Real-time Demand Forecasting for an Urban Delivery Platform

Alexander Hess[a,1], Stefan Spinler[a,1], Matthias Winkenbach[b,1]

[a] *WHU - Otto Beisheim School of Management, Burgplatz 2, 56179 Vallendar, Germany*
[b] *Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, United States*

Abstract

Meal delivery platforms like Uber Eats shape the landscape in cities around the world. This paper addresses forecasting demand into the short-term future. We propose an approach incorporating both classical forecasting and machine learning methods. Model evaluation and selection is adapted to demand typical for such a platform (i.e., intermittent with a double-seasonal pattern). The results of an empirical study with a European meal delivery service show that machine learning models become competitive once the average daily demand passes a threshold. As a main contribution, the paper explains how a forecasting system must be set up to enable predictive routing.

Keywords: demand forecasting, intermittent demand, machine learning, urban delivery platform

[1]Emails: alexander.hess@whu.edu, stefan.spinler@whu.edu, mwinkenb@mit.edu

1. Introduction

In recent years, many meal delivery platform providers (e.g., Uber Eats, GrubHub, DoorDash, Deliveroo) with different kinds of business models have entered the markets in cities around the world. A study by [26] estimates the global market size to surpass 20 billion Dollars by 2025. A common feature of these platforms is that they do not operate kitchens but focus on marketing their partner restaurants' meals, unifying all order related processes in simple smartphone apps, and managing the delivery via a fleet of either employees or crowd-sourced sub-contractors.

Various kind of urban delivery platforms (UDP) have received attention in recent scholarly publications. [29] look into heuristics to simultaneously optimize courier scheduling and routing in general, while [40] do so for the popular dial-a-ride problem and [56] investigate the effect of different fulfillment strategies in the context of urban meal delivery. [21] and [1] focus their research on the routing aspect, which is commonly modeled as a so-called vehicle routing problem (VRP).

Not covered in the recent literature is research focusing on the demand forecasting problem a UDP faces. Due to the customers' fragmented locations and the majority of the orders occurring ad-hoc for immediate delivery in the case of a meal delivery platform, forecasting demand for the near future (i.e., several hours) and distinct locations of the city in real-time is an essential factor in achieving timely fulfillment. In general, demand forecasting is a well-researched discipline with a decades-long history in scholarly journals as summarized, for example, by [19]. Even some meal delivery platforms themselves publish their practices: For example, [5] provide a general overview of supply and demand forecasting at Uber and benchmarks of the methods used while [38] investigate how extreme events can be incorporated.

The conditions such platforms face are not limited to meal delivery: Any entity that performs ad-hoc requested point-to-point transportation at scale in an urban area benefits from a robust forecasting system. Examples include ride-hailing, such as the original Uber offering, or bicycle courier services. The common characteristics are:

- **Geospatial Slicing**: Forecasts for distinct parts of a city in parallel

- **Temporal Slicing**: Forecasts on a sub-daily basis (e.g., 60-minute windows)
- **Order Sparsity**: The historical order time series exhibit an intermittent pattern
- **Double Seasonality**: Demand varies with the day of the week and the time of day

Whereas the first two points can be assumed to vary with the concrete application's requirements, it is the last two that pose challenges for forecasting a platform's demand: Intermittent demand (i.e., many observations in the historic order time series exhibit no demand at all) renders most of the commonly applied error metrics useless. Moreover, many of the established forecasting methods can only handle a single and often low seasonality (i.e., repeated regular pattern), if at all.

In this paper, we develop a rigorous methodology as to how to build and evaluate a robust forecasting system for an urban delivery platform (UDP) that offers ad-hoc point-to-point transportation of any kind. We implement such a system with a broad set of commonly used forecasting methods. We not only apply established (i.e., "classical") time series methods but also machine learning (ML) models that have gained traction in recent years due to advancements in computing power and availability of larger amounts of data. In that regard, the classical methods serve as benchmarks for the ML methods. Our system is trained on and evaluated with a dataset obtained from an undisclosed industry partner that, during the timeframe of our study, was active in several European countries and, in particular, in France. Its primary business strategy is the delivery of meals from upper-class restaurants to customers in their home or work places via bicycles. In this empirical study, we identify the best-performing methods. Thus, we answer the following research questions:

Q1: Which forecasting methods work best under what circumstances?
Q2: How do classical forecasting methods compare with ML models?
Q3: How does the forecast accuracy change with more historic data available?
Q4: Can real-time information on demand be exploited?
Q5: Can external data (e.g., weather data) improve the forecast accuracy?

To the best of our knowledge, no such study has yet been published in a scholarly journal.

The subsequent Section 2 reviews the literature on the forecasting methods included in the system. Section 3 introduces our forecasting system, and Section 4 discusses the results obtained in the empirical study. Lastly, Section 5 summarizes our findings and concludes with an outlook on further research opportunities.

2. Literature Review

In this section, we review the specific forecasting methods that make up our forecasting system. We group them into classical statistics and ML models. The two groups differ mainly in how they represent the input data and how accuracy is evaluated.

A time series is a finite and ordered sequence of equally spaced observations. Thus, time is regarded as discrete and a time step as a short period. Formally, a time series Y is defined as $Y = \{y_t : t \in I\}$, or y_t for short, where I is an index set of positive integers. Besides its length $T = |Y|$, another property is the a priori fixed and non-negative periodicity k of a seasonal pattern in demand: k is the number of time steps after which a pattern repeats itself (e.g., $k = 12$ for monthly sales data).

2.1. Demand Forecasting with Classical Forecasting Methods

Forecasting became a formal discipline starting in the 1950s and has its origins in the broader field of statistics. [30] provide a thorough overview of the concepts and methods established, and [44] indicate business-related applications such as demand forecasting. These "classical" forecasting methods share the characteristic that they are trained over the entire Y first. Then, for prediction, the forecaster specifies the number of time steps for which he wants to generate forecasts. That is different for ML models.

2.1.1. Naïve Methods, Moving Averages, and Exponential Smoothing.

Simple forecasting methods are often employed as a benchmark for more sophisticated ones. The so-called naïve and seasonal naïve methods forecast the next time step in a time series, y_{T+1}, with the last observation, y_T, and, if a seasonal pattern is present, with the observation k steps before, y_{T+1-k}. As variants, both methods can be generalized to include drift terms in the presence of a trend or changing seasonal amplitude.

If a time series exhibits no trend, a simple moving average (SMA) is a generalization of the naïve method that is more robust to outliers. It is defined as follows: $\hat{y}_{T+1} = \frac{1}{h} \sum_{i=T-h}^{T} y_i$

where h is the horizon over which the average is calculated. If a time series exhibits a seasonal pattern, setting h to a multiple of the periodicity k suffices that the forecast is unbiased.

Starting in the 1950s, another popular family of forecasting methods, so-called exponential smoothing methods, was introduced by [13], [28], and [58]. The idea is that forecasts $\hat{y}_{T+1}$ are a weighted average of past observations where the weights decay over time; in the case of the simple exponential smoothing (SES) method we obtain: $\hat{y}_{T+1} = \alpha y_T + \alpha(1-\alpha)y_{T-1} + \alpha(1-\alpha)^2 y_{T-2} + \cdots + \alpha(1-\alpha)^{T-1} y_1$ where α (with $0 \leq \alpha \leq 1$) is a smoothing parameter.

Exponential smoothing methods are often expressed in an alternative component form that consists of a forecast equation and one or more smoothing equations for unobservable components. Below, we present a generalization of SES, the so-called Holt-Winters' seasonal method, in an additive formulation. ℓ_t, b_t, and s_t represent the unobservable level, trend, and seasonal components inherent in y_t, and β and γ complement α as smoothing parameters:

$$\begin{aligned}
\hat{y}_{t+1} &= \ell_t + b_t + s_{t+1-k} \\
\ell_t &= \alpha(y_t - s_{t-k}) + (1-\alpha)(\ell_{t-1} + b_{t-1}) \\
b_t &= \beta(\ell_t - \ell_{t-1}) + (1-\beta)b_{t-1} \\
s_t &= \gamma(y_t - \ell_{t-1} - b_{t-1}) + (1-\gamma)s_{t-k}
\end{aligned}$$

With b_t, s_t, β, and γ removed, this formulation reduces to SES. Distinct variations exist: Besides the three components, [22] add dampening for the trend, [45] provides multiplicative formulations, and [52] adds dampening to the latter. The accuracy measure commonly employed is the sum of squared errors between the observations and their forecasts.

Originally introduced by [2], [31] show how the Theta method can be regarded as an equivalent to SES with a drift term. We mention this method here only because [5] emphasize that it performs well at Uber. However, in our empirical study, we find that this is not true in general.

[35] introduce statistical processes, so-called innovations state-space models, to generalize the methods in this sub-section. They call this family of models ETS as they capture error,

trend, and seasonal terms. Linear and additive ETS models have a structure like so:

$$y_t = \vec{w} \cdot \vec{x}_{t-1} + \epsilon_t$$

$$\vec{x_t} = \boldsymbol{F}\vec{x}_{t-1} + \vec{g}\epsilon_t$$

y_t denote the observations as before while $\vec{x}_t$ is a state vector of unobserved components. ϵ_t is a white noise series and the matrix $\boldsymbol{F}$ and the vectors $\vec{g}$ and $\vec{w}$ contain a model's coefficients. Just as the models in the next sub-section, ETS models are commonly fitted with maximum likelihood and evaluated using information theoretical criteria against historical data. We refer to [34] for a thorough summary.

2.1.2. Autoregressive Integrated Moving Averages.

[7], [8], and more papers by the same authors in the 1960s introduce a type of model where observations correlate with their neighbors and refer to them as autoregressive integrated moving average (ARIMA) models for stationary time series. For a thorough overview, we refer to [9] and [12].

A time series y_t is stationary if its moments are independent of the point in time where it is observed. A typical example is a white noise ϵ_t series. Therefore, a trend or seasonality implies non-stationarity. [37] provide a test to check the null hypothesis of stationary data. To obtain a stationary time series, one chooses from several techniques: First, to stabilize a changing variance (i.e., heteroscedasticity), one applies a Box-Cox transformation (e.g., log) as first suggested by [6]. Second, to factor out a trend (or seasonal) pattern, one computes differences of consecutive (or of lag k) observations or even differences thereof. Third, it is also common to pre-process y_t with one of the decomposition methods mentioned in Sub-section 2.1.3 below with an ARIMA model then trained on an adjusted y_t.

In the autoregressive part, observations are modeled as linear combinations of its predecessors. Formally, an $AR(p)$ model is defined with a drift term c, coefficients ϕ_i to be estimated (where i is an index with $0 < i \leq p$), and white noise ϵ_t like so: $AR(p)$: $y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + \epsilon_t$. The moving average part considers observations to be regressing towards a linear combination of past forecasting errors. Formally, a $MA(q)$ model is defined with a drift term c, coefficients θ_j to be estimated, and white noise terms ϵ_t (where j is an index with $0 < j \leq q$) as follows: $MA(q)$: $y_t =$

$c + \epsilon_t + \theta_1\epsilon_{t-1} + \theta_2\epsilon_{t-2} + \cdots + \theta_q\epsilon_{t-q}$. Finally, an $ARIMA(p, d, q)$ model unifies both parts and adds differencing where d is the degree of differences and the $'$ indicates differenced values: $ARIMA(p, d, q): \quad y'_t = c + \phi_1 y'_{t-1} + \cdots + \phi_p y'_{t-p} + \theta_1\epsilon_{t-1} + \cdots + \theta_q\epsilon_{t-q} + \epsilon_t$.

$ARIMA(p, d, q)$ models are commonly fitted with maximum likelihood estimation. To find an optimal combination of the parameters p, d, and q, the literature suggests calculating an information theoretical criterion (e.g., Akaike's Information Criterion) that evaluates the fit on historical data. [32] provide a step-wise heuristic to choose p, d, and q, that also decides if a Box-Cox transformation is to be applied, and if so, which one. To obtain a one-step-ahead forecast, the above equation is reordered such that t is substituted with $T + 1$. For forecasts further into the future, the actual observations are subsequently replaced by their forecasts. Seasonal ARIMA variants exist; however, the high frequency k in the kind of demand a UDP faces typically renders them impractical as too many coefficients must be estimated.

2.1.3. Seasonal and Trend Decomposition using Loess.

A time series y_t may exhibit different types of patterns; to fully capture each of them, the series must be decomposed. Then, each component is forecast with a distinct model. Most commonly, the components are the trend t_t, seasonality s_t, and remainder r_t. They are themselves time series, where only s_t exhibits a periodicity k. A decomposition may be additive (i.e., $y_t = s_t + t_t + r_t$) or multiplicative (i.e., $y_t = s_t * t_t * r_t$); the former assumes that the effect of the seasonal component is independent of the overall level of y_t and vice versa. The seasonal component is centered around 0 in both cases such that its removal does not affect the level of y_t. Often, it is sufficient to only seasonally adjust the time series, and model the trend and remainder together, for example, as $a_t = y_t - s_t$ in the additive case.

Early approaches employed moving averages (cf., Sub-section 2.1.1) to calculate a trend component, and, after removing that from y_t, averaged all observations of the same seasonal lag to obtain the seasonal component. The downsides of this are the subjectivity in choosing the window lengths for the moving average and the seasonal averaging, the incapability of the seasonal component to vary its amplitude over time, and the missing handling of outliers.

The X11 method developed at the U.S. Census Bureau and described in detail by [18] overcomes these disadvantages. However, due to its background in economics, it is designed primarily for quarterly or monthly data, and the change in amplitude over time cannot be controlled. Variants of this method are the SEATS decomposition by the Bank of Spain and the newer X13-SEATS-ARIMA method by the U.S. Census Bureau. Their main advantages stem from the fact that the models calibrate themselves according to statistical criteria without manual work for a statistician and that the fitting process is robust to outliers.

[16] introduce a seasonal and trend decomposition using a repeated locally weighted regression - the so-called Loess procedure - to smoothen the trend and seasonal components, which can be viewed as a generalization of the methods above and is denoted by the acronym STL. In contrast to the X11, X13, and SEATS methods, the STL supports seasonalities of any lag k that must, however, be determined with additional statistical tests or set with out-of-band knowledge by the forecaster (e.g., hourly demand data implies $k = 24 * 7 = 168$ assuming customer behavior differs on each day of the week). Moreover, the seasonal component's rate of change, represented by the ns parameter and explained in detail with Figure 4 in Section 3.5, must be set by the forecaster as well, while the trend's smoothness may be controlled via setting a non-default window size. Outliers are handled by assignment to the remainder such that they do not affect the trend and seasonal components. In particular, the manual input needed to calibrate the STL explains why only the X11, X13, and SEATS methods are widely used by practitioners. However, the widespread adoption of concepts like cross-validation (cf., Sub-section 2.2.2) in recent years enables the usage of an automated grid search to optimize the parameters. The STL's usage within a grid search is facilitated even further by its being computationally cheaper than the other methods discussed.

2.2. Demand Forecasting with Machine Learning Methods

ML methods have been employed in all kinds of prediction tasks in recent years. In this section, we restrict ourselves to the models that performed well in our study: Random Forest (RF) and Support Vector Regression (SVR). RFs are in general well-suited for datasets without a priori knowledge about the patterns, while SVR is known to perform well on time series data, as shown by [23] in general and [3] specifically for intermittent demand. Gradient Boosting, another popular ML method, was consistently outperformed by RFs, and artificial neural networks require an amount of data exceeding what our industry partner has by far.

2.2.1. Supervised Learning.

A conceptual difference between classical and ML methods is the format for the model inputs. In ML models, a time series Y is interpreted as labeled data. Labels are collected into a vector $\vec{y}$ while the corresponding predictors are aligned in an $(T-n) \times n$ matrix $\boldsymbol{X}$:

$$\vec{y} = \begin{pmatrix} y_T \\ y_{T-1} \\ \dots \\ y_{n+1} \end{pmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} y_{T-1} & y_{T-2} & \cdots & y_{T-n} \\ y_{T-2} & y_{T-3} & \cdots & y_{T-(n+1)} \\ \dots & \dots & \dots & \dots \\ y_n & y_{n-1} & \cdots & y_1 \end{bmatrix}$$

The $m = T - n$ rows are referred to as samples and the n columns as features. Each row in $\boldsymbol{X}$ is "labeled" by the corresponding entry in $\vec{y}$, and ML models are trained to fit the rows to their labels. Conceptually, we model a functional relationship f between $\boldsymbol{X}$ and $\vec{y}$ such that the difference between the predicted $\vec{\hat{y}} = f(\boldsymbol{X})$ and the true $\vec{y}$ are minimized according to some error measure $L(\vec{\hat{y}}, \vec{y})$, where L summarizes the goodness of the fit into a scalar value (e.g., the well-known mean squared error [MSE]; cf., Section 3.4). $\boldsymbol{X}$ and $\vec{y}$ show the ordinal character of time series data: Not only overlap the entries of $\boldsymbol{X}$ and $\vec{y}$, but the rows of $\boldsymbol{X}$ are shifted versions of each other. That does not hold for ML applications in general (e.g., the classical example of predicting spam vs. no spam emails, where the features model properties of individual emails), and most of the common error measures presented in introductory texts on ML, are only applicable in cases without such a structure in $\boldsymbol{X}$ and $\vec{y}$. n, the number of past time steps required to predict a y_t, is an exogenous model parameter. For prediction, the forecaster supplies the trained ML model an input vector in the same format as a row $\vec{x}_i$ in $\boldsymbol{X}$. For example, to predict y_{T+1}, the model takes the vector $(y_T, y_{T-1}, ..., y_{T-n+1})$ as input. That is in contrast to the classical methods, where we only supply the number of time steps to be predicted as a scalar integer.

2.2.2. Cross-Validation.

Because ML models are trained by minimizing a loss function L, the resulting value of L underestimates the true error we see when predicting into the actual future by design. To counter that, one popular and model-agnostic approach is cross-validation (CV), as summarized, for example, by [24]. CV is a resampling technique, which ranomdly splits the samples into a training and a test set. Trained on the former, an ML model makes forecasts on the latter. Then, the value of L calculated only on the test set gives a realistic and unbiased estimate of the true forecasting error, and may be used for one of two distinct aspects: First, it assesses the quality of a fit and provides an idea as to how the model would perform in production when predicting into the actual future. Second, the errors of models of either different methods or the same method with different parameters may be compared with each other to select the best model. In order to first select the best model and then assess its quality, one must apply two chained CVs: The samples are divided into training, validation, and test sets, and all models are trained on the training set and compared on the validation set. Then, the winner is retrained on the union of the training and validation sets and assessed on the test set.

Regarding the splitting, there are various approaches, and we choose the so-called k-fold CV, where the samples are randomly divided into k folds of the same size. Each fold is used as a test set once and the remaining $k-1$ folds become the corresponding training set. The resulting k error measures are averaged. A k-fold CV with $k=5$ or $k=10$ is a compromise between the two extreme cases of having only one split and the so-called leave-one-out CV where $k=m$: Computation is still relatively fast and each sample is part of several training sets maximizing the learning from the data. We adapt the k-fold CV to the ordinal stucture in $\boldsymbol{X}$ and $\vec{y}$ in Sub-section 3.3.

2.2.3. Random Forest Regression.

[11] introduce the classification and regression tree (CART) model that is built around the idea that a single binary decision tree maps learned combinations of intervals of the feature columns to a label. Thus, each sample in the training set is associated with one leaf node that is reached by following the tree from its root and branching along the arcs according to some learned splitting rule per intermediate node that compares the sample's realization for the feature specified by the rule to the learned decision rule. While such models are computationally fast and offer a high degree of interpretability, they tend to overfit strongly to the training set as the splitting rules are not limited to any functional form (e.g., linear) in the relationship between the features and the labels. In the regression case, it is common

to maximize the variance reduction I_V from a parent node N to its two children, $C1$ and $C2$, as the splitting rule. [11] formulate this as follows:

$$I_V(N) = \frac{1}{|S_N|^2}\sum_{i\in S_N}\sum_{j\in S_N}\frac{1}{2}(y_i - y_j)^2 - \left(\frac{1}{|S_{C1}|^2}\sum_{i\in S_{C1}}\sum_{j\in S_{C1}}\frac{1}{2}(y_i - y_j)^2 + \frac{1}{|S_{C2}|^2}\sum_{i\in S_{C2}}\sum_{j\in S_{C2}}\frac{1}{2}(y_i - y_j)^2\right)$$

S_N, S_{C1}, and S_{C2} are the index sets of the samples in N, $C1$, and $C2$.

[27] and then [10] generalize this method by combining many CART models into one forest of trees where every single tree is a randomized variant of the others. Randomization is achieved at two steps in the training process: First, each tree receives a distinct training set resampled with replacement from the original training set, an idea also called bootstrap aggregation. Second, at each node a random subset of the features is used to grow the tree. Trees can be fitted in parallel speeding up the training significantly. For prediction at the tree level, the average of all the samples at a particular leaf node is used. Then, the individual values are combined into one value by averaging again across the trees. Due to the randomization, the trees are decorrelated offsetting the overfitting. Another measure to counter overfitting is pruning the tree, either by specifying the maximum depth of a tree or the minimum number of samples at leaf nodes.

The forecaster must tune the structure of the forest. Parameters include the number of trees in the forest, the size of the random subset of features, and the pruning criteria. The parameters are optimized via grid search: We train many models with parameters chosen from a pre-defined list of values and select the best one by CV. RFs are a convenient ML method for any dataset as decision trees do not make any assumptions about the relationship between features and labels. [25] use RFs to predict the hourly demand for water in an urban context, a similar application as the one in this paper, and find that RFs work well with time series type of data.

2.2.4. Support Vector Regression.

[55] and [54] introduce the so-called support vector machine (SVM) model, and [53] summarizes the research conducted since then. In its basic version, SVMs are linear classifiers, modeling a binary decision, that fit a hyperplane into the feature space of $\boldsymbol{X}$ to maximize the margin around the hyperplane seperating the two groups of labels. SVMs were popularized in the 1990s in the context of optical character recognition, as shown in [47].

[20] and [50] adapt SVMs to the regression case, and [49] provide a comprehensive introduction thereof. [42] and [43] focus on SVRs in the context of time series data and find that they tend to outperform classical methods. [14] and [15] apply SVRs to predict the hourly demand for water in cities, an application similar to the UDP case.

In the SVR case, a linear function $\hat{y}_i = f(\vec{x}_i) = \langle \vec{w}, \vec{x}_i \rangle + b$ is fitted so that the actual labels y_i have a deviation of at most ϵ from their predictions $\hat{y}_i$ (cf., the constraints below). SVRs are commonly formulated as quadratic optimization problems as follows:

$$\text{minimize } \frac{1}{2} \left\| \vec{w} \right\|^2 + C \sum_{i=1}^{m} (\xi_i + \xi_i^*) \quad \text{subject to } \begin{cases} y_i - \langle \vec{w}, \vec{x}_i \rangle - b \leq \epsilon + \xi_i, \\ \langle \vec{w}, \vec{x}_i \rangle + b - y_i \leq \epsilon + \xi_i^* \end{cases}$$

$\vec{w}$ are the fitted weights in the row space of $\boldsymbol{X}$, b is a bias term in the column space of $\boldsymbol{X}$, and $\langle \cdot, \cdot \rangle$ denotes the dot product. By minimizing the norm of $\vec{w}$, the fitted function is flat and not prone to overfitting strongly. To allow individual samples outside the otherwise hard ϵ bounds, non-negative slack variables ξ_i and ξ_i^* are included. A non-negative parameter C regulates how many samples may violate the ϵ bounds and by how much. To model non-linear relationships, one could use a mapping $\Phi(\cdot)$ for the $\vec{x}_i$ from the row space of $\boldsymbol{X}$ to some higher dimensional space; however, as the optimization problem only depends on the dot product $\langle \cdot, \cdot \rangle$ and not the actual entries of $\vec{x}_i$, it suffices to use a kernel function k such that $k(\vec{x}_i, \vec{x}_j) = \langle \Phi(\vec{x}_i), \Phi(\vec{x}_j) \rangle$. Such kernels must fulfill certain mathematical properties, and, besides polynomial kernels, radial basis functions with $k(\vec{x}_i, \vec{x}_j) = exp(\gamma \left\| \vec{x}_i - \vec{x}_j \right\|^2)$ are a popular candidate where γ is a parameter controlling for how the distances between any two samples influence the final model. SVRs work well with sparse data in high dimensional spaces, such as intermittent demand data, as they minimize the risk of misclassification or predicting a significantly far off value by maximizing the error margin, as also noted by [3].

3. Model Formulation

In this section, we describe how the platform's raw data are pre-processed into model inputs and how the forecasting models are built and benchmarked against each other.

3.1. Overall Approach

On a conceptual level, there are three distinct aspects of the model development process. First, a pre-processing step transforms the platform's tabular order data into either time series in Sub-section 3.2 or feature matrices in Sub-section 3.6.4. Second, a benchmark methodology is developed in Sub-section 3.3 that compares all models on the same scale, in particular, classical models with ML ones. Concretely, the CV approach is adapted to the peculiar requirements of sub-daily and ordinal time series data. This is done to maximize the predictive power of all models into the future and to compare them on the same scale. Third, the forecasting models are described with respect to their assumptions and training requirements. Four classification dimensions are introduced:

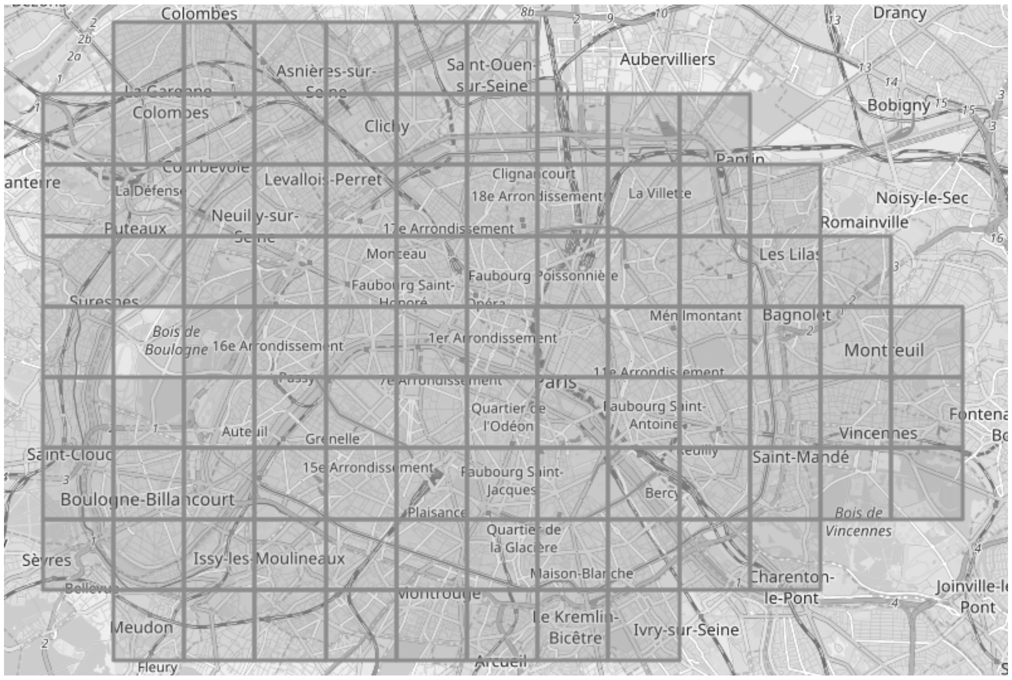
1. **Timeliness of the Information**: whole-day-ahead vs. real-time forecasts
2. **Time Series Decomposition**: raw vs. decomposed
3. **Algorithm Type**: "classical" statistics vs. ML
4. **Data Sources**: pure vs. enhanced (i.e., with external data)

Not all of the possible eight combinations are implemented; instead, the models are varied along these dimensions to show different effects and answer the research questions.

3.2. Gridification, Time Tables, and Time Series Generation

The platform's tabular order data are sliced with respect to both location and time and then aggregated into time series where an observation tells the number of orders in an area for a time step/interval. Figure 1 shows how the orders' delivery locations are each matched to a square-shaped cell, referred to as a pixel, on a grid covering the entire service area within a city. This gridification step is also applied to the pickup locations separately. The lower-left corner is chosen at random. [57] apply the same gridification idea and slice an urban area to model a location-routing problem, and [48] portray it as a standard method in the field of urban analytics. With increasing pixel sizes, the time series exhibit more order aggregation with a possibly stronger demand pattern. On the other hand, the larger the pixels, the less valuable become the generated forecasts as, for example, a courier sent to a pixel preemptively then faces a longer average distance to a restaurant in the pixel.

Figure 1: Gridification for delivery locations in Paris with a pixel size of 1 km^2



After gridification, the ad-hoc orders within a pixel are aggregated by their placement timestamps into sub-daily time steps of pre-defined lengths to obtain a time table as exemplified in Figure 2 with one-hour intervals.

Figure 2: Aggregation into a time table with hourly time steps

Time \ Day	...	Mon	Tue	Wed	Thu	Fri	Sat	Sun	...
11:00	...	$y_{11,Mon}$	$y_{11,Tue}$	$y_{11,Wed}$	$y_{11,Thu}$	$y_{11,Fri}$	$y_{11,Sat}$	$y_{11,Sun}$	...
12:00	...	$y_{12,Mon}$	$y_{12,Tue}$	$y_{12,Wed}$	$y_{12,Thu}$	$y_{12,Fri}$	$y_{12,Sat}$	$y_{12,Sun}$	...
...	...	...	...	...	...	...	...	...	...
20:00	...	$y_{20,Mon}$	$y_{20,Tue}$	$y_{20,Wed}$	$y_{20,Thu}$	$y_{20,Fri}$	$y_{20,Sat}$	$y_{20,Sun}$	...
21:00	...	$y_{21,Mon}$	$y_{21,Tue}$	$y_{21,Wed}$	$y_{21,Thu}$	$y_{21,Fri}$	$y_{21,Sat}$	$y_{21,Sun}$	...
...	...	...	...	...	...	...	...	...	...

Consequently, each $y_{t,d}$ in Figure 2 is the number of all orders within the pixel for the time of day t and day of week d (y_t and $y_{t,d}$ are the same but differ in that the latter acknowledges a 2D view). The same trade-off as with gridification applies: The shorter the interval, the weaker is the demand pattern to be expected in the time series due to less aggregation while longer intervals lead to less usable forecasts. We refer to time steps by their start time, and their number per day, H, is constant. Given a time table as in Figure 2 there are two ways to generate a time series by slicing:

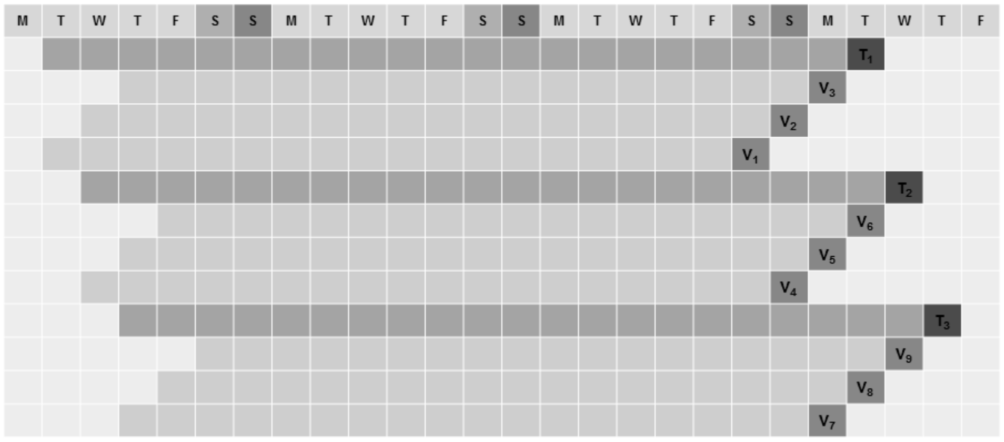
1. **Horizontal View**: Take only the order counts for a given time of the day
2. **Vertical View**: Take all order counts and remove the double-seasonal pattern induced by the weekday and time of the day with decomposition

Distinct time series are retrieved by iterating through the time tables either horizontally or vertically in increments of a single time step. Another property of a generated time series is its length, which, following the next sub-section, can be interpreted as the sum of the production training set and the test day. In summary, a distinct time series is generated from the tabular order data based on a configuration of parameters for the dimensions pixel size, number of daily time steps H, shape (horizontal vs. vertical), length, and the time step to be predicted.

3.3. Unified Cross-Validation and Training, Validation, and Test Sets

The standard k-fold CV, which assumes no structure in the individual features of the samples, as shown in $\boldsymbol{X}$ above, is adapted to the ordinal character of time series data: A model must be evaluated on observations that occurred strictly after the ones used for training as, otherwise, the model knows about the future. Furthermore, some models predict only a single to a few time steps before being retrained, while others predict an entire day without retraining (cf., Sub-section 3.6.4). Consequently, we must use a unified time interval wherein all forecasts are made first before the entire interval is evaluated. As whole days are the longest prediction interval for models without retraining, we choose that as the unified time interval. In summary, our CV methodology yields a distinct best model per pixel and day to be forecast. Whole days are also practical for managers who commonly monitor, for example, the routing and thus the forecasting performance on a day-to-day basis. Our methodology assumes that the models are trained at least once per day. As we create operational forecasts into the near future in this paper, retraining all models with the latest available data is a logical step.

Figure 3: Training, validation, and test sets during cross validation



The training, validation, and test sets are defined as follows. To exemplify the logic, we refer to Figure 3 that shows the calendar setup (i.e., weekdays on the x-axis) for three days T_1, T_2, and T_3 (shown in dark gray) for which we generate forecasts. Each of these days is, by definition, a test day, and the test set comprises all time series, horizontal or vertical, whose last observation lies on that day. With an assumed training horizon of three weeks, the 21 days before each of the test days constitute the corresponding training sets (shown in lighter gray on the same rows as T_1, T_2, and T_3). There are two kinds of validation sets, depending on the decision to be made. First, if a forecasting method needs parameter tuning, the original training set is divided into as many equally long series as validation days are needed to find stable parameters. The example shows three validation days per test day named V_n (shown in darker gray below each test day). The $21 - 3 = 18$ preceding days constitute the training set corresponding to a validation day. To obtain the overall validation error, the three errors are averaged. We call these *inner* validation sets because they must be repeated each day to re-tune the parameters and because the involved time series are true subsets of the original series. Second, to find the best method per day and pixel, the same averaging logic is applied on the outer level. For example, if we used two validation days to find the best method for T_3, we would average the errors of T_1 and T_2 for each method and select the winner; then, T_1 and T_2 constitute an *outer* validation set. Whereas the number of inner validation days is method-specific and must be chosen before generating any test day forecasts in the first place, the number of outer validation days may be varied after the fact and is determined empirically as we show in Section 4.

Our unified CV approach is also optimized for large-scale production settings, for example, at companies like Uber. As [5] note, there is a trade-off as to when each of the inner time series in the example begins. While the forecasting accuracy likely increases with more training days, supporting inner series with increasing lengths, cutting the series to the same length allows caching the forecasts and errors. In the example, V_3, V_5, and V_7, as well as V_6 and V_8 are identical despite belonging to different inner validation sets. Caching is also possible on the outer level when searching for an optimal number of validation days for model selection. We achieved up to 80% cache hit ratios in our implementation in the empirical study, thereby saving computational resources by the same amount. Lastly, we assert that our suggested CV, because of its being unified around whole test days and usage of fix-sized time series, is also suitable for creating consistent learning curves and, thus, answering **Q3** on the relationship between forecast accuracy and amount of historic data: We simply increase the length of the outer training set holding the test day fixed. Thus, independent of a method's need for parameter tuning, all methods have the same demand history available for each test day forecast.

3.4. Accuracy Measures

Choosing an error measure for both model selection and evaluation is not straightforward when working with intermittent demand, as shown, for example, by [51], and one should understand the trade-offs between measures. [33] provide a study of measures with real-life data taken from the popular M3-competition and find that most standard measures degenerate under many scenarios. They also provide a classification scheme for which we summarize the main points as they apply to the UDP case:

1. **Scale-dependent Errors**: The error is reported in the same unit as the raw data. Two popular examples are the root mean square error (RMSE) and mean absolute error (MAE). They may be used for model selection and evaluation within a pixel, and are intuitively interpretable; however, they may not be used to compare errors of,

for example, a low-demand pixel (e.g., at the UDP's service boundary) with that of a high-demand pixel (e.g., downtown).

2. **Percentage Errors**: The error is derived from the percentage errors of individual forecasts per time step, and is also intuitively interpretable. A popular example is the mean absolute percentage error (MAPE) that is the primary measure in most forecasting competitions. Whereas such errors could be applied both within and across pixels, they cannot be calculated reliably for intermittent demand. If only one time step exhibits no demand, the result is a divide-by-zero error. This often occurs even in high-demand pixels due to the slicing.
3. **Relative Errors**: A workaround is to calculate a scale-dependent error for the test day and divide it by the same measure calculated with forecasts of a simple benchmark method (e.g., naïve method). An example could be RelMAE = MAE/MAE_{bm}. Nevertheless, even simple methods create (near-)perfect forecasts, and then MAE_{bm} becomes (close to) 0. These numerical instabilities occurred so often in our studies that we argue against using such measures.
4. **Scaled Errors**: [33] contribute this category and introduce the mean absolute scaled error (MASE). It is defined as the MAE from the actual forecasting method on the test day (i.e., "out-of-sample") divided by the MAE from the (seasonal) naïve method on the entire training set (i.e., "in-sample"). A MASE of 1 indicates that a forecasting method has the same accuracy on the test day as the (seasonal) naïve method applied on a longer horizon, and lower values imply higher accuracy. Within a pixel, its results are identical to the ones obtained with MAE. Also, we acknowledge recent publications, for example, [46] or [36], showing other ways of tackling the difficulties mentioned. However, only the MASE provided numerically stable results for all forecasts in our study.

Consequently, we use the MASE with a seasonal naïve benchmark as the primary measure in this paper. With the previously introduced notation, it is defined as follows:

$$\text{MASE} := \frac{\text{MAE}_{\text{out-of-sample}}}{\text{MAE}_{\text{in-sample}}} = \frac{\text{MAE}_{\text{forecasts}}}{\text{MAE}_{\text{training}}} = \frac{\frac{1}{H}\sum_{h=1}^{H}|y_{T+h} - \hat{y}_{T+h}|}{\frac{1}{T-k}\sum_{t=k+1}^{T}|y_t - y_{t-k}|}$$

The denominator can only become 0 if the seasonal naïve benchmark makes a perfect forecast on each day in the training set except the first seven days, which never happened in our case study involving hundreds of thousands of individual model trainings. Further, as per the discussion in the subsequent Section 3.5, we also calculate peak-MASEs where we leave out the time steps of non-peak times from the calculations. For this analysis, we define all time steps that occur at lunch (i.e., noon to 2 pm) and dinner time (i.e., 6 pm to 8 pm) as peak. As time steps in non-peak times typically average no or very low order counts, a UDP may choose to not actively forecast these at all and be rather interested in the accuracies of forecasting methods during peaks only.

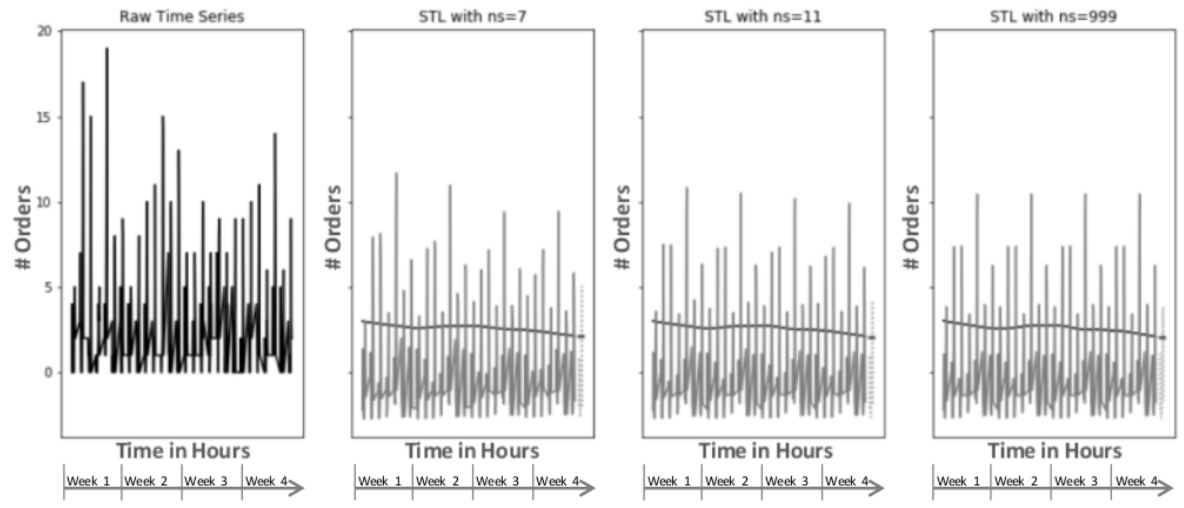
We conjecture that percentage error measures may be usable for UDPs facing a higher overall demand with no intra-day down-times in between but have to leave that to a future study. Yet, even with high and steady demand, divide-by-zero errors are likely to occur.

3.5. Time Series Decomposition

Concerning the time table in Figure 2, a seasonal demand pattern is inherent to both horizontal and vertical time series. First, the weekday influences if people eat out or order in with our partner receiving more orders on Thursday through Saturday than the other four days. This pattern is part of both types of time series. Second, on any given day, demand peaks occur around lunch and dinner times. This only regards vertical series. Statistical analyses show that horizontally sliced time series indeed exhibit a periodicity of $k = 7$, and vertically sliced series only yield a seasonal component with a regular pattern if the periodicity is set to the product of the number of weekdays and the daily time steps indicating a distinct intra-day pattern per weekday.

Figure 4 shows three exemplary STL decompositions for a 1 km^2 pixel and a vertical time series with 60-minute time steps (on the x-axis) covering four weeks: With the noisy raw data y_t on the left, the seasonal and trend components, s_t and t_t, are depicted in light and dark gray for increasing ns parameters. The plots include (seasonal) naïve forecasts for the subsequent test day as dotted lines. The remainder components r_t are not shown for conciseness. The periodicity is set to $k = 7 * 12 = 84$ as our industry partner has 12 opening hours per day.

Figure 4: STL decompositions for a medium-demand pixel with hourly time steps and periodicity $k = 84$



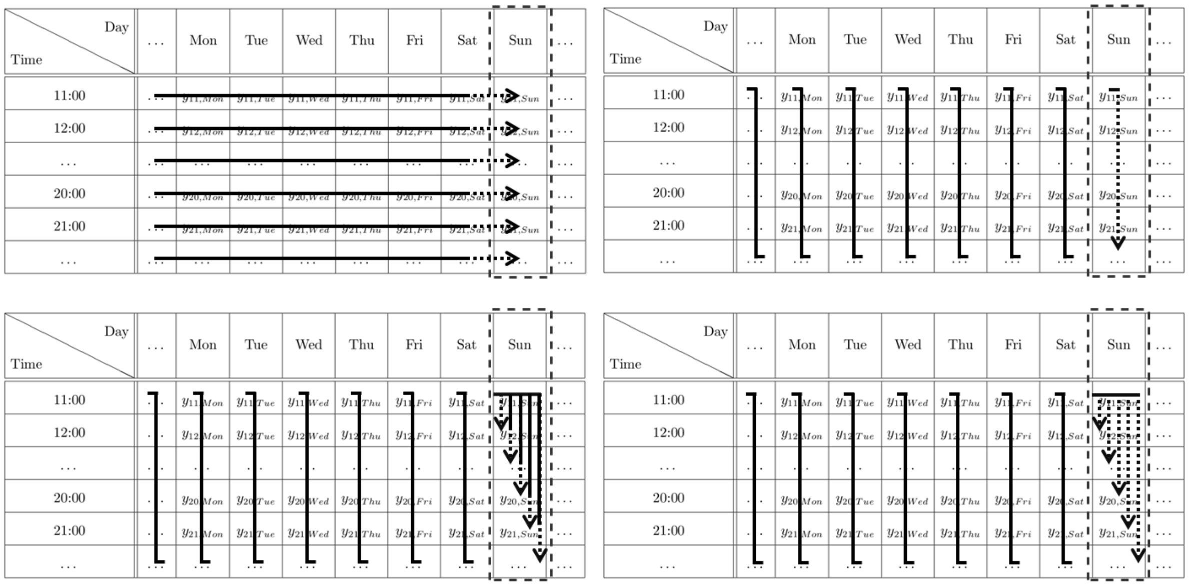
As described in Sub-section 2.1.3, with k being implied by the application, at the very least, the length of the seasonal smoothing window, represented by the ns parameter, must be calibrated by the forecaster: It controls how many past observations go into each smoothened s_t. Many practitioners, however, skip this step and set ns to a big number, for example, 999, then referred to as "periodic." For the other parameters, it is common to use the default values as specified in [16]. The goal is to find a decomposition with a regular pattern in s_t. In Figure 4, this is not true for $ns = 7$ where, for example, the four largest bars corresponding to the same time of day a week apart cannot be connected by an approximately straight line. On the contrary, a regular pattern in the most extreme way exists for $ns = 999$, where the same four largest bars are of the same height. This observation holds for each time step of the day. For $ns = 11$, s_t exhibits a regular pattern whose bars adapt over time: The pattern is regular as bars corresponding to the same time of day can be connected by approximately straight lines, and it is adaptive as these lines are not horizontal. The trade-off between small and large values for ns can thus be interpreted as allowing the average demand during peak times to change over time: If demand is intermittent at non-peak times, it is reasonable to expect the bars to change over time as only the relative differences between peak and non-peak times impact the bars' heights with the seasonal component being centered around 0. To confirm the goodness of a decomposition statistically, one way is to verify that r_t can be modeled as a typical error process like white noise ϵ_t.

However, we suggest an alternative way of calibrating the STL method in an automated fashion based on our unified CV approach. As hinted at in Figure 4, we interpret an STL decomposition as a forecasting method on its own by just adding the (seasonal) naïve forecasts for s_t and t_t and predicting 0 for r_t. Then, the ns parameter is tuned just like a parameter for an ML model. To the best of our knowledge, this has not yet been proposed before. Conceptually, forecasting with the STL method can be viewed as a naïve method with built-in smoothing, and it outperformed all other benchmark methods in all cases.

3.6. Forecasting Models

This sub-section describes the concrete models in our study. Figure 5 shows how we classify them into four families with regard to the type of the time series, horizontal or vertical, and the moment at which a model is trained: Solid lines indicate that the corresponding time steps lie before the training, and dotted lines show the time horizon predicted by a model. For conciseness, we only show the forecasts for one test day. The setup is the same for each inner validation day.

Figure 5: Classification of the models by input type and training moment



3.6.1. Horizontal and Whole-day-ahead Forecasts.

The upper-left in Figure 5 illustrates the simplest way to generate forecasts for a test day before it has started: For each time of the day, the corresponding horizontal slice becomes the input for a model. With whole days being the unified time interval, each model is trained H times, providing a one-step-ahead forecast. While it is possible to have models of a different type be selected per time step, that did not improve the accuracy in the empirical study. As the models in this family do not include the test day's demand data in their training sets, we see them as benchmarks to answer **Q4**, checking if a UDP can take advantage of real-time information. The models in this family are as follows; we use prefixes, such as "h" here, when methods are applied in other families as well:

1. *naive*: Observation from the same time step one week prior
2. *trivial*: Predict 0 for all time steps
3. *hcroston*: Intermittent demand method introduced by [17]
4. *hholt*, *hhwinters*, *hses*, *hsma*, and *htheta*: Exponential smoothing without calibration
5. *hets*: ETS calibrated as described by [34]
6. *harima*: ARIMA calibrated as described by [32]

naive and *trivial* provide an absolute benchmark for the actual forecasting methods. *hcroston* is often mentioned in the context of intermittent demand; however, the method did not perform well at all. Besides *hhwinters* that always fits a seasonal component, the calibration heuristics behind *hets* and *harima* may do so as well. With $k = 7$, an STL decomposition is unnecessary here.

3.6.2. Vertical and Whole-day-ahead Forecasts without Retraining.

The upper-right in Figure 5 shows an alternative way to generate forecasts for a test day before it has started: First, a seasonally-adjusted time series a_t is obtained from a vertical time series by STL decomposition. Then, the actual forecasting model, trained on a_t, makes an H-step-ahead prediction. Lastly, we add the H seasonal naïve forecasts for the seasonal component s_t to them to obtain the actual predictions for the test day. Thus, only one training is required per model type, and no real-time data is used. By decomposing the raw time series, all long-term patterns are assumed to be in the seasonal component s_t, and

a_t only contains the level with a potential trend and auto-correlations. The models in this family are:

1. *fnaive*, *pnaive*: Sum of STL's trend and seasonal components' naïve forecasts
2. *vholt*, *vses*, and *vtheta*: Exponential smoothing without calibration and seasonal fit
3. *vets*: ETS calibrated as described by [34]
4. *varima*: ARIMA calibrated as described by [32]

As mentioned in Sub-section 3.3, we include the sum of the (seasonal) naïve forecasts of the STL's trend and seasonal components as forecasts on their own: For *fnaive*, we tune the "flexible" ns parameter, and for *pnaive*, we set it to a "periodic" value. Thus, we implicitly assume that there is no signal in the remainder r_t, and predict 0 for it. *fnaive* and *pnaive* are two more simple benchmarks.

3.6.3. Vertical and Real-time Forecasts with Retraining.

The lower-left in Figure 5 shows how models trained on vertical time series are extended with real-time order data as it becomes available during a test day: Instead of obtaining an H-step-ahead forecast, we retrain a model after every time step and only predict one step. The remainder is as in the previous sub-section, and the models are:

1. *rtholt*, *rtses*, and *rttheta*: Exponential smoothing without calibration and seasonal fit
2. *rtets*: ETS calibrated as described by [34]
3. *rtarima*: ARIMA calibrated as described by [32]

Retraining *fnaive* and *pnaive* did not increase accuracy, and thus we left them out. A downside of this family is the significant increase in computing costs.

3.6.4. Vertical and Real-time Forecasts without Retraining.

The lower-right in Figure 5 shows how ML models take real-time order data into account without retraining. Based on the seasonally-adjusted time series a_t, we employ the feature matrix and label vector representations from Sub-section 2.2.1 and set n to the number of daily time steps, H, to cover all potential auto-correlations. The ML models are trained once before a test day starts. For training, the matrix and vector are populated such that y_T is set to the last time step of the day before the forecasts, a_T. As the splitting during CV is done with whole days, the ML models are trained with training sets consisting of samples from all times of a day in an equal manner. Thus, the ML models learn to predict each time of the day. For prediction on a test day, the H observations preceding the time step to be forecast are used as the input vector after seasonal adjustment. As a result, real-time data are included. The models in this family are:

1. *vrfr*: RF trained on the matrix as described
2. *vsvr*: SVR trained on the matrix as described

We tried other ML models such as gradient boosting machines but found only RFs and SVRs to perform well in our study. In the case of gradient boosting machines, this is to be expected as they are known not to perform well in the presence of high noise - as is natural with low count data - as shown, for example, by [39] or [41]. Also, deep learning methods are not applicable as the feature matrices only consist of several hundred to thousands of rows (cf., Sub-section 4.2). In Appendix A, we provide an alternative feature matrix representation that exploits the two-dimensional structure of time tables without decomposing the time series. In Appendix B, we show how feature matrices are extended to include predictors other than historical order data. However, to answer **Q5** already here, none of the external data sources improves the results in our study. Due to the high number of time series in our study, to investigate why no external sources improve the forecasts, we must us some automated approach to analyzing individual time series. [4] provide a spectral density estimation approach, called the Shannon entropy, that measures the signal-to-noise ratio in a database with a number normalized between 0 and 1 where lower values indicate a higher signal-to-noise ratio. We then looked at averages of the estimates on a daily level per pixel and find that including any of the external data sources from Appendix B always leads to significantly lower signal-to-noise ratios. Thus, we conclude that at least for the demand faced by our industry partner the historical data contains all of the signal.

4. Empirical Study: A Meal Delivery Platform in Europe

In the following, we first give a brief overview of the case study dataset and the parameters we applied to calibrate the time series generation. Then, we discuss the overall results.

4.1. Case Study Dataset

The studied dataset consists of a meal delivery platform's entire transactional data covering the French market from launch in February of 2016 to January of 2017. The platform operated in five cities throughout this period and received a total of 686,385 orders. The forecasting models were developed based on the data from Lyon and Paris in the period from August through December; this ensures comparability across cities and avoids irregularities in demand assumed for a new service within the first operating weeks. The data exhibit a steady-state as the UDP's service area remained unchanged, and the numbers of orders and of couriers grew in lock-step and organically. This does not mean that no new restaurants were openend: If that happened, the new restaurant did not attract new customers, but demand was shifted from other member restaurants. Results are similar in both cities, and we only report them for Paris for greater conciseness. Lastly, the platform recorded all incoming orders, and lost demand does not exist. See Appendix C for details on the raw data.

4.2. Calibration of the Time Series Generation Process

Independent of the concrete forecasting models, the time series generation must be calibrated. We concentrate our forecasts on the pickup side for two reasons. First, the restaurants come in a significantly lower number than the customers resulting in more aggregation in the order counts and thus a better pattern recognition. Second, from an operational point of view, forecasts for the pickups are more valuable because of the waiting times due to meal preparation. We choose pixel sizes of 0.5 km^2, 1 km^2, 2 km^2, and 4 km^2, and time steps covering 60, 90, and 120 minute windows resulting in $H_{60} = 12$, $H_{90} = 9$, and $H_{120} = 6$ time steps per day with the platform operating between 11 a.m. and 11 p.m. and corresponding frequencies $k_{60} = 7 * 12 = 84$, $k_{90} = 7 * 9 = 63$, and $k_{120} = 7 * 6 = 42$ for the vertical time series. Smaller pixels and shorter time steps yield no recognizable patterns, yet would have been more beneficial for tactical routing. 90 and 120 minute time steps are most likely not desirable for routing; however, we keep them for comparison and note that a UDP may employ such forecasts to activate more couriers at short notice if a (too) high demand is forecasted in an hour from now. This could, for example, be implemented by paying couriers a premium if they show up for work at short notice. Discrete lengths of 3, 4, 5, 6, 7, and 8 weeks are chosen as training horizons. We do so as the structure within the pixels (i.e., number and kind of restaurants) is not stable for more than two months in a row in the covered horizon. That is confirmed by the empirical finding that forecasting accuracy improves with longer training horizon but this effect starts to level off after about six to seven weeks. So, the demand patterns of more than two months ago do not resemble more recent ones.

In total, 100,000s of distinct time series are forecast in the study.

4.3. Overall Results

Table 1 summarizes the overall best-performing models grouped by training horizon and a pixel's average daily demand (ADD) for a pixel size of 1 km^2 and 60-minute time steps. Each combination of pixel and test day counts as one case, and the total number of cases is denoted as n. Clustering the individual results revealed that a pixel's ADD over the training horizon is the primary indicator of similarity and three to four clusters suffice to obtain cohesive clusters: We labeled them "no", "low", "medium", and "high" demand pixels with increasing ADD, and present the average MASE per cluster. The n do not vary significantly across the training horizons, which confirms that the platform did not grow area-wise and is indeed in a steady-state. We use this table to answer **Q1** regarding the overall best methods under different ADDs. All result tables in the main text report MASEs calculated with all time steps of a day. In contrast, Appendix D shows the same tables with MASEs calculated with time steps within peak times only (i.e., lunch from 12 pm to 2 pm and dinner from 6 pm to 8 pm). The differences lie mainly in the decimals of the individual MASE averages while the ranks of the forecasting methods do not change except in rare cases. That shows that the presented accuracies are driven by the forecasting methods' accuracies at peak times. Intuitively, they all correctly predict zero demand for non-peak times.

Unsurprisingly, the best model for pixels without demand (i.e., $0 < \text{ADD} < 2.5$) is *trivial*. Whereas *hsma* also adapts well, its performance is worse. None of the more sophisticated models reaches a similar accuracy. The intuition behind is that *trivial* is the least distorted by the relatively large proportion of noise given the low-count nature of the time series.

For low demand (i.e., $2.5 < \text{ADD} < 10$), there is also a clear best-performing model, namely *hsma*. As the non-seasonal *hses* reaches a similar accuracy as its potentially seasonal generalization, the *hets*, we conclude that the seasonal pattern from weekdays is not yet strong enough to be recognized in low demand pixels. So, in the absence of seasonality, models that only model a trend part are the least susceptible to the noise.

For medium demand (i.e., $10 < \text{ADD} < 25$) and training horizons up to six weeks, the best-performing models are the same as for low demand. For longer horizons, *hets* provides the highest accuracy. Thus, to fit a seasonal pattern, longer training horizons are needed. While *vsvr* enters the top three, *hets* has the edge as they neither require parameter tuning nor real-time data.

Table 1: Top-3 models by training weeks and average demand (1 km^2 pixel size, 60-minute time steps)

Training	**Rank**	**No Demand** (0 - 2.5)			**Low Demand** (2.5 - 10)			**Medium Demand** (10 - 25)			**High Demand** (25 - ∞)		
		Method	MASE	n	Method	MASE	n	Method	MASE	n	Method	MASE	n
3	1	***trivial***	0.785	4586	***hsma***	0.819	2975	***hsma***	0.839	2743	***rtarima***	0.872	2018
	2	*hsma*	0.809		*hses*	0.844		*hses*	0.858		*rtses*	0.873	
	3	*pnaive*	0.958		*hets*	0.846		*hets*	0.859		*rtets*	0.877	
4	1	***trivial***	0.770	4532	***hsma***	0.825	3033	***hsma***	0.837	2687	***vrfr***	0.855	2016
	2	*hsma*	0.788		*hses*	0.848		*hses*	0.850		***rtarima***	0.855	
	3	*pnaive*	0.917		*hets*	0.851		*hets*	0.854		*rtses*	0.860	
5	1	***trivial***	0.780	4527	***hsma***	0.841	3055	***hsma***	0.837	2662	***vrfr***	0.850	2019
	2	*hsma*	0.803		*hses*	0.859		*hets*	0.845		***rtarima***	0.852	
	3	*pnaive*	0.889		*hets*	0.861		*hses*	0.845		*vsvr*	0.854	
6	1	***trivial***	0.741	4470	***hsma***	0.847	3086	***hsma***	0.840	2625	***vrfr***	0.842	2025
	2	*hsma*	0.766		*hses*	0.863		*hets*	0.842		***hets***	0.847	
	3	*pnaive*	0.837		*hets*	0.865		*hses*	0.848		*vsvr*	0.848	
7	1	***trivial***	0.730	4454	***hsma***	0.858	3132	***hets***	0.845	2597	***hets***	0.840	2007
	2	*hsma*	0.754		*hses*	0.871		*hsma*	0.847		***vrfr***	0.845	
	3	*pnaive*	0.813		*hets*	0.872		***vsvr***	0.850		*vsvr*	0.847	
8	1	***trivial***	0.735	4402	***hsma***	0.867	3159	***hets***	0.846	2575	***hets***	0.836	2002
	2	*hsma*	0.758		*hets*	0.877		***vsvr***	0.850		***vrfr***	0.842	
	3	*pnaive*	0.811		*hses*	0.880		*hsma*	0.851		*vsvr*	0.849	

In summary, except for high demand, simple models trained on horizontal time series work best. By contrast, high demand (i.e., $25 < \text{ADD} < \infty$) and less than six training weeks is the only situation where classical models trained on vertical time series work well. Then, *rtarima* outperforms their siblings from Sub-sections 3.6.2 and 3.6.3. We conjecture that intra-day auto-correlations as caused, for example, by weather, are the reason for that. Intuitively, a certain amount of demand (i.e., a high enough signal-to-noise ratio) is required such that models with auto-correlations can see them through all the noise. That idea is supported by *vrfr* reaching a similar accuracy under high demand as their tree-structure allows them to fit auto-correlations. As both *rtarima* and *vrfr* incorporate recent demand, real-time information can indeed improve accuracy. However, once models are trained on longer horizons, *hets* is more accurate than *vrfr*. Thus, to answer **Q4**, we conclude that real-time information only improves accuracy if three or four weeks of training material are available.

In addition to looking at the results in tables covering the entire one-year horizon, we also created sub-analyses on the distinct seasons spring, summer (incl. the long holiday season in France), and fall. Yet, none of the results portrayed in this and the subsequent sections change is significant ways. We conjecture that there could be differences if the overall demand of the UDP increased to a scale beyond the one this case study covers and leave that up to a follow-up study with a bigger UDP.

4.4. Impact of the Training Horizon

Whereas it is reasonable to assume that forecasts become more accurate as the training horizon expands, our study reveals some interesting findings. First, without demand, *trivial* indeed performs better with more training material, but improved pattern recognition cannot be the cause here. Instead, we argue that the reason for this is that the longer there has been no steady demand, the higher the chance that this will not change soon. Further, if we focus on shorter training horizons, the sample will necessarily contain cases where a pixel is initiated after a popular-to-be restaurant joined the platform: Demand grows fast making *trivial* less accurate, and the pixel moves to another cluster soon.

Second, with low demand, the best-performing *hsma* becomes less accurate with more training material. While one could argue that this is due to *hsma* not fitting a trend, the less accurate *hses* and *hets* do fit a trend. Instead, we argue that any low-demand time series naturally exhibits a high noise-to-signal ratio, and *hsma* is the least susceptible to noise. Then, to counter the missing trend term, the training horizon must be shorter.

With medium demand, a similar argument can be made; however, the signal already becomes more apparent favoring *hets* with more training data.

Lastly, with high demand, the signal becomes so clear that more sophisticated models can exploit longer training horizons.

4.5. Results by Model Families

Besides the overall results, we provide an in-depth comparison of models within a family. Instead of reporting the MASE per model, we rank the models holding the training horizon fixed to make comparison easier. Table 2 presents the models trained on horizontal time series. In addition to *naive*, we include *fnaive* and *pnaive* already here as more competitive benchmarks. The tables in this section report two rankings simultaneously: The first number is the rank resulting from lumping the low and medium clusters together, which yields almost the same rankings when analyzed individually. The ranks from only high demand pixels are in parentheses if they differ.

Table 2: Ranking of benchmark and horizontal models (1 km^2 pixel size, 60-minute time steps): the table shows the ranks for cases with $2.5 < ADD < 25$ (and $25 < ADD < \infty$ in parentheses if they differ)

Training	**Benchmarks**			**Horizontal (whole-day-ahead)**							
	naive	*fnaive*	*paive*	*harima*	*hcroston*	*hets*	*hholt*	*hhwinters*	*hses*	*hsma*	*htheta*
3	11	7 (2)	8 (5)	5 (7)	4	3	9 (10)	10 (9)	2 (6)	1	6 (8)
4	11	7 (2)	8 (3)	5 (6)	4 (5)	3 (1)	9 (10)	10 (9)	2 (7)	1 (4)	6 (8)
5	11	7 (2)	8 (4)	5 (3)	4 (9)	3 (1)	9 (10)	10 (5)	2 (8)	1 (6)	6 (7)
6	11	8 (5)	9 (6)	5 (4)	4 (7)	2 (1)	10	7 (2)	3 (8)	1 (9)	6 (3)
7	11	8 (5)	10 (6)	5 (4)	4 (7)	2 (1)	9 (10)	7 (2)	3 (8)	1 (9)	6 (3)
8	11	9 (5)	10 (6)	5 (4)	4 (7)	2 (1)	8 (10)	7 (2)	3 (8)	1 (9)	6 (3)

A first insight is that *fnaive* is the best benchmark in all scenarios: Decomposing flexibly by tuning the *ns* parameter is worth the computational cost. Further, if one is limited in the number of non-naïve methods, *hets* is the best compromise and works well across all demand levels. It is also the best model independent of the training horizon for high demand. With low or medium demand, *hsma* is the clear overall winner; yet, with high demand, models with a seasonal fit (i.e., *harima*, *hets*, and *hhwinters*) are more accurate, in particular, for longer training horizons. This is due to demand patterns in the weekdays becoming stronger with higher overall demand.

Table 3: Ranking of classical models on vertical time series (1 km^2 pixel size, 60-minute time steps): the table shows the ranks for cases with $2.5 < ADD < 25$ (and $25 < ADD < \infty$ in parentheses if they differ)

Training	**Benchmarks**		**Vertical (whole-day-ahead)**					**Vertical (real-time)**				
	hets	*hsma*	*varima*	*vets*	*vholt*	*vses*	*vtheta*	*rtarima*	*rtets*	*rtholt*	*rtses*	*rttheta*
3	2 (10)	1 (7)	6 (4)	8 (6)	10 (9)	7 (5)	11 (12)	4 (1)	5 (3)	9 (8)	3 (2)	12 (11)
4	2 (8)	1 (10)	6 (4)	8 (6)	10 (9)	7 (5)	12 (11)	3 (1)	5 (3)	9 (7)	4 (2)	11 (12)
5	2 (3)	1 (10)	7 (5)	8 (7)	10 (9)	6	11	4 (1)	5 (4)	9 (8)	3 (2)	12
6	2 (1)	1 (10)	6 (5)	8 (7)	10 (9)	7 (6)	11 (12)	3 (2)	5 (4)	9 (8)	4 (3)	12 (11)
7	2 (1)	1 (10)	8 (5)	7	10 (9)	6	11 (12)	5 (2)	4	9 (8)	3	12 (11)
8	2 (1)	1 (9)	8 (5)	7 (6)	10 (8)	6	12 (10)	5 (2)	4	9 (7)	3	11

Table 3 extends the previous analysis to classical models trained on vertical time series. Now, the winners from before, *hets* and *hsma*, serve as benchmarks. Whereas for low and medium demand, no improvements can be obtained, *rtarima* and *rtses* are the most accurate with high demand and short training horizons. For six or more training weeks, *hets* is still optimal. Independent of retraining and the demand level, the models' relative performances are consistent: The **arima* and **ses* models are best, followed by **ets*, **holt*, and **theta*. Thus, models that can deal with auto-correlations and short-term forecasting errors, as expressed by moving averages, and that cannot be distracted by trend terms are optimal for vertical series.

Finally, Table 4 compares the two ML-based models against the best-performing classical models and answers **Q2**: With low and medium demand, no improvements can be obtained again; however, with high demand, *vrfr* has the edge over *rtarima* for training horizons up to six weeks. We conjecture that *vrfr* fits auto-correlations better than *varima* and is not distracted by short-term noise as *rtarima* may be due to the retraining. With seven or eight training weeks, *hets* remains the overall winner. Interestingly, *vsvr* is more accurate than *vrfr* for low and medium demand. We assume that *vrfr* performs well only with strong auto-correlations, which are not present with low and medium demand.

Table 4: Ranking of ML models on vertical time series (1 km^2 pixel size, 60-minute time steps): the table shows the ranks for cases with $2.5 < ADD < 25$ (and $25 < ADD < \infty$ in parentheses if they differ)

Training	**Benchmarks**				**ML**	
	fnaive	*hets*	*hsma*	*rtarima*	*vrfr*	*vsvr*
3	6	2 (5)	1 (4)	3 (1)	5 (2)	4 (3)
4	6 (5)	2 (4)	1 (6)	3 (2)	5 (1)	4 (3)
5	6 (5)	2 (4)	1 (6)	4 (2)	5 (1)	3
6	6 (5)	2	1 (6)	4	5 (1)	3
7	6 (5)	2 (1)	1 (6)	4	5 (2)	3
8	6 (5)	2 (1)	1 (6)	4	5 (2)	3

Analogously, we created tables like Table 2 to 4 for the forecasts with time steps of 90 and 120 minutes and find that the relative rankings do not change significantly. The same holds true for the rankings with changing pixel sizes. For conciseness reasons, we do not include these additional tables in this article. In summary, the relative performances exhibited by certain model families are shown to be rather stable in this case study.

4.6. Effects of the Pixel Size and Time Step Length

As elaborated in Sub-section 3.2, more order aggregation leads to a higher overall demand level and an improved pattern recognition in the generated time series. Consequently, individual cases tend to move to the right in tables equivalent to Table 1. With the same ADD clusters, forecasts for pixel sizes of 2 km^2 and 4 km^2 or time intervals of 90 and 120 minutes or combinations thereof yield results similar to the best models as revealed in Tables 1, 2, 3, and 4 for high demand. By contrast, forecasts for 0.5 km^2 pixels have most of the cases (i.e., n) in the no or low demand clusters. In that case, the pixels are too small, and pattern recognition becomes harder. While it is true, that *trivial* exhibits the overall lowest MASE for no demand cases, these forecasts become effectively worthless for operations. In the extreme, with even smaller pixels we would be forecasting 0 orders in all pixels for all time steps. In summary, the best model and its accuracy are determined primarily by the ADD, and the pixel size and interval length are merely parameters to control that. The forecaster's goal is to create a grid with small enough pixels without losing a recognizable pattern.

5. Conclusion

Glossary

ADD Average Daily Demand. 26

CART Classification and Regression Trees. 10

CV Cross Validation. 9

fnaive "Flexible" STL Decomposition, with tuned ns parameter. 23

harima Autoregressive Integrated Moving Average Method, trained on horizontal time series. 22

hcroston Croston's Method, trained on horizontal time series. 22

hets ETS State Space Method, trained on horizontal time series. 22

hholt Holt's Linear Trend Method, trained on horizontal time series. 22

hhwinters Holt-Winter's Seasonal Method, trained on horizontal time series. 22

hses Simple Exponential Smoothing Method, trained on horizontal time series. 22

hsma Simple Moving Average Method, trained on horizontal time series. 22

htheta Theta Method, trained on horizontal time series. 22

MASE Mean Absolute Scaled Error. 18

ML Machine Learning. 3, 24

naive (Seasonal) Naïve Method. 22

pnaive "Periodic" STL Decomposition, with ns parameter set to large number. 23

RF Random Forest. 8

rtarima Autoregressive Integrated Moving Average Method, (re)trained on vertical time series. 23

rtets ETS State Space Method, (re)trained on vertical time series. 23

rtholt Holt's Linear Trend Method, (re)trained on vertical time series. 23

rtses Simple Exponential Smoothing Method, (re)trained on vertical time series. 23

rttheta Theta Method, (re)trained on vertical time series. 23

STL Seasonal and Trend Decomposition using Loess. 8

SVM Support Vector Machine. 11

SVR Support Vector Regression. 8

trivial Trivial Method. 22

UDP Urban Delivery Platform. 3

varima Autoregressive Integrated Moving Average Method, trained on vertical time series. 23

vets ETS State Space Method, trained on vertical time series. 23

vholt Holt's Linear Trend Method, trained on vertical time series. 23

vrfr Random Forest Regression Method, trained on vertical time series. 24

VRP Vehicle Routing Problem. 2

vses Simple Exponential Smoothing Method, trained on vertical time series. 23

vsvr Support Vector Regression Method, trained on vertical time series. 24

vtheta Theta Method, trained on vertical time series. 23

Appendix A. Tabular and Real-time Forecasts without Retraining

Regarding the structure of the feature matrix for the ML models in Sub-section 3.6.4, we provide an alternative approach that works without the STL method. Instead of decomposing a time series and arranging the resulting seasonally-adjusted time series a_t into a matrix $\boldsymbol{X}$, one can create a matrix with two types of feature columns mapped to the raw observations in $\vec{y}$: While the first group of columns takes all observations of the same time of day over a horizon of, for example, one week ($n_h = 7$), the second group takes all observations covering a pre-defined time horizon, for example 3 hours ($n_r = 3$ for 60-minute time steps), preceding the time step to be fitted. Thus, we exploit the two-dimensional structure of time tables as well, and conceptually model historical and recent demand. The alternative feature matrix appears as follows where the first three columns are the historical and the last three the recent demand features:

$$\vec{y} = \begin{pmatrix} y_T \\ y_{T-1} \\ \dots \\ y_{1+n_h H} \end{pmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} y_{T-H} & y_{T-2H} & \dots & y_{T-n_h H} & y_{T-1} & y_{T-2} & \dots & y_{T-n_r} \\ y_{T-1-H} & y_{T-1-2H} & \dots & y_{T-1-n_h H} & y_{T-2} & y_{T-3} & \dots & y_{T-n_r-1} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ y_{1+(n_h-1)H} & y_{1+(n_h-2)H} & \dots & y_1 & y^*_{1+n_h H-1} & y^*_{1+n_h H-2} & \dots & y^*_{1+n_h H-n_r} \end{bmatrix}$$

Being a detail, we note that the recent demand features lying on the end of the previous day are set to 0, which is shown with the * notation above. This alignment of the undecomposed order data y_t ensures that the ML models learn the two seasonal patterns independently. The parameters n_h and n_r must be adapted to the data, but we found the above values to work well.

As such matrices resemble time tables, we refer to them as tabular. However, we found the ML models with vertical time series to outperform the tabular ML models, which is why we disregarded them in the study. This tabular form could be beneficial for UDPs with a demand that exhibits a weaker seasonality such as a meal delivery platform.

Appendix B. Enhancing Forecasting Models with External Data

In this appendix, we show how the feature matrix in Sub-section 3.6.4 can be extended with features other than historical order data. Then, we provide an overview of what external data we tried out as predictors in our empirical study.

Appendix B.1. Enhanced Feature Matrices

Feature matrices can naturally be extended by appending new feature columns $x_{t,f}$ or x_f on the right where the former represent predictors changing throughout a day and the latter being static either within a pixel or across a city. f refers to an external predictor variable, such as one of the examples listed below. In the SVR case, the columns should be standardized before fitting as external predictors are most likely on a different scale than the historic order data. Thus, for a matrix with seasonally-adjusted order data a_t in it, an enhanced matrix looks as follows:

$$\vec{y} = \begin{pmatrix} a_T \\ a_{T-1} \\ \dots \\ a_{H+1} \end{pmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} a_{T-1} & a_{T-2} & \dots & a_{T-H} & & x_{T,A} & \dots & x_B & \dots \\ a_{T-2} & a_{T-3} & \dots & a_{T-(H+1)} & & x_{T-1,A} & \dots & x_B & \dots \\ \dots & \dots & \dots & \dots & & \dots & \dots & \dots & \dots \\ a_H & a_{H-1} & \dots & a_1 & & x_{H+1,A} & \dots & x_B & \dots \end{bmatrix}$$

Similarly, we can also enhance the tabular matrices from Appendix A. The same comments as for their pure equivalents in Sub-section 3.6.4 apply, in particular, that ML models trained with an enhanced matrix can process real-time data without being retrained.

Appendix B.2. External Data in the Empirical Study

In the empirical study, we tested four groups of external features that we briefly describe here.

Calendar Features:

- Time of day (as synthesized integers: e.g., 1,050 for 10:30 am, or 1,600 for 4 pm)
- Day of week (as one-hot encoded booleans)

- Work day or not (as booleans)

Features derived from the historical Order Data:

- Number of pre-orders for a time step (as integers)
- 7-day SMA of the percentages of discounted orders (as percentages): The platform is known for running marketing campaigns aimed at first-time customers at irregular intervals. Consequently, the order data show a wave-like pattern of coupons redeemed when looking at the relative share of discounted orders per day.

Neighborhood Features:

- Ambient population (as integers) as obtained from the ORNL LandScan database
- Number of active platform restaurants (as integers)
- Number of overall restaurants, food outlets, retailers, and other businesses (as integers) as obtained from the Google Maps and Yelp web services

Real-time Weather (raw data obtained from IBM's Wunderground database):

- Absolute temperature, wind speed, and humidity (as decimals and percentages)
- Relative temperature with respect to 3-day and 7-day historical means (as decimals)
- Day vs. night defined by sunset (as booleans)
- Summarized description (as indicators -1, 0, and $+1$)
- Lags of the absolute temperature and the summaries covering the previous three hours

Unfortunately, we must report that none of the mentioned external data improved the accuracy of the forecasts. Some led to models overfitting the data, which could not be regulated. Manual tests revealed that real-time weather data are the most promising external source. Nevertheless, the data provided by IBM's Wunderground database originate from weather stations close to airports, which implies that we only have the same aggregate weather data for the entire city. If weather data is available on a more granular basis in the future, we see some potential for exploitation.

Appendix C. Raw Order Data in the Case Study

The raw data for the empirical study in Section 4 was provided by a meal delivery platform operating in five cities in France in 2016. The platform received a total of 686,385 orders distributed as follows:

City	**Launch Day**	**Orders**
Bordeaux	July 18	64,012
Lille	October 30	14,362
Lyon	February 21	214,635
Nantes	October 31	12,900
Paris	March 7	380,476

The part of the database relevant for forecasting can be thought of as one table per city, where each row represents one order and consists of the following groups of columns:

1. **Restaurant Data**
   (a) unique ID and name
   (b) pickup location as latitude-longitude pair
2. **Customer Data**
   (a) unique ID, name, and phone number
   (b) delivery location as latitude-longitude pair (mostly physical addresses but also public spots)
3. **Timestamps**
   (a) placement via the smartphone app
   (b) fulfillment workflow (pickup, delivery, cancellation, re-deliveries)
4. **Courier Data**
   (a) unique ID, name, and phone number
   (b) shift data (begin, breaks, end)
   (c) average speed
5. **Order Details**
   (a) meals and drinks
   (b) prices and discounts granted

Appendix D. Forecasting Accuracies during Peak Times

This appendix shows all tables from the main text with the MASE averages calculated from time steps within peak times that are defined to be from 12 pm to 2 pm (=lunch) or from 6 pm to 8 pm (=dinner). While the exact decimals of the MASEs differ, the relative ranks of the forecasting methods are the same except in rare cases.

Table D.5: Top-3 models by training weeks and average demand (1 km^2 pixel size, 60-minute time steps)

Training	**Rank**	**No Demand** (0 - 2.5)			**Low Demand** (2.5 - 10)			**Medium Demand** (10 - 25)			**High Demand** (25 - ∞)		
		Method	MASE	n	Method	MASE	n	Method	MASE	n	Method	MASE	n
3	1	***trivial***	0.794	4586	***hsma***	0.817	2975	***hsma***	0.838	2743	***rtarima***	0.871	2018
	2	*hsma*	0.808		*hses*	0.847		*hses*	0.851		*rtses*	0.872	
	3	*pnaive*	0.938		*hets*	0.848		*hets*	0.853		*rtets*	0.874	
4	1	***trivial***	0.791	4532	***hsma***	0.833	3033	***hsma***	0.839	2687	***vrfr***	0.848	2016
	2	*hsma*	0.794		*hses*	0.838		*hses*	0.847		***rtarima***	0.851	
	3	*pnaive*	0.907		*hets*	0.841		*hets*	0.851		*rtses*	0.857	
5	1	***trivial***	0.782	4527	***hsma***	0.844	3055	***hsma***	0.841	2662	***vrfr***	0.849	2019
	2	*hsma*	0.802		*hses*	0.851		*hets*	0.844		***rtarima***	0.851	
	3	*pnaive*	0.888		*hets*	0.863		*hses*	0.845		*vsvr*	0.853	
6	1	***trivial***	0.743	4470	***hsma***	0.843	3086	***hsma***	0.841	2625	***vrfr***	0.844	2025
	2	*hsma*	0.765		*hses*	0.853		*hses*	0.844		***hets***	0.847	
	3	*pnaive*	0.836		*hets*	0.861		*hets*	0.844		*vsvr*	0.849	
7	1	***trivial***	0.728	4454	***hsma***	0.855	3132	***hets***	0.843	2597	***hets***	0.839	2007
	2	*hsma*	0.744		*hses*	0.862		*hsma*	0.845		***vrfr***	0.842	
	3	*pnaive*	0.812		*hets*	0.868		***vsvr***	0.849		*vsvr*	0.846	
8	1	***trivial***	0.736	4402	***hsma***	0.865	3159	***hets***	0.843	2575	***hets***	0.837	2002
	2	*hsma*	0.759		*hets*	0.874		***vsvr***	0.848		***vrfr***	0.841	
	3	*pnaive*	0.820		*hses*	0.879		*hsma*	0.850		*vsvr*	0.847	

Table D.6: Ranking of benchmark and horizontal models (1 km^2 pixel size, 60-minute time steps): the table shows the ranks for cases with $2.5 < ADD < 25$ (and $25 < ADD < \infty$ in parentheses if they differ)

Training	**Benchmarks**			**Horizontal (whole-day-ahead)**							
	naive	*fnaive*	*paive*	*harima*	*hcroston*	*hets*	*hholt*	*hhwinters*	*hses*	*hsma*	*htheta*
3	11	7 (2)	8 (5)	5 (7)	4	3	9 (10)	10 (9)	2 (6)	1	6 (8)
4	11	7 (2)	8 (3)	5 (6)	4 (5)	3 (1)	9 (10)	10 (9)	2 (8)	1 (4)	6 (7)
5	11	7 (2)	8 (4)	5 (3)	4 (9)	3 (1)	9 (10)	10 (5)	2 (8)	1 (6)	6 (7)
6	11	8 (5)	9 (6)	5 (4)	4 (7)	2 (1)	10	7 (2)	3 (8)	1 (9)	6 (3)
7	11	8 (5)	10 (6)	5 (4)	4 (7)	2 (1)	9 (10)	7 (2)	3 (8)	1 (9)	6 (3)
8	11	9 (5)	10 (6)	5 (4)	4 (7)	2 (1)	8 (10)	7 (2)	3 (8)	1 (9)	6 (3)

Table D.7: Ranking of classical models on vertical time series (1 km^2 pixel size, 60-minute time steps): the table shows the ranks for cases with $2.5 < ADD < 25$ (and $25 < ADD < \infty$ in parentheses if they differ)

Training	**Benchmarks**		**Vertical (whole-day-ahead)**					**Vertical (real-time)**				
	hets	*hsma*	*varima*	*vets*	*vholt*	*vses*	*vtheta*	*rtarima*	*rtets*	*rtholt*	*rtses*	*rttheta*
3	2 (10)	1 (7)	6 (4)	8 (6)	10 (9)	7 (5)	11 (12)	4 (1)	5 (3)	9 (8)	3 (2)	12 (11)
4	2 (7)	1 (10)	6 (4)	8 (6)	10 (9)	7 (5)	12 (11)	3 (1)	5 (3)	9 (8)	4 (2)	11 (12)
5	2 (3)	1 (10)	7 (5)	8 (7)	10 (9)	6	11	4 (1)	5 (4)	9 (8)	3 (2)	12
6	2 (1)	1 (10)	6 (5)	8 (7)	10 (9)	7 (6)	11 (12)	3 (2)	5 (4)	9 (8)	4 (3)	12 (11)
7	2 (1)	1 (10)	8 (5)	7	10 (9)	6	11 (12)	5 (2)	4	9 (8)	3	12 (11)
8	2 (1)	1 (9)	8 (5)	7	10 (8)	6	12 (10)	5 (2)	4	9 (6)	3	11

Table D.8: Ranking of ML models on vertical time series (1 km^2 pixel size, 60-minute time steps): the table shows the ranks for cases with $2.5 < ADD < 25$ (and $25 < ADD < \infty$ in parentheses if they differ)

Training	**Benchmarks**				**ML**	
	fnaive	*hets*	*hsma*	*rtarima*	*vrfr*	*vsvr*
3	6	2 (5)	1 (3)	3 (1)	5 (2)	4
4	6 (5)	2 (3)	1 (6)	3 (2)	5 (1)	4
5	6 (5)	2 (4)	1 (6)	4 (2)	5 (1)	3
6	6 (5)	2	1 (6)	4	5 (1)	3
7	6 (5)	2 (1)	1 (6)	4	5 (2)	3
8	6 (5)	2 (1)	1 (6)	4	5 (2)	3

References

[1] Alcaraz, J.J., Caballero-Arnaldos, L., Vales-Alonso, J., 2019. Rich vehicle routing problem with last-mile outsourcing decisions. Transportation Research Part E: Logistics and Transportation Review 129, 263–286.

[2] Assimakopoulos, V., Nikolopoulos, K., 2000. The theta model: a decomposition approach to forecasting. International Journal of Forecasting 16, 521–530.

[3] Bao, Y., Wang, W., Zhang, J., 2004. Forecasting intermittent demand by svms regression, in: 2004 IEEE International Conference on Systems, Man and Cybernetics, pp. 461–466.

[4] Barbour, A.J., Parker, R.L., 2014. psd: Adaptive, sine multitaper power spectral density estimation for r. Computers & Geosciences 63, 1–8.

[5] Bell, F., Smyl, S., 2018. Forecasting at uber: An introduction. `https://eng.uber.com/forecasting-introduction/`. Accessed: 2020-10-01.

[6] Box, G., Cox, D., 1964. An analysis of transformations. Journal of the Royal Statistical Society. Series B (Methodological) 26, 211–252.

[7] Box, G., Jenkins, G., 1962. Some statistical aspects of adaptive optimization and control. Journal of the Royal Statistical Society. Series B (Methodological) 24, 297–343.

[8] Box, G., Jenkins, G., 1968. Some recent advances in forecasting and control. Journal of the Royal Statistical Society. Series C (Applied Statistics) 17, 91–109.

[9] Box, G., Jenkins, G., Reinsel, G., Ljung, G., 2015. Time Series Analysis: Forecasting and Control. Wiley Series in Probability and Statistics, Wiley.

[10] Breiman, L., 2001. Random forests. Machine Learning 45, 5–32.

[11] Breiman, L., Friedman, J., Olshen, R., Stone, C., 1984. Classification and Regression Trees. Wadsworth.

[12] Brockwell, P., Davis, R., 2016. Introduction to Time Series and Forecasting. Springer Texts in Statistics, Springer.

[13] Brown, R., 1959. Statistical Forecasting for Inventory Control. McGraw/Hill.

[14] Chen, L., Zhang, T.q., 2006a. Hourly water demand forecast model based on bayesian least squares support vector machine. Journal of Tianjin University 39, 1037–1042.

[15] Chen, L., Zhang, T.q., 2006b. Hourly water demand forecast model based on least squares support vector machine. Journal of Harbin Institute of Technology 38, 1528–1530.

[16] Cleveland, R., Cleveland, W., McRae, J., Terpenning, I., 1990. Stl: A seasonal-trend decomposition procedure based on loess. Journal of Official Statistics 6, 3–73.

[17] Croston, J.D., 1972. Forecasting and stock control for intermittent demands. Journal of the Operational Research Society 23, 289–303.

[18] Dagum, E., Bianconcini, S., 2016. Seasonal Adjustment Methods and Real Time Trend-Cycle Estimation. Statistics for Social and Behavioral Sciences, Springer.

[19] De Gooijer, J., Hyndman, R., 2006. 25 years of time series forecasting. International Journal of Forecasting 22, 443–473.

[20] Drucker, H., Burges, C., Kaufman, L., Smola, A., Vapnik, V., 1997. Support vector regression machines, in: Advances in Neural Information Processing Systems, Springer. pp. 155–161.

[21] Ehmke, J.F., Campbell, A.M., Thomas, B.W., 2018. Optimizing for total costs in vehicle routing in urban areas. Transportation Research Part E: Logistics and Transportation Review 116, 242–265.

[22] Gardner, E., McKenzie, E., 1985. Forecasting trends in time series. Management Science 31, 1237–1246.

[23] Hansen, J., McDonald, J., Nelson, R., 2006. Some evidence on forecasting time-series with support vector machines. Journal of the Operational Research Society 57, 1053–1063.

[24] Hastie, T., Tibshirani, R., Friedman, J., 2013. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.

[25] Herrera, M., Torgo, L., Izquierdo, J., Pérez-García, R., 2010. Predictive models for forecasting hourly urban water demand. Journal of Hydrology 387, 141–150.

[26] Hirschberg, C., Rajko, A., Schumacher, T., Wrulich, M., 2016. Mckinsey: The changing market for food delivery. `https://www.mckinsey.com/industries/high-tech/our-insights/the-changing-market-for-food-delivery`. Accessed: 2020-10-01.

[27] Ho, T.K., 1998. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence 20, 832–844.

[28] Holt, C., 1957. Forecasting seasonals and trends by exponentially weighted moving averages. ONR Memorandum 52.

[29] Hou, L., Li, D., Zhang, D., 2018. Ride-matching and routing optimisation: Models and a large neighbourhood search heuristic. Transportation Research Part E: Logistics and Transportation Review 118, 143–162.

[30] Hyndman, R., Athanasopoulos, G., 2018. Forecasting: Principles and Practice. OTexts.

[31] Hyndman, R., Billah, B., 2003. Unmasking the theta method. International Journal of Forecasting 19, 287–290.

[32] Hyndman, R., Khandakar, Y., 2008. Automatic time series forecasting: The forecast package for r. Journal of Statistical Software 26.

[33] Hyndman, R., Koehler, A., 2006. Another look at measures of forecast accuracy. International Journal of Forecasting 22, 679–688.

[34] Hyndman, R., Koehler, A., Ord, K., Snyder, R., 2008. Forecasting with Exponential Smoothing: the State Space Approach. Springer.

[35] Hyndman, R., Koehler, A., Snyder, R., Grose, S., 2002. A state space framework for automatic forecasting using exponential smoothing methods. International Journal of Forecasting 18, 439–454.

[36] Kim, S., Kim, H., 2016. A new metric of absolute percentage error for intermittent demand forecasts. International Journal of Forecasting 32, 669–679.

[37] Kwiatkowski, D., Phillips, P., Schmidt, P., Shin, Y., 1992. Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root? Journal of Econometrics 54, 159–178.

[38] Laptev, N., Smyl, S., Shanmugam, S., 2017. Engineering extreme event forecasting at uber with recurrent neural networks. `https://eng.uber.com/neural-networks/`. Accessed: 2020-10-01.

[39] Ma, L., Zhang, G., Lu, E., 2018. Using the gradient boosting decision tree to improve the delineation of hourly rain areas during the summer from advanced himawari imager data. Journal of Hydrometeorology 19, 761–776.

[40] Masmoudi, M.A., Hosny, M., Demir, E., Genikomsakis, K.N., Cheikhrouhou, N., 2018. The dial-a-ride problem with electric vehicles and battery swapping stations. Transportation research part E: logistics and transportation review 118, 392–420.

[41] Mason, L., Baxter, J., Bartlett, P.L., Frean, M.R., 2000. Boosting algorithms as gradient descent, in: Advances in neural information processing systems, pp. 512–518.

[42] Müller, K.R., Smola, A., Rätsch, G., Schölkopf, B., Kohlmorgen, J., Vapnik, V., 1997. Predicting time series with support vector machines, in: International Conference on Artificial Neural Networks, Springer. pp. 999–1004.

[43] Müller, K.R., Smola, A., Rätsch, G., Schölkopf, B., Kohlmorgen, J., Vapnik, V., 1999. Using support vector machines for time series prediction. Advances in Kernel Methods — Support Vector Learning , 243–254.

[44] Ord, K., Fildes, R., Kourentzes, N., 2017. Principles of Business Forecasting. WESSEX Press.

[45] Pegels, C., 1969. Exponential forecasting: Some new variations. Management Science 15, 311–315.

[46] Prestwich, S., Rossi, R., Tarim, A., Hnich, B., 2014. Mean-based error measures for intermittent demand forecasting. International Journal of Production Research 52, 6782–6791.

[47] Schölkopf, B., Knirsch, P., Smola, A., Burges, C., 1998. Fast approximation of support vector kernel expansions, and an interpretation of clustering as approximation in feature spaces, in: Mustererkennung 1998. Springer, pp. 125–132.

[48] Singleton, A.D., Spielman, S., Folch, D., 2017. Urban Analytics. Sage.

[49] Smola, A., Schölkopf, B., 2004. A tutorial on support vector regression. Statistics and Computing 14, 199–222.

[50] Stitson, M., Gammerman, A., Vapnik, V., Vovk, V., Watkins, C., Weston, J., 1999. Support vector regression with anova decomposition kernels. Advances in Kernel Methods — Support Vector Learning , 285–292.

[51] Syntetos, A., Boylan, J., 2005. The accuracy of intermittent demand estimates. International Journal of forecasting 21, 303–314.

[52] Taylor, J., 2003. Exponential smoothing with a damped multiplicative trend. International Journal of Forecasting 19, 715–725.

[53] Vapnik, V., 2013. The Nature of Statistical Learning Theory. Springer.

[54] Vapnik, V., Chervonenkis, A., 1964. A note on one class of perceptrons. Automation and Remote Control 25.

[55] Vapnik, V., Lerner, A., 1963. Pattern recognition using generalized portrait method. Automation and Remote Control 24, 774–780.

[56] Wang, Z., 2018. Delivering meals for multiple suppliers: Exclusive or sharing logistics service. Transportation Research Part E: Logistics and Transportation Review 118, 496–512.

[57] Winkenbach, M., Kleindorfer, P.R., Spinler, S., 2015. Enabling urban logistics services at la poste through multi-echelon location-routing. Transportation Science 50, 520–540.

[58] Winters, P., 1960. Forecasting sales by exponentially weighted moving averages. Management Science 6, 324–342.