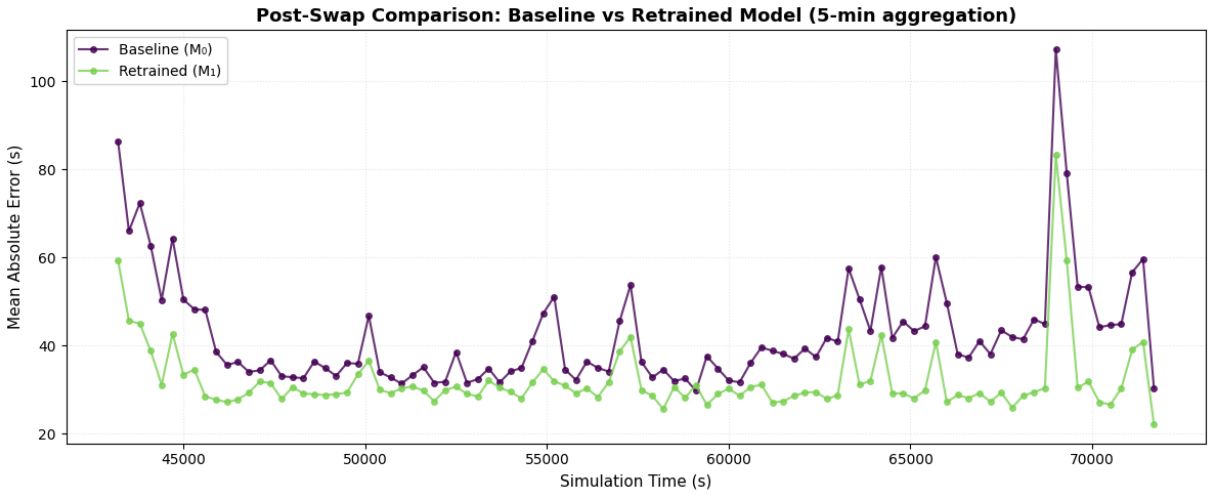


Figure 6.2: Post-Swap Comparison: Baseline vs Retrained Model.

## 6.7 Takeaways

**Baseline drift impact.** From normal to rain, MAE rises from 30.36 to 56.93 and MAPE from 13.20% to 19.02%, confirming a substantial distribution shift.

**Adaptation effectiveness.** On the identical post-swap window $W_{\text{eval}}$, $M_1$ improves over $M_0$ by **11.00 s MAE** (25.42%) and **2.34 pp MAPE** (13.85%).

**Operational closure of the loop.** The platform detects drift, fine-tunes on $W_{\text{retrain}}$, and hot-swaps at $t$=43,200, showcasing a successful and seamless adaptation process.

## 6.8 Platform Demonstration

Figures 6.3–6.12 illustrate the end-to-end run from an operator's perspective. Each step shows either the *Admin Dashboard* or the *User Interface* to connect system state with user-facing predictions.

**Narrative highlights.** *Start* → *Stable* establishes the nominal baseline. During *Drifted*, rain conditions are active, all models experience performance degradation and all tasks are flagged as drifted. In *Retraining*, only ETA has been retrained and swapped, so the User Interface shows a higher ETA prediction for the same route while fuel and stops remain at their pre-adaptation values. Finally, in *Retrained*, all tasks are adapted and calibrated, and the User Interface reflects consistent post-recovery predictions. The *Report* view closes the loop with the AI generated summary report of the simulation, after it has completed.

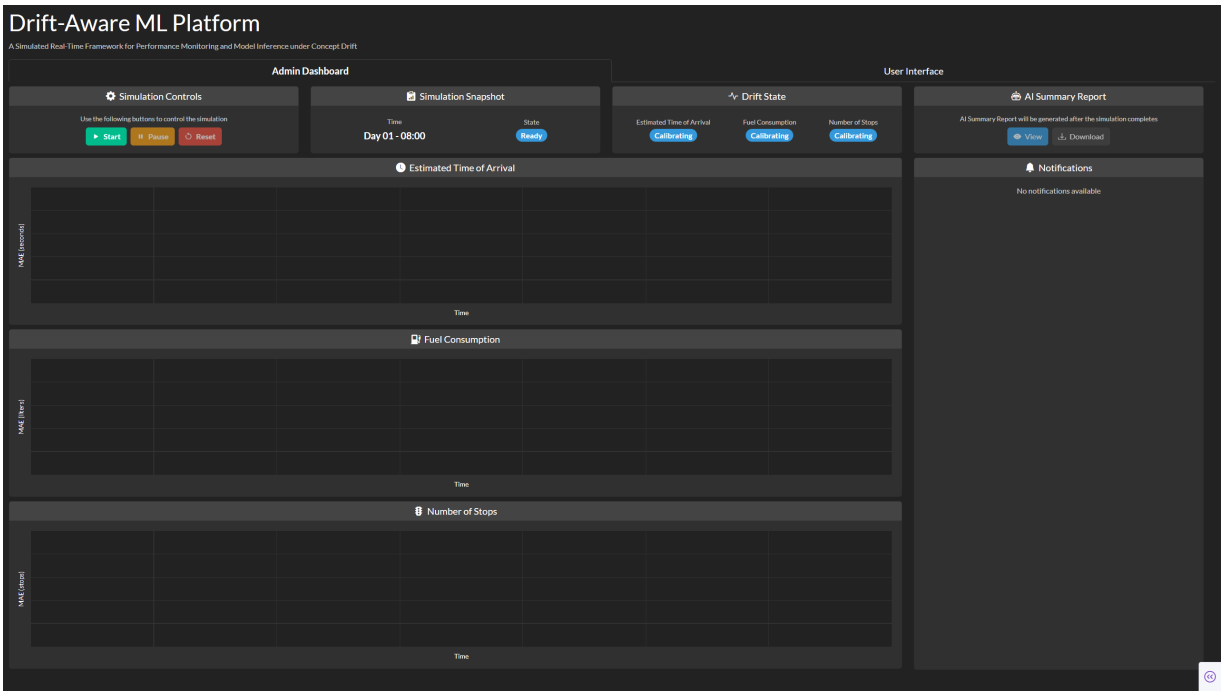


Figure 6.3: Admin dashboard — *Start*. Detectors calibrating, graphs empty, simulation not yet running.

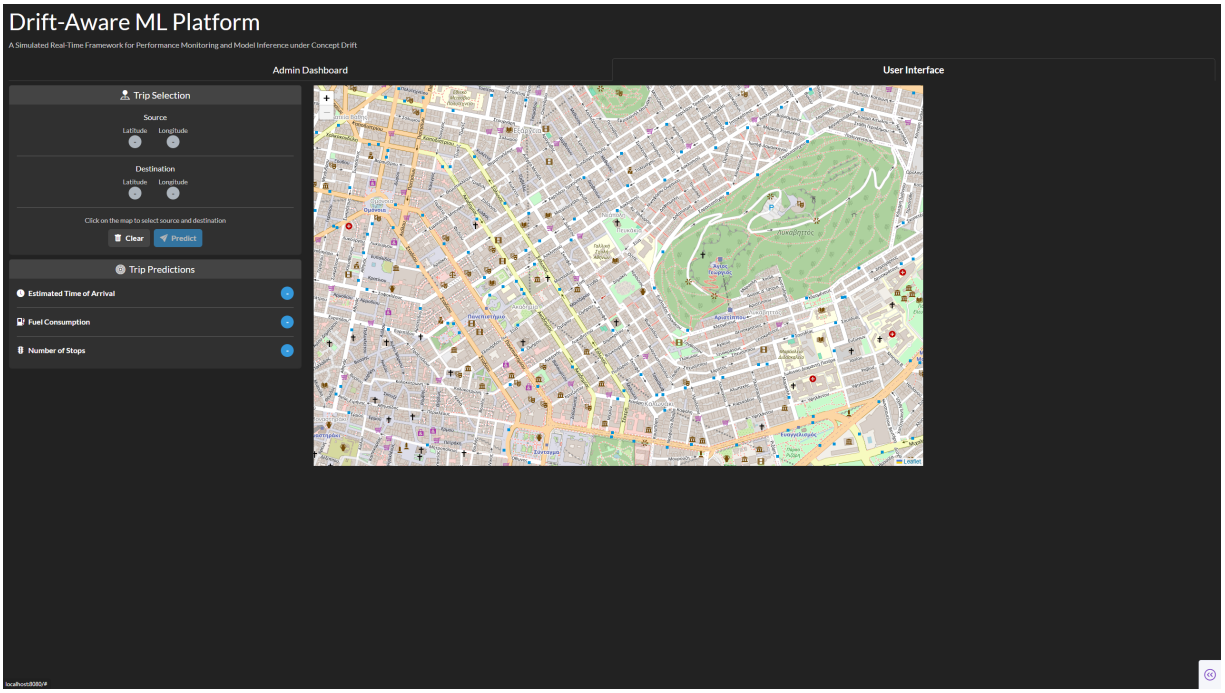


Figure 6.4: User interface — *Start*. No Source and Destination selected, prediction panels empty.

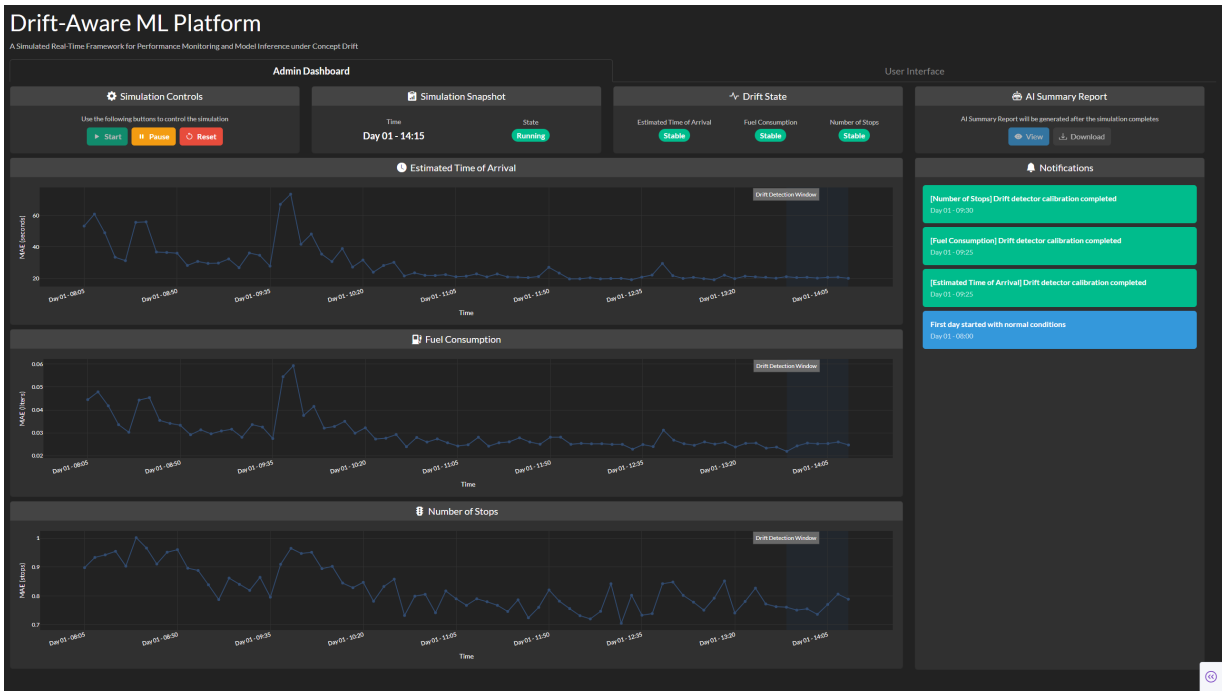


Figure 6.5: Admin dashboard — *Stable*. Models stable under normal conditions, detectors calibrated.

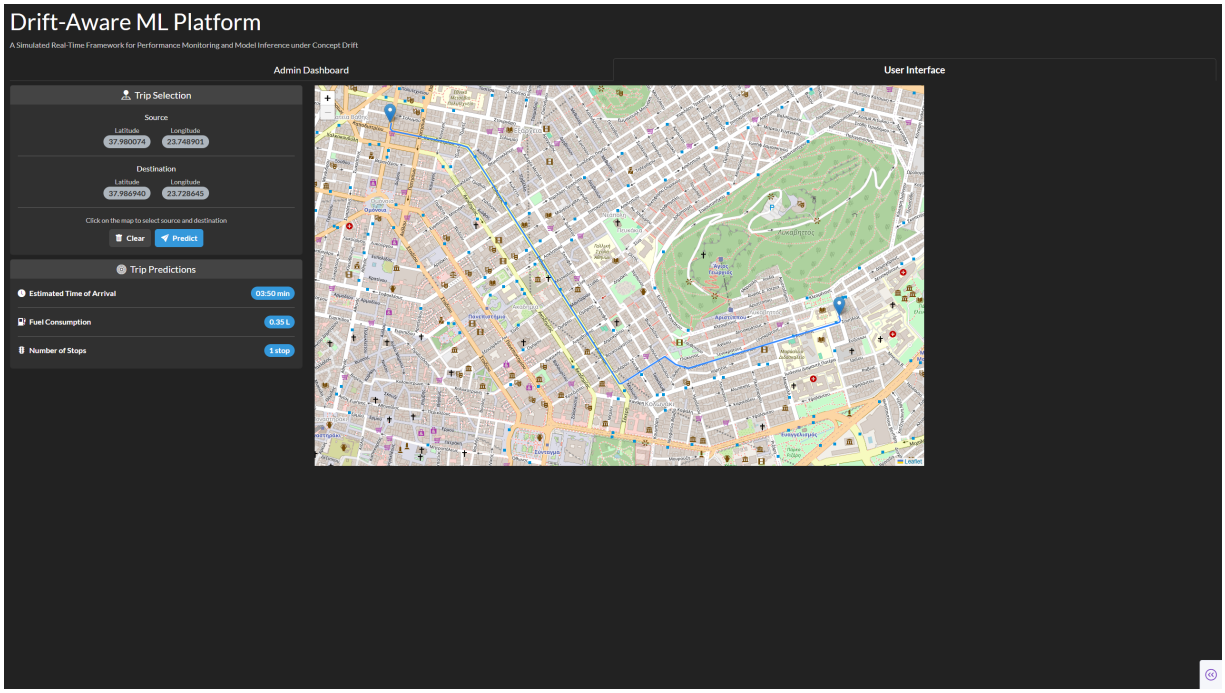


Figure 6.6: User interface — *Stable*. Example route with baseline predictions: `03:50 min`, `0.35 L`, `1 stop`.


Figure 6.7: Admin dashboard — *Drifted*. Rain conditions active, ETA/Fuel/Stops flagged as drifted and errors elevated.


Figure 6.8: Admin dashboard — *Retraining*. ETA retrained and swapped, Stops currently retraining, Fuel has not started retraining yet.

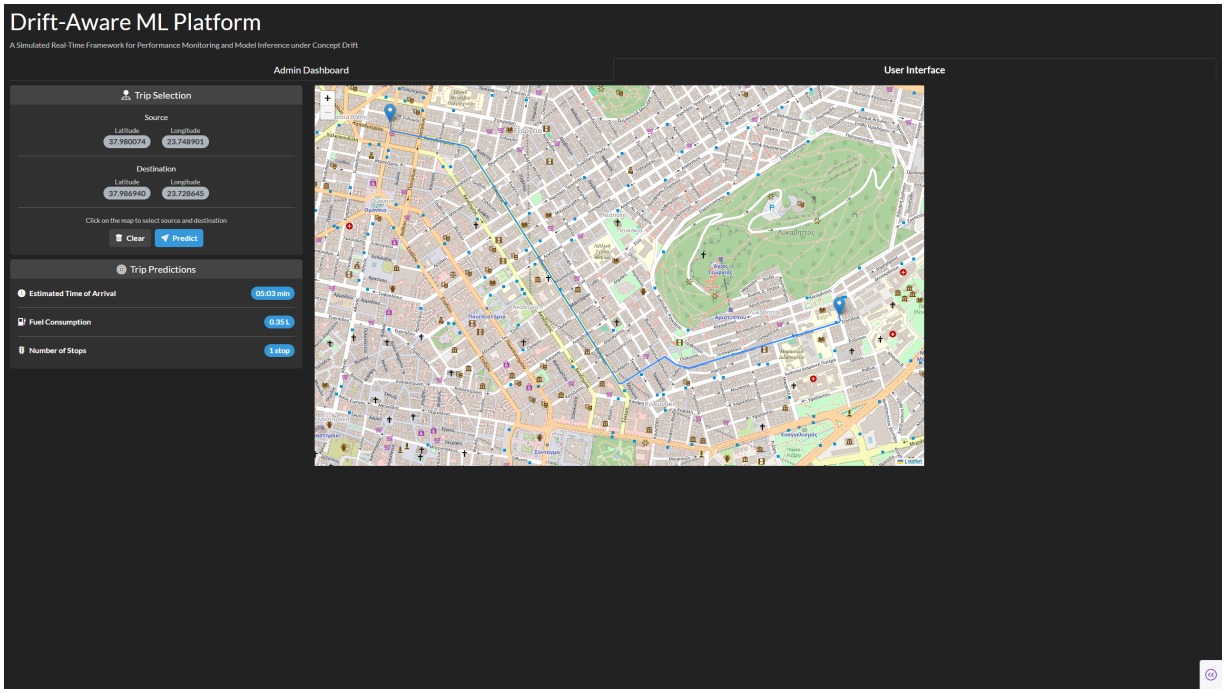


Figure 6.9: User interface — *Retraining*. Same route now reflects rain for ETA only: `05:03 min`, `0.35 L`, `1 stop`.


Figure 6.10: Admin dashboard — *Retrained*. All three tasks adapted and calibrated, system back to stable.

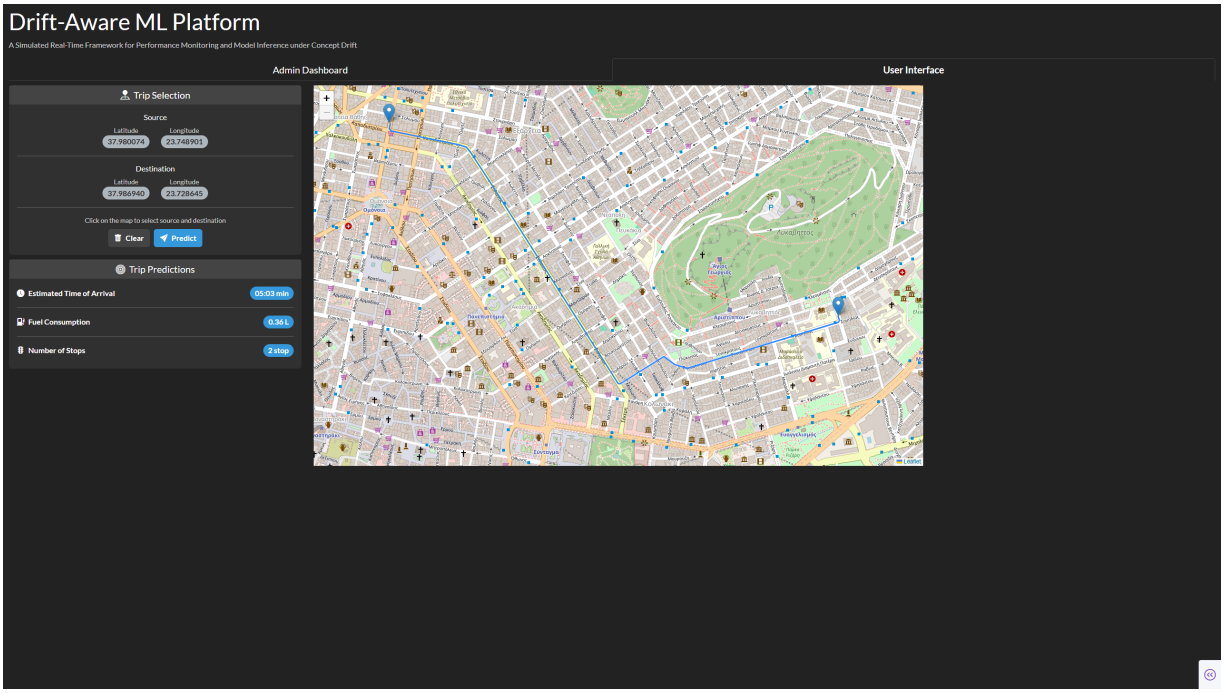


Figure 6.11: User interface — *Retrained*. Same route after full adaptation: `05:03 min`, `0.36 L`, `2 stops`.

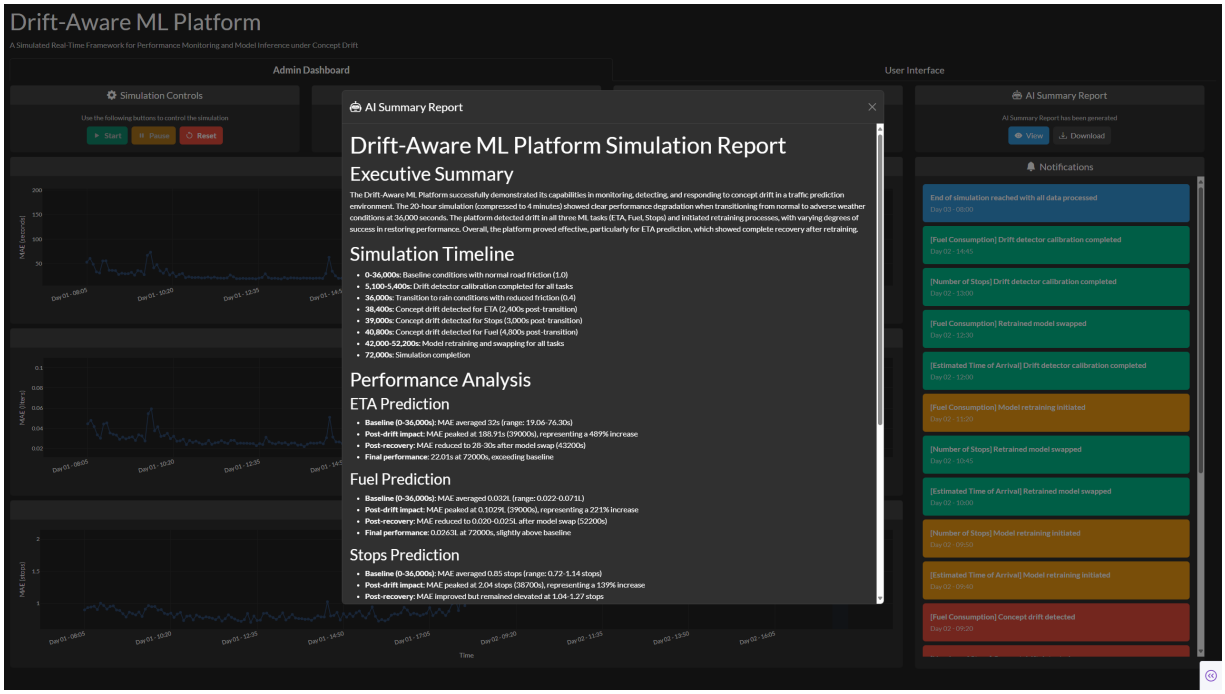


Figure 6.12: AI summary report shown in Admin Dashboard after simulation completion.

# Chapter 7

# Conclusion

## 7.1 Summary of Contributions

This thesis designed, implemented, and validated a drift-aware platform for continuous machine learning in cooperative, connected and automated mobility (CCAM) applications, using a central Athens case study. It delivered four concrete artifacts. First, a reproducible SUMO-based synthetic data generation pipeline produced train/test scenarios and a controlled concept-drift scenario via reduced road friction to emulate heavy rain. Second, an ETA prediction model based on gradient-boosted decision trees with domain-informed feature engineering was developed and tuned, serving as the primary workload to study drift. Third, a microservice platform was built to operate ML in a closed loop: it replays/streams data, monitors errors, detects drift via a complementary set of detectors with consensus, triggers retraining, and performs model swaps during operation with no downtime. Fourth, an evaluation methodology tied the above components together in a time-lapse experiment, demonstrating detection, adaptation, and recovery within one end-to-end system. Collectively, these contributions show that continuous machine learning for mobility can be realized with practical engineering trade-offs and clear operational boundaries, while maintaining a high level of reproducibility.

## 7.2 Key Findings

The study yields four main findings. (i) Controlled, domain-realistic drift matters: degrading road friction shifts ETA error distributions in ways that are observable, repeatable, and actionable, providing a reliable proving ground for adaptation policies. (ii) Consensus-based drift detection stabilizes decisions under noisy signals; requiring agreement among complementary detectors reduced spurious triggers without masking material changes. (iii) Warm retraining followed by a controlled swap restores performance effectively in the short term when the post-drift window is representative, indicating that periodic, targeted updates can be sufficient for tabular ETA workloads. (iv) Operational visibility is a prerequisite for trust: aligning dashboards, detector states, and deployment notifications with the simulation timeline made adaptation auditable and supported fast failure analysis when behavior deviated from expectation.

## 7.3 Limitations

Three limitations qualify the results. First, the data are realistic enough but still remain synthetic and scenario-bounded. While SUMO supports realistic dynamics even in a dense urban network like central Athens, the distributional support and covariate interactions are still curated. External validity to heterogeneous, multi-source urban data is not guaranteed. Second, features are primarily precomputed offline to simplify reproducibility and latency. This this underplays challenges of on-line feature materialization (late-arriving signals, joins, and leakage control). Third, the system operates at a single-node, Docker Compose scale. Although services are isolated and performant, questions of horizontal scaling and elasticity under bursty demand remain open.

## 7.4 Future Work

Future extensions follow from these limits and the platform's design. Real data ingestion should be added, for example using provider-specific FCD connectors, with schema validation and late-label handling. Online learning variants for ETA, like incremental boosting or streaming variants, merit evaluation as complements to the warm-retrain path, to reduce adaptation latency and detector sensitivity. Cloud-native deployment with autoscaling (container orchestration, queue backpressure, and GPU/CPU bin-packing) would enable elasticity under variable loads. A persistent store, acting as a feature store plus model registry, would support consistent real-time features, lineage, and rollback. The addition of a time-series database, like TimescaleDB, can retain metrics and notifications for historical analysis and trend detection. Continuous monitoring should be expanded beyond drift and error to include service-level objectives (e.g., tail inference latency), label/ground-truth availability, and post-deployment evaluation with delayed ground truth. Finally, the dataset simulation pipeline can include additional perturbations, such as incidents, closures, or demand shocks, to study interaction effects and to benchmark adaptation policies under multi-factor drift.

## 7.5 Closing Remarks

In summary, the thesis demonstrates that a drift-aware, closed-loop ML platform for urban mobility is both feasible and useful. Under a controlled but realistic drift, the system detected change, adapted promptly, and restored predictive quality, while keeping its operation observable. Bridging from this validated sandbox to production, where data are heterogeneous, labels are delayed, and scale is non-negotiable, defines the next phase: turning a working prototype into a dependable mobility service.

# Appendix A

# Code Availability

To ensure transparency and reproducibility, the complete source code associated with this diploma thesis has been made publicly available on GitHub at:

`https://github.com/geokyr/diploma-thesis`

The repository includes:

- Project documentation and usage instructions
- Dataset generation simulation with SUMO
- Machine learning research experiments
- Platform implementation (backend, predictor, drift, summarizer, frontend)
- Containerization and deployment artifacts (Dockerfiles, Docker Compose, uv)

*Note on Data Availability:* details on the public dataset and its DOI are provided in subsection 4.1.10.

# Bibliography


[1] ERTRAC Working Group "Connectivity and Automated Driving". *Connected, Cooperative and Automated Mobility Roadmap*. Tech. rep. Version 10. European Road Transport Research Advisory Council (ERTRAC), 2022.

[2] Abdi, A. and Amrit, C. "A Review of Travel and Arrival-Time Prediction Methods on Road Networks: Classification, Challenges and Opportunities". In: *PeerJ Computer Science* 7 (2021), e689.

[3] Arifi, A., Bouros, P., and Chondrogiannis, T. "A Study on ETA Prediction using Machine Learning and Recovered Routes". In: *Proceedings of the Workshops of the EDBT/ICDT 2024 Joint Conference*. CEUR Workshop Proceedings. Paestum, Italy: CEUR-WS.org, 2024.

[4] Gama, J., Žliobaitė, I., Bifet, A., Pechenizkiy, M., and Bouchachia, A. "A survey on concept drift adaptation". In: *ACM Computing Surveys (CSUR)* 46.4 (2014), pp. 1–37.

[5] Lu, J., Liu, A., Dong, F., Gu, F., Gama, J., and Zhang, G. "Learning under concept drift: A review". In: *IEEE Transactions on Knowledge and Data Engineering* 31.12 (2018), pp. 2346–2363.

[6] Baier, L., Hofmann, M., Kühl, N., Mohr, M., and Satzger, G. "Handling Concept Drifts in Regression Problems—the Error Intersection Approach". In: *arXiv preprint arXiv:2004.00438* (2020).

[7] Greco, S., Vacchetti, B., Apiletti, D., and Cerquitelli, T. "Unsupervised Concept Drift Detection from Deep Learning Representations in Real-time". In: *arXiv preprint arXiv:2406.17813* (2025).

[8] Xin, H., Zhang, X., Tang, R., Yan, S., Zhao, Q., Yang, C., Cui, W., and Yang, Z. "LitSim: A Conflict-aware Policy for Long-term Interactive Traffic Simulation". In: *arXiv preprint arXiv:2403.04299* (2024).

[9] Modesto, C., Borges, J., Nahum, C., Matni, L., Both, C. B., Cardoso, K., Gonçalves, G., Correa, I., Lins, S., Silva, A., and Klautau, A. "Towards a Robust Transport Network With Self-adaptive Network Digital Twin". In: *arXiv preprint arXiv:2507.20971* (2025).

[10] Krajzewicz, D., Erdmann, J., Behrisch, M., and Bieker, L. "Recent development and applications of SUMO - Simulation of Urban MObility". In: *International Journal On Advances in Systems and Measurements* 5.3&4 (2012), pp. 128–138.

[11] Weber, T., Driesch, P., and Schramm, D. "Introducing Road Surface Conditions into a Microscopic Traffic Simulation". In: *SUMO User Conference 2019*. EasyChair, 2019. URL: `https://easychair.org/publications/paper/3S4X`.

[12] Angelis, G. *Fuel Consumption Prediction Model*. Unpublished work, integrated in drift-aware platform. 2025.

[13] Tzelepis, S. *Number of Stops Prediction Model*. Unpublished work, integrated in drift-aware platform. 2025.

[14] Shalev-Shwartz, S. and Ben-David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[15] Sarker, I. H. "Machine Learning: Algorithms, Real-World Applications and Research Directions". In: *SN Computer Science* 2.160 (2021). DOI: 10.1007/s42979-021-00522-x.

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

[17] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". In: *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*. 2017, pp. 3146–3154.

[18] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., and Gulin, A. "CatBoost: unbiased boosting with categorical features". In: *Advances in Neural Information Processing Systems 31 (NeurIPS)* (2018), pp. 6638–6648.

[19] LeCun, Y., Bengio, Y., and Hinton, G. "Deep learning". In: *Nature* 521 (2015), pp. 436–444. DOI: 10.1038/nature14539.

[20] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[21] Parisi, G. I., Kemker, R., Part, J. L., Kanan, C., and Wermter, S. "Continual lifelong learning with neural networks: A review". In: *Neural Networks* 113 (2019), pp. 54–71. DOI: 10.1016/j.neunet.2019.01.012.

[22] Tabassam, A. I. U. "MLOps: A Step Forward to Enterprise Machine Learning". In: *arXiv preprint arXiv:2305.19298* (2023). URL: https://arxiv.org/abs/2305.19298.

[23] Bifet, A. and Gavaldà, R. "Learning from time-changing data with adaptive windowing". In: *Proceedings of the 2007 SIAM International Conference on Data Mining (SDM)*. 2007, pp. 443–448.

[24] Page, E. S. "Continuous Inspection Schemes". In: *Biometrika* 41.1/2 (1954), pp. 100–115. DOI: 10.2307/2333009.

[25] Jr., F. J. M. "The Kolmogorov-Smirnov Test for Goodness of Fit". In: *Journal of the American Statistical Association* 46.253 (1951), pp. 68–78.

[26] Montgomery, D. C. *Introduction to Statistical Quality Control*. 9th. Wiley, 2020.

[27] Montiel, J., Halford, M., Marti, S., Becker, C., Gonzalez-Hernandez, A., Lendasse, A., and Read, J. "River: machine learning for streaming data in Python". In: *Journal of Machine Learning Research* 22.111 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1346.html.

[28] Montiel, J., Read, J., Bifet, A., and Abdessalem, T. "Scikit-Multiflow: A Multi-output Streaming Framework". In: *Journal of Machine Learning Research* 19.72 (2018), pp. 1–5. URL: http://jmlr.org/papers/v19/18-251.html.

[29] Looveren, A. V. and Klyn, J. "Alibi Detect: Outlier, Adversarial and Concept Drift Detection for Tabular, Text and Image Data". In: *NeurIPS 2020 Workshop on Machine Learning Open Source Software*. 2020. URL: https://arxiv.org/abs/2006.07272.

[30] AI, E. *Evidently AI Documentation*. https://docs.evidentlyai.com. Accessed 2025-10-26. 2025.

[31] Grootendorst, M., Koning, B. de, Miltenburg, E. van, and Walt, S. van der. "Confidence-Based Performance Estimation for Post-Deployment Machine Learning Models". In: *arXiv preprint arXiv:2305.19388* (2023). URL: https://arxiv.org/abs/2305.19388.

[32] Duan, Y., Zhang, L., and Song, D. "Travel Time Prediction using LSTM Neural Network". In: *International Journal of Computer Science and Mobile Computing* 5 (2016), pp. 44–53.

[33] Shen, Y., Wang, D., Li, J., Wang, Z., and Chen, X. "DeepTTE: Predicting Travel Time with Deep Neural Network Architecture". In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)* (2019).

[34] Guin, A. "Travel Time Prediction Using Gradient Boosting Regression Tree". In: *Proceedings of the 7th International Conference on Applications of Advanced Technologies in Transportation*. 2006.

[35] Zhang, L. and Haghani, A. "Travel Time Prediction Using GBDT". In: *Transportation Research Record* 2442 (2015), pp. 45–55.

[36] Athens Mobility Observatory, National Technical University of Athens. *Athens Mobility Observatory*. `https://amob.ntua.gr/traffic/`. Accessed 2025-10-26. 2025.

[37] Kyriakopoulos, G. *Synthetic 10-Hour Traffic Simulations for Central Athens (Train/Test/Rain)*. Zenodo, Aug. 2025. DOI: `10.5281/zenodo.16950674`. URL: `https://zenodo.org/records/16950674`.

[38] Lundberg, S. M. and Lee, S.-I. "A Unified Approach to Interpreting Model Predictions". In: *Advances in Neural Information Processing Systems 30 (NIPS)*. 2017, pp. 4765–4774.

[39] Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. "Optuna: A Next-generation Hyperparameter Optimization Framework". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), pp. 2623–2631. DOI: `10.1145/3292500.3330701`.

[40] Newman, S. "Building Microservices". In: *O'Reilly Media* (2015).

[41] Fielding, R. T. "Architectural Styles and the Design of Network-based Software Architectures". In: *Doctoral dissertation, University of California, Irvine* (2000). URL: `https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.

[42] Maheshwari, H., Yang, L., and Pazzi, R. W. "Machine Learning Advancements in Urban Traffic Simulation: A Comprehensive Survey". In: *IEEE Open Journal of Intelligent Transportation Systems* 6 (2025), pp. 1027–1052.

[43] Zliobaitė, I. "A survey on concept drift adaptation". In: *ACM Computing Surveys (CSUR)* 46.4 (2014). DOI: `10.1145/2523813`.

[44] Foundation, E. and contributors. *Eclipse SUMO - Simulation of Urban MObility*. `https://eclipse.dev/sumo/`. Accessed 2025-10-26. 2024.

[45] Espinosa, A. V. "Traffic Modeling with SUMO: a Tutorial". In: *arXiv preprint arXiv:2304.05982* (2021). URL: `https://arxiv.org/abs/2304.05982`.

[46] Merkel, D. "Docker: lightweight Linux containers for consistent development and deployment". In: *Linux Journal* 239 (2014), p. 2.

[47] Inc., D. *Docker Compose Documentation*. `https://docs.docker.com/compose/`. Accessed 2025-10-26. 2025.

[48] Ramírez, S. and contributors. *FastAPI Documentation*. `https://fastapi.tiangolo.com/`. Accessed 2025-10-26. 2025.

[49] Inc., P. T. *Dash Documentation & User Guide*. `https://dash.plotly.com/`. Accessed 2025-10-26. 2025.

[50] Inc., P. T. *Plotly Python Documentation*. `https://plotly.com/python/`. Accessed 2025-10-26. 2025.

[51] Colvin, S. and contributors. *Pydantic Documentation*. `https://docs.pydantic.dev/`. Accessed 2025-10-26. 2025.

[52] Vohra, D. "Apache Parquet". In: *Practical Hadoop Ecosystem*. Apress, 2016, pp. 325–351.

[53] Müller, R., Boscolo, G., Wirth, F., and Niggemann, O. "Open-Source Drift Detection Tools in Action". In: *arXiv preprint arXiv:2404.18673* (2024). URL: `https://arxiv.org/abs/2404.18673`.

[54] Hyndman, R. J. and Athanasopoulos, G. *Forecasting: principles and practice*. OTexts, 2018. URL: https://otexts.com/fpp2/arima.html.

[55] Wang, D., Sun, W., Wang, Z., and Li, J. "DeepTTE: Predicting Travel Time with Recurrent Neural Network Architecture". In: *Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI)* (2018).

[56] Li, M., Feng, Y., and Wu, X. "AttentionTTE: A Deep Learning Model for Estimated Time of Arrival". In: *Frontiers in Artificial Intelligence* 7 (2024), p. 1258086. DOI: 10.3389/frai.2024.1258086.

[57] Guo, Y., Wu, Y., Mao, F., Wang, E., Zhang, J., and Hu, X. "Wide & Deep Learning for Recommender Systems". In: *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems - DLRS '19*. 2019. DOI: 10.1145/3331184.3331287. URL: https://dl.acm.org/doi/10.1145/3331184.3331287.

[58] Qian, Z., Zhang, Z., Xu, Z., Fan, X., and Wang, M. "CoDriver: Multi-Task Learning for Driver Behavior and Route Prediction". In: *Proceedings of the 29th ACM International Conference on Multimedia*. 2021, pp. 2957–2965. DOI: 10.1145/3474085.3475550. URL: https://dl.acm.org/doi/10.1145/3474085.3475550.

[59] Derrow-Pinion, A., She, J., Wong, D., Lange, O., Hester, T., Perez, L., Nunkesser, M., Lee, S., Guo, X., Wiltshire, B., Battaglia, P. W., Gupta, V., Li, A., Xu, Z., Sanchez-Gonzalez, A., Li, Y., and Veličković, P. "ETA Prediction with Graph Neural Networks in Google Maps". In: *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM)*. 2021. DOI: 10.1145/3459637.3481916.

[60] Jia, A.-F. and Guo, X.-Y. "Spatio-Temporal Graph Neural Networks for Urban Traffic Flow Prediction". In: *IEEE Transactions on Knowledge and Data Engineering* (2021).

[61] Guo, X.-Y. and Jia, A.-F. "Spatio-Temporal Graph Neural Networks for Urban Traffic Flow Prediction". In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[62] Antypas, D., Psychas, A., Mylonaki, N.-K., Stergiopoulou, D., and Giakoumakis, G. "Bus ETA Prediction Using Gradient Boosting Machines in a Smart City Context". In: *IEEE Access* 10 (2022), pp. 55007–55019. DOI: 10.1109/ACCESS.2022.3178195.

[63] Taxi, N. and Commission, L. *New York City Taxi Trip Duration Dataset*. https://www.kaggle.com/c/nyc-taxi-trip-duration. Accessed October 2025.

[64] OpenStreetMap contributors. *OpenStreetMap*. https://www.openstreetmap.org. Accessed 2025-10-26. 2025.

[65] Hellenic Statistical Authority. *Greece Vehicle Fleet Statistics*. https://www.statistics.gr/en/statistics/-/publication/SME18/-. Accessed 2025-10-26. 2025.

[66] Krauss, S. "Microscopic Modeling of Traffic Flow: Investigation of Collision Free Vehicle Dynamics". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 356.1742 (1998), pp. 927–938. DOI: 10.1098/rsta.1998.0196.

[67] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, É. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[68] Box, G. E. and Cox, D. R. "An Analysis of Transformations". In: *Journal of the Royal Statistical Society Series B (Methodological)* 26 (1964), pp. 211–243.

[69] Himeno, R., Kojima, K., Cecotti, H., and Kubota, M. "Progress in Normalization Methods for Radar Signal Processing". In: *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 92.4 (2009), pp. 926–930.

[70] Lloyd, S. P. "Least Squares Quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[71] Jolliffe, I. T. “Principal Component Analysis”. In: *Springer Series in Statistics* (1986).
[72] Breiman, L. “Random Forests”. In: *Machine Learning* 45 (2001), pp. 5–32.
[73] Pearson, K. “Notes on Regression and Inheritance in the Case of Two Parents”. In: *Proceedings of the Royal Society of London* 58 (1895), pp. 240–242.
[74] Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. “Algorithms for Hyper-parameter Optimization”. In: *Advances in Neural Information Processing Systems 24 (NIPS)*. 2011, pp. 2546–2554.
[75] Astral. *uv Documentation*. 2025. URL: `https://docs.astral.sh/uv/`.
[76] Christie, T. and contributors. *HTTPX: Async HTTP client for Python*. `https://www.python-httpx.org/`. Accessed 2025-10-26. 2025.
[77] McKinney, W. “Data Structures for Statistical Computing in Python”. In: *Proceedings of the 9th Python in Science Conference* (2010), pp. 56–61.
[78] Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., and al., et. “Array programming with NumPy”. In: *Nature* 585 (2020), pp. 357–362.
[79] Developers, A. A. *Apache Arrow: PyArrow*. `https://arrow.apache.org/docs/python/`. Accessed 2025-10-26. 2025.