Reasoning over Time into Models with DataTime

Gauthier Lyan, Jean-Marc Jézéquel, David Gross-Amblard, Romain Lefeuvre, Benoit Combemale

Noname manuscript No.
(will be inserted by the editor)

Reasoning over Time into Models with DataTime

Gauthier LYAN · Jean-Marc JÉZÉQUEL · David GROSS-AMBLARD · Romain LEFEUVRE · Benoit COMBEMALE

Received: date / Accepted: date

Abstract Models at runtime have been initially investigated for adaptive systems. Models are used as a reflective layer of the current state of the system to support the implementation of a feedback loop. More recently, models at runtime have also been identified as key for supporting the development of full-fledged digital twins. However, this use of models at runtime raises new challenges, such as the ability to seamlessly interact with the past, present and future states of the system. In this paper, we propose a framework called DataTime to implement models at runtime which capture the state of the system according to the dimensions of both time and space, here modeled as a directed graph where both nodes and edges bear local states (ie. values of properties of interest). DataTime offers a unifying interface to query the past, present and future (predicted) states of the system. This unifying interface provides i) an optimized structure of the time series that capture the past states of the system, possibly evolving over time, ii) the ability to get the last available value provided by the system's sensors, and iii) a continuous micro-learning over graph edges of a predictive model to make it possible to query future states, either locally or more globally, thanks to a composition law. The framework has been developed and evaluated in the context of the Intelligent Public Transportation Systems of the city of Rennes (France). This experimentation has demonstrated how DataTime can be used for managing data from the past, the present and the future, and facilitate the development of digital twins.

Keywords Models at Runtime · Digital Twins · Data Analysis · Intelligent Public Transportation Systems

G. Lyan
Keolis Rennes
Rennes, FRANCE
E-mail: gauthier.lyan@keolis.com

J.-M. Jézéquel, D. Gross-Amblard, R. Lefeuvre, B. Combemale
Univ Rennes, CNRS, IRISA,
Rennes, FRANCE
E-mail: first.last@irisa.fr

1 Introduction

So called Intelligent Public Transportation Systems (IPTS) are complex socio-technical systems, involving people (*e.g.,* the users of the networks) as well as supporting infrastructures, from the transportation means themselves (*e.g.,* buses) to the IT supporting them [1, 23]. One key feature of IPTSs are their information systems allowing a network operator to plan, analyze and manage the network with respect to metrics such as transportation time (or commercial speed), energy consumption or accident rates. When an IPTS at least partially relies on buses, these metrics are difficult to predict. Traffic indeed varies widely during the day, with rush hours further slowing down bus loading and unloading and compromising planned connections, with a significant impact on travel time. Furthermore, when some roadworks (or some unforeseen condition such as recurrent flood due to deferral of traffic) happen on a street, the IPTS should be reconfigured. This reconfiguration, which is a strategical decision, should be handled by the operator. This means diverting affected lines using the best possible new routes, a challenge for many centuries-old European cities built around crowded, winding city centers. Supporting IT systems for IPTSs are typically made of two relatively independent parts, one organized around *space* and the second one around *time*. The spatial one is an a priori model of

the network: what are the network topology (modeled as a directed graph), the transportation means (e.g., bus, tramways, metros), their itineraries, the infrastructure (e.g., kinds of roads), the scheduled departures, etc. The temporal one is made of the (huge) time series of data gathered from the many sensors available in the IPTS. The existing time series give the opportunity to not only provide a *model at runtime* [20], but also a full-fledged *digital twin* [5] of the IPTS. That is to say a virtual representation (or replica) of an actual IPTS that is continuously updated with real-time data throughout its life-cycle. At the same time, it can interact with and influence the IPTS to analyze, plan and manage the operations using machine learning techniques. However these space and time parts of the supporting IT are most often not well integrated, making it hard in practice for network operators to leverage the avalanche of data gathered from the IPTS. For instance diverted lines (i.e., space modification) have no historical data to learn from, so even machine learning techniques that have successfully been applied on graph-based structures typical of an IPTS [3, 11, 25] cannot work out of the box. Moreover, as for many other network problems (e.g., electricity or water networks), properties of interest (e.g., commercial speed of a bus along a line) are compositions of smaller independent parts (e.g., the speed on each bus inter-stops along the line). This enables the prediction on the behaviour of composite objects (a path in the graph), based on the predictions of their sub-parts (i.e., on edges) and relevant composition laws, aka. a micro-learning framework [10–12]. The composition laws depend on the nature of the properties of interest, for instance an additive property such as speed can use simple sum as composition law, based on travel time and distance of each part. This opens what-if analytics scenarios, where new objects never observed before can be predicted based on their simpler parts, and can then be used e.g., to optimize detours in case of unforeseen events such as roadworks or floods. These scenarios are mostly managed manually by the technical experts, for example in the case of a detour, the new path is proposed thanks to the expert's knowledge. Our framework provide tools to these experts and offer them a way exploiting data. For instance, it can help them to compare different deviation proposals by providing predictions of the interest property for each path at the given time. We can also take as an example what-if scenarios such as the creation of a new line which can be brought back to a problem of shortest path which is an np-complete problem. In that case, the usefulness of a tool that automatically proposes a solution is very clear. A possible direction for integrating these models (spatial, temporal and predictive) would be to articulate them around the notion of digital twin and the concept of time, i.e., digital twins extending themselves towards Past, Present, and Future [13].

For that purpose, we propose a new framework, called DataTime, to implement models at runtime which capture the state of the system according to both time and space, here modeled as a directed graph. In this graph, both nodes and edges bear local and independent states (ie. values of properties of interest). DataTime offers a unifying interface to query the past, present and future (predicted) states of the system. This unifying interface provides i) an optimized structure of the time series that capture the past states of the system, possibly evolving over time, ii) the ability to get the last available value provided by the system's sensors, and iii) a continuous micro-learning over graph edges of a predictive model to make it possible to query future states, either locally or more globally, thanks to a composition law. We apply our framework in the context of an urban transportation system, and concretely deploy it as a decision making tool and evaluate it on the IPTS of the city of Rennes (France), that is operated by the Keolis company.

The main contribution of this paper is the DataTime framework that allows seamless spatio-temporal data analysis and strategical decision making. Also it extends [16] with an evaluation of the predictive model. In particular, we demonstrate the relevance of the micro-learning approach with regard to the macro-learning approach, and discuss both the rationales wrt. the compositional law and the possible limits of the approach.

The rest of this paper is organized as follows. We first introduce the DataTime framework (Section 2), and then we present its use in the context of an IPTS (Section 3). Furthermore, in Section 4 we describe how it was deployed at Keolis Rennes in the context of the real bus network of the city of Rennes (France), and what lessons we have learned from this experimentation. Subsequently, in Section 5 we evaluate the approach based on the composition laws (aka. compositional prediction) used in the framework for prospective analysis in graphs. Finally, Section 6 discusses related work, before we conclude and raise several perspectives in Section 7.

2 The DataTime Framework

Figure 1 shows a simplified class diagram of DataTime. The objective of this framework is to enable graph-based seamless space and time exploration through the analysis of historical data, real time data and predicted data. It is composed of two main parts: i) the spatial

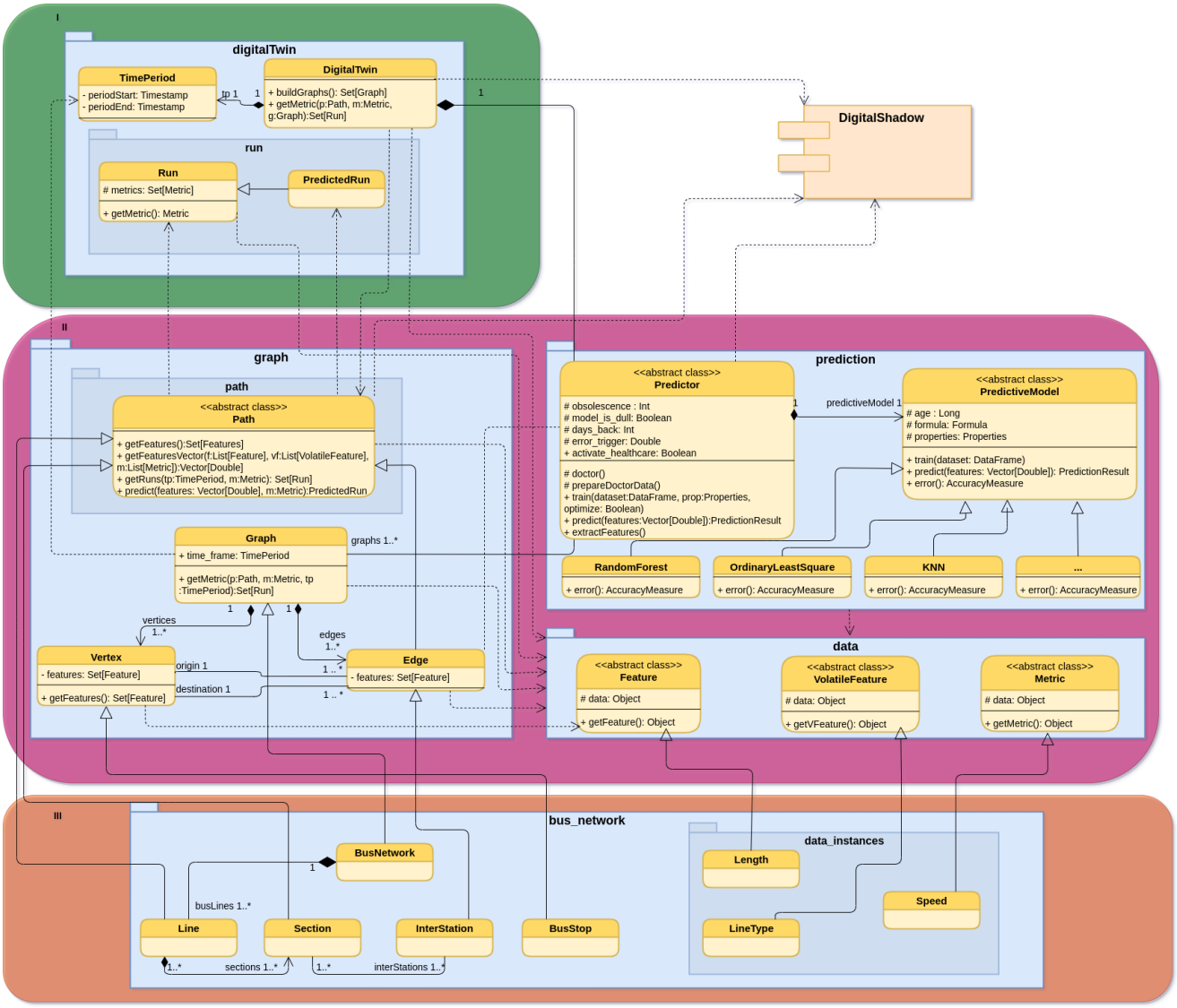

Figure 1: The DataTime framework (excerpt)

model and predictors configuration (part II of Fig. 1) that is dedicated to the designers who develop a specific system based on DataTime. The second part ii) is the digital twin (part I of Fig. 1) which is the entry point of the end-users for reasoning over time about the spatial model. It includes a digital shadow that collects all the required historical data. A third component has been added in the part III of Fig. 1 to represent the required endeavour to use DataTime for the development of a particular system (see Section 3 the IPTS example).

2.1 Spatial model and predictors configuration

2.1.1 Graph

The spatial model is defined as a classical directed graph structure as seen in part II of Fig. 1. Focusing on the `graph` package: a set of vertices that are nodes with individual characteristics, and a set of edges that are one way connections between two vertices (a bidirectional edge is simply represented with two directed edges). Graphs have a built-in `time_frame` attribute that represents the time-frame (or window) for which corresponding referential and historical data exists. Consequently it is to the discretion of the designer to consider that changes amongst a graph structure yield a new graph, or if it just update its `time_frame`

by expanding, reducing it or leaving it unchanged (*e.g.*, when minor changes occur).

`Edge` and `vertex` characteristics (features) are made abstract in order to use any kind of data. In particular, the architecture of `path`, `edge`, and `vertex` is a two layers loose occurrence of the `composite` pattern. `path` is playing the role of `component`, `edge` is playing the role of leaf. At the concrete level, in part III of Figure 1, `line` is playing the role of composite. The package `data` contains the different features and metrics definitions. In order to make those generic, we separated their identification from their actual data and type (following the type-object design pattern). This package contains three abstract classes that have to be specialized for a specific system: `Features`, `VolatileFeatures` and `Metrics`. `Features` represent characteristics of the edges and vertices. `VolatileFeatures` are characteristics that are not represented in the data, but that can be computed on the go (e.g., bus line type extrapolated from its identifier, electrical cable resistance computed from its length, etc.). `Metrics` are data that are not characteristics, such as measures (speed, volume of water per hour, etc.) or time related information. Thus, data instances that extend those classes must be able to query data from the `Digital Shadow` (cf. Subsection 2.2). The package `path` represents the abstraction of a path through the graph, that is, a sequence of at least one edge for the thinnest grain. `Paths` are abstract, as a result one can easily create a path hierarchy if relevant for the targeted application domain (e.g., transportation networks, supply chains, smart-grids, drinking water networks, ...). Paths allow the building of hierarchical structures within the graphs while offering analysis and predictions scalability using micro-learning for the predictive aspects.

2.1.2 Micro-learning

Recently, micro-learning approaches have been successfully applied to graph structures to provide what-if scenarios (e.g., in the context of power grid management [10]). Micro-learning, as explained by [11], refers to small fine-grained learning units as opposed to macro-learning which consists of a single coarse-grained learning unit. In these approaches, specific models of local data are built instead of one large model on the overall dataset. Micro-learning is typically useful for incremental predictive models, where one has to perform step-by-step decisions based on local properties, while yielding quality predictions. For instance for an IPTS, prediction of travel time or road reliability can be made using micro-learning over each edge and then results can be aggregated using a specific composition law (sum for travel time, product for reliability, etc.). In DataTime, data analysis and predictions are made at the level of edges (provided a predictive model has been configured for this purpose). For graphs on which data can be aggregated from edge level to different implementations of parent paths, the prediction or analysis at edge level will be automatically aggregated to higher levels. Specific user defined composition laws are used for this purpose (e.g., speed of a bus between two bus stops, aggregated to the whole bus line, water leaks at single pipes sections, aggregated to the whole pipe, etc.).

Figures 2 and 3 highlight the compositional prediction mechanism in IPTS context. They show how predictions are made on a complete line by aggregating the predictions on each inter-station. Furthermore, they enlighten the comparison with the classical macro approach, where the prediction is done directly on the whole line without taking into account the inter-stations.

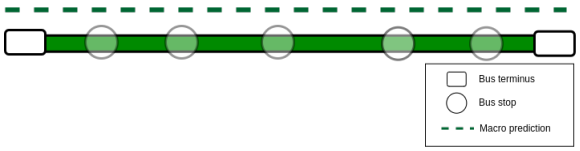

Figure 2: Macro prediction of a property of interest on a bus line

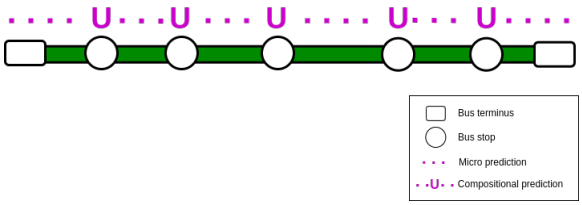

Figure 3: Compositional prediction of a property of interest on a bus line

Let us consider a directed graph G = (V, E) and a measure $\mathcal{M}$ (metric) defined on paths p in G. Let D be a learning dataset of examples of $\mathcal{M}$. We propose the following notion of compositional prediction using micro-learning:

- (Micro-learning) we first learn a model of $\mathcal{M}$ on each individual edge of G.
- (Compositional prediction) Then we build the model of $\mathcal{M}$ on a new path by composition.

More precisely, let $p = (e_1, \ldots, e'_x, \ldots, e_n)$ be a path in the graph G, containing edges e for which training data exists in D and edges e' that are dedicated to p, i.e., created ex-nihilo, with no training data in D. Our goal is to predict a given measure $\mathcal{M}(p)$ for any path p on this graph. We possess some local projections $m(e)$ of $\mathcal{M}$ on a limited number of known edges e, and some local predictions $m(e')$ of $\mathcal{M}$ on the other new edges e' along this path and, typically, a composition law $\mathcal{C}$ links these measures:

$$\mathcal{M}(p) = \mathcal{C}(m(e_1), \ldots, m(e'_i), \ldots,$$
$$m(e_j), \ldots, m(e'_k), \ldots, m(e_n)).$$

We then define our compositional prediction approach:

Let $G = (V, E)$ be a directed graph, $\mathcal{M}$ a target measure, a composition law $\mathcal{C}$ and a dataset $\mathcal{D}$ of observations $m(e)$ on edges of G. Given a target path $p = (e_1, \ldots, e'_x, \ldots, e_n)$, the compositional prediction of $\mathcal{M}(p)$ is given by:

- building a model $m^*(e)$ to predict $m(e)$ for each edge e and $e' \in V \times V$, according to edge features $F(e)$ and $F(e')$;
- approximating the prediction of $\mathcal{M}(p)$ by

$$\mathcal{M}^*(p) = \mathcal{C}(m^*(e_1), \ldots, m^*(e'_x), \ldots, m^*(e_n)).$$

For non cumulative metrics such as speed, we merely calculate it by relating the sum of sub-parts distance to the sum of sub-parts travel time. The composition law is defined as follows:

$$\mathcal{C}(\sum distance(e_i) / \sum m(e_i))$$

where metric m is the travel time on edges e_i, either real or predicted.

2.1.3 Predictors

The `prediction` package contains the abstract class `Predictor` that embodies the predictive system of DataTime. It encapsulates predictive models and associated tools for various purposes, from training to prediction, both in space and time. It proposes the following services:

- Choice of the predictive model, in order to choose the best suited predictive model depending on the prediction issue (using the strategy design pattern).
- Optimization of the model by searching for the best hyper-parameters. It embeds a grid search system that can either use a set of properties to seek from, or a random one (through the parameter `optimize` of `Predictor.train()` in Fig. 1).
- Training of the model, by feeding it with a training dataset, hyper-parameters and choosing whether to optimize the predictive model using grid search (through the parameters of `Predictor.train()` in Fig. 1).
- Saving and loading the predictors by transferring their data to the `Digital Shadow` interfaces (through the `Predictor.train()` method in Fig. 1).
- The models are monitored so that they remain relevant over time. Two variables are considered for monitoring purposes : models' lifetime and models' error rate. They become obsolete if their lifetime is exceeded or if their error rate when tested against recent data is higher then a threshold. An obsolete model should be resp. fine tuned or abandoned for a new one trained on fresher data. This service can be disabled if needed (using the attribute `Predictor.activate_healthcare` in Fig. 1).
- Features extraction for prediction, that explores the features of the edges to make predictions (`Predictor.extractFeatures()` in Fig. 1).
- Predictions, by returning a `PredictedRun` when called (`Predictor.predict()` in Fig. 1).

To configure a predictor, a designer has to extend `Predictor` and implement the `prepareDoctorData()` method, which aims at preparing data for the health check-up of the predictive model and the training of a new one when needed. The `doctor` is called when one wants to predict something and that the predictive model is more than `obsolescence` days old compared to the current system date. The objective behind that is to keep predictors up to date and accurate, in respect to the evolution of data coming from the `digital shadow`. The second health check is based on real data and the model's accuracy against most recent data in the system. We achieve this by comparing the residuals obtained through predicting from a sample of the last available data from now to `back_days` back in time with their base error (the deviation of the predictive model obtained during the training phase). If the predictive models yield an average residual that is over `error_trigger` times the base error, then a new model is trained with the data yielded by `prepareDoctorData()`. The `doctor` can be deactivated if needed (for example when the models are not subject to deviation because, e.g., the data is quite consistent through time), by setting `activate_healthcare` to false.

`Predictors` can use different predictive models thanks to the abstract class `PredictiveModel`. If one uses a single machine learning API in which predictive models are specializations of a single abstract

class, then any model of such an API should be made available by encapsulating them in a class extending `PredictiveModel`. Then overriding the `error()` method for each model (because the error computation varies, and the result is not always kept into a trained model), which is mandatory for the `doctor()`. Of course, if one wants to code her own predictive model, the simplest way is to extend `PredictiveModel` and override the relevant methods.

Since our system relies on compositional prediction, predictors are referred by edges only. In order to make any prediction with any machine learning model, predictors should have their own set of features, and prediction targets (i.e., formula). Thereby the edges that call the predictors are responsible for giving them the right features by using their inner method `getFeaturesVector` that transforms the features into a vector of doubles for the predictor (with a dummy encoding for categorical features). Predictors should then take the data they need directly within this object. The training of predictors is made by the `Edge` class, that calls the method `train(...)` of `Predictor`. This method has to be fed with a dataframe containing the appropriate training data (i.e., in accordance with the predictor feature set and target), obtained by querying the digital shadow, and a set of hyper-parameters through the parameters `prop`. The parameter `optimize` triggers a randomized grid search algorithm when set to true in order to find the best configuration for the predictive models hyper-parameters. Thanks to this, any machine learning model with any features set and prediction target can be trained and used at any path level in the framework. Finally, the class `Graph` is responsible for the manipulation in a single graph instance such as seamless data analysis (through the method `getMetric(...)`), getting the corresponding historical or real-time data from the `Digital Shadow`. `Graph` is also the entry point for what-if scenarios (creating new edges, vertices, path and analysis against those new objects), and the orchestration of the data between the different parts of the model such as instantiating and feeding predictors with data when they need to be trained (through its edges). The main goal of the class `Graph` is to provide a way to obtain a set of `runs` when one calls the method `getMetric(...)` by seamlessly returning historical, real time or predicted results. It gathers it from historical runs or predicted runs over the given `path` and time period passed to the method `getMetric(...)`. Note that it does not matter if there is one or more instances of graphs (e.g., a set of independent graphs distributed through time, with contiguous time-frames). In both cases it is the `getMetric(..)` method of `Digital Twin` that is used by the end-user to explore data in time and space.

2.2 Digital Twin and Digital Shadow

The `Digital Shadow` is either the representation of an existing data environment on which DataTime can be plugged in a read-only manner in order not to have any side effect on the existing information system. Also, it can be bound in an actual fully managed data environment dedicated to DataTime, making it responsible for the management of all the data flows between the digital twin and the real system. The digital shadow should also be responsible for the saving of the graphs and predictors instances. Moreover, it has the responsibility of creating time-series used for data analysis while keeping them optimized and consistent through time. In short, any data that is yielded either by the `Digital Twin` of DataTime or the real system sensors, etc. should transit and be governed by the `Digital Shadow`. Fig. 4 shows an example of how data transits through the system when `getMetric(...)` is called. The `Digital Twin` of DataTime represents the set of `graph` instances of the model, over time. It contains the time representation through the class `TimePeriod` that embeds two timestamps. This class helps the representation of time frames and instants (when `periodStart == periodEnd`), and time manipulation in the framework. The time periods are used to request the set of graphs that are part of the class `Digital Twin`. When calling the method `getMetric(...)`, it returns a set of `runs` which represent the result of the queries made by the `Digital Twin` over its set of `graphs`. In other words, `runs` represent the action of traveling through a `path` at a given time frame, or a specific event on a `path` for a given time frame. As an example, a bus traveling along a bus line, an amount of water traveling through a section of a pipe network, the loss of electrical current between two poles, etc. Runs are built using historical ore predicted data. Hence, they consist of different measures & metrics or predictions made on the network represented by an instance of the `graph`. Runs should be atomic, that is to say that they should be unique and they should be bound to `edges` only. Runs of longer paths are built by aggregating runs of their sub-parts, and finally, `edges`.Depending on the size of the historical data, runs should be lazily loaded when analyzed. However, saving the runs can be a bad idea if the historical data is huge: it would lead to storage starving in addition to duplicated data. However, this decision depends on the `Digital Shadow` structure. The `Digital Twin` class is

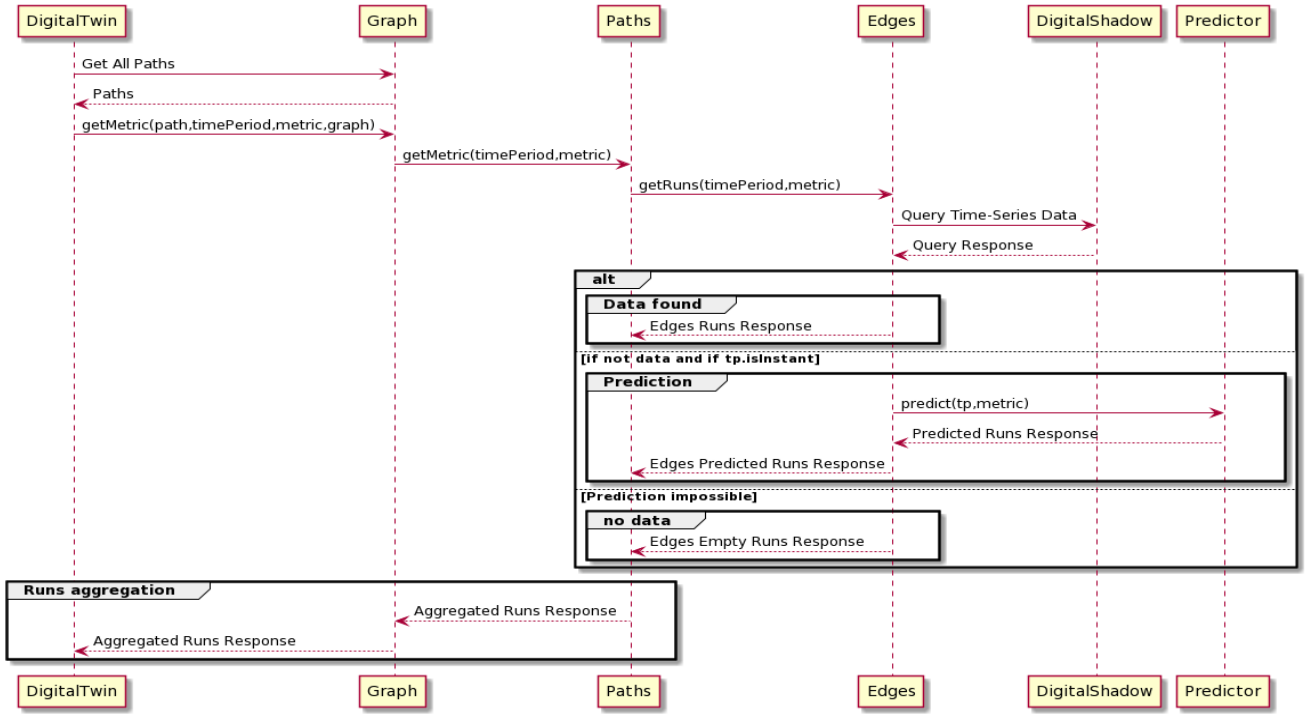

Figure 4: Sequence diagram for getMetric

also responsible for building graph instances by querying the `Digital Shadow` for referential data, yielding `Graph` instances. Which means that the `Digital Twin` is the end-user entry point on the framework.

3 Application to an Intelligent Public Transportation System

To assess the applicability of DataTime, we developed a Scala implementation of the framework adapted to urban bus networks.

IPTS such as modern urban bus networks are complex socio-technical systems. It is made out of hundreds of stakeholders, including humans, sensors, vehicles, information system, etc. Hence, a major concern with such an infrastructure is to gather, organize and normalize data, analysis and decisions in integrated tools. IPTS are slowly evolving networks: an important part of the bus lines is non changing for very long and continuous periods. However, there exist some local variations within their structure, such as the changes on bus lines when long-term roadworks are planned, the creation of new bus lines, etc. As a consequence we considered that the `time_frame` of a single instance of the bus network is corresponding to the period that was defined by the operator (usually 1 to 2 weeks), during which there are no major modifications except for some day to day bus line deviations. In our applicative environment, the information systems were already provided, also used by others. We plugged our framework on top of the existing `Digital Shadow` in order to avoid any side effect on it. In our implementation, the `Digital Twin` is responsible for the creation of as many graph instances corresponding to the different referential files that exist in the data. Referential files contain actual definitions of the bus network structure, as defined by the operator. i.e., if there are four sets of referential files, each of them defining a bus network that is valid for a given period of time, four different `graphs` instances will be created.

3.1 Graph and paths adaptation

We specialized DataTime for bus networks issues as shown in the bottom part of Fig. 1. The bus network is an extension of the class `Graph`, for which the longest path within it are `BusLines`, made of `Sections` that are an ordered collection of `inter-stations` (Edges). The class `Vertex` is representing the bus stops (stations) in this context. All of the elements of the graphs are immutable (but not the graph itself), in order to keep data consistency when one wants to make, e.g., an analysis over a line/section/inter-station that exists in all the members of a set of graphs. The graph itself being mutable, it is possible to create new (immutable)

elements inside it, such as new inter-stations, bus stops, lines, etc... for strategical purposes such as the building of a new bus line.

3.2 Features adaptation

`inter-stations` contain `Features` such as length of the inter-station (in meters), type of road (one-way, two ways, reserved for buses, etc.), number of traffic lights, number of pedestrian crossings, etc. All those features are physical features, extracted from OpenStreetMaps[1] Section and line features are obtained by aggregating their respective subpart features. In order to manage features in the bus network implementation, we created `data instances` that extend the abstract classes from the package `data` (following the type-object design pattern). We then created several data features classes (e.g., `Length`, `TrafficSignalsCount`, etc.), one `VolatileFeature` class, called `LineType` and as many `Metrics` classes as needed amongst which `Speed`, `TravelTime`, etc.

3.3 Runs adaptation

If we take a look at `Run`, the data they correspond to in the `Digital Shadow` is the data gathered from bus trips all over the network. We describe trips as follows:

- Trip in edges: Bus trips from bus stop `origin` to bus stop `destination`;
- Trip in sections: Bus trip from an `origin` bus stop to a `destination` bus stop, within its ordered collection of edges;
- Trip in lines: Bus trip from an `origin` bus stop to a terminus bus stop (`destination`), a line is composed of an ordered collection of sections.

Trips contain metrics such as start time, arrival time, travel time, speed, dwell time (for sub paths only). One could easily add any external information such as smart card data, weather, traffic, etc.

3.4 Prediction scenario for IPTS

In public bus transportation networks, bus speed is considered to be a Key Performance Indicator (KPI) that translates the level of efficacy and attractiveness of a bus network [7–9] Consequently, bus networks operators struggle daily to maintain high bus speed and are in need of reliable, complete and flexible prospective methods to predict the performance (such as speed) of future bus lines.

Our running example is a bus network forming a directed graph $G = (V, E)$, where V is the set of bus stops and $E \subseteq V \times V$ is the set of possibles one-stop trips. Such a graph can be considered as a static graph as long as most of its structure does not evolve in the short term (few months to few years). However, its edges generate a lot of data over time. We consider a non-empty set of features F, associated to each edge, with their corresponding types T_F. A typical set of features associated to each element of E is $F = (time, line, length, road - type, bicycle)$, where $time$ is a timestamp, $line$ is an integer denoting a bus line number, $length$ is the length of the inter-stop section, $road - type$ indicates the kind of road the bus runs on (dedicated road or not), and $bicycle$ indicates whether bicycles are allowed. Figure 5 gives an overview of a bus network graph in which green vertices are departure terminals, blue vertices are transition/departure bus stops, white vertices are transitional bus stops and red vertices are ending terminals.

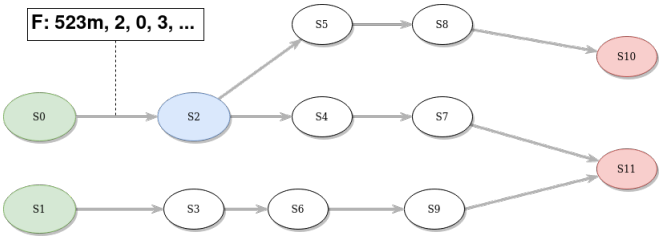

Figure 5: A bus network with featured edges (F: length of the road, type of line, etc.)

Let us consider a measure $\mathcal{M}$ of the graph, defined on paths p in G. Let $\mathcal{D}$ be a learning dataset of examples of $\mathcal{M}$. Let us consider a decision problem in the graph, e.g., rerouting due to roadworks. Given an origin and a destination vertex, two approaches can be envisioned:

- **Classical prediction scenario**: enumerate possible path p from origin to destination and predict $\mathcal{M}(p)$ for a given timestamp, and choose the best option.
- **What-if prediction scenario**: starting from the origin, make a local prediction using the micro-model on outgoing edges, choose the best option and continue in a step-by-step greedy search (e.g., minimizing or maximizing a target variable along the path) towards the destination. The prediction on the new path p, $\mathcal{M}(p)$, is the composition of these micro-predictions. We can also apply other shortest path algorithm such as Dijkstra. Figure 6 gives an

[1] https://gitlab.inria.fr/glyan/osm_bus_extractor

example of such a task, where we want to predict the duration for a given timestamp of trip $S0 - S10$ through $S0 - S5$ and $S5 - S7$ (detours), edges that do not exist in the example dataset.

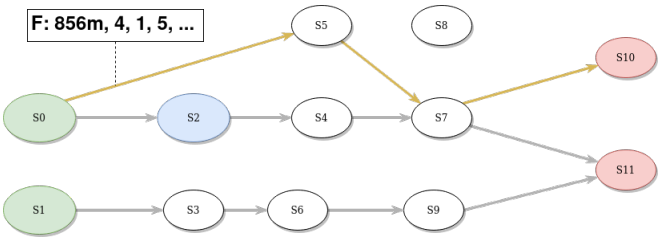

Figure 6: What-if scenario: predict $S0 - S10$ while edges $S0 - S5$, $S5 - S7$ and $S7 - S10$ do not exist in the dataset

3.5 Data mass and lazy loading

Since an IPTS typically produces several GigaBytes of data per month, we soon end up with too much data for a local file system. We thus need a kind of lazy loading mechanism when querying trip data. Accordingly, runs are "built" on demand when a call is made through the method `getMetric()` from the `!DigitalTwin`. The resulting trips can be kept in memory up to the maximum available memory, but not saved. An advantage of lazy loading is that this allows the ingestion of data in pseudo-real time (e.g., when there is a need to observe what is going on in the bus network within a short delay).

4 Experimentation at Keolis Rennes

4.1 The Keolis Rennes Bus Network

Keolis Rennes is the company that manages the Urban Public Transportation Network (UPTN) of the city of Rennes, France. This network, named STAR[2], is based on a central subway line, and a wide bus network that serves both the city of Rennes and all the suburban areas. In total, the bus network covers more than $550km^2$ The company manages a total of 116 bus lines over which more than 600 buses can be traveling during rush hours. The bus network information system is made of several sub systems including an Automatic Vehicle Location (AVL) that yields large amount of fine-grain data (inter-station) both in real and delayed time. These data contain timestamped and located information such as bus speed, travel time, dwell time, etc. The data we gathered for 2 years uses more than 40GB.

[2] cf. `https://www.star.fr/`

4.2 DataTime in Practice at Keolis

Our DataTime implementation was designed to be used by bus networks operators. Thanks to the framework we could develop the following features:

- The creation of new elements over the bus network, i.e., bus lines (or new bus lines sections), bus stops, inter-stations.
- The analysis of any event on the network at any place and anytime in the bus network. For instance lines 1 to 20 in Algorithm 1 access past data using the method `getMetric()` with a time period located in the past
- The prediction of any metric on the network at any place and anytime in the bus network, e.g., Algorithm 1 lines 21 to 29 where the method `getMetric()` is call with a parameter meaning *now*
- Providing the operators hints on which detour should be applied on a bus line depending on, e.g., the expected speed of this detour. Even in a complex bus network with multiple possible paths between two bus stops, finding alternative possible routes is quite standard. It is typically solved with a search based greedy algorithm, parameterized with a maximum number of paths of a maximum length. The complex part is of course *evaluating* the suitability of the possible routes that the search algorithm would yield. Depending on the goal of the network operator (e.g., smallest travel time, best reliability, etc.), the system would then just have to pick the best path among the existing ones. As long as the system is able to predict a metric (speed, reliability) on any known or unknown edge, along any path, compositional prediction makes it possible to aggregate those predictions on any number of different paths. The operator can then choose the best detour, given one or more specific heuristics.
  Algorithms 2 and 3 summarize the main steps of putting everything together to automatically apply a bus detour if the bus operator asks the system to. Algorithm 2 explains how to find all possible paths between two vertices in the graph, with some constraints on the number of iterations and paths found size. Algorithm 3 shows how to call algorithm 2 to yield an optimized deviation for a bus line, searching to maximize the speed of the deviation.
  In particular, the algorithm 2 parameters are: an origin and destination bus stop, the max length of each output path and the maximum number of detours

that should be generated. We use a set of candidates as a return value to avoid duplication of candidates. Next, the overall idea is to build candidates over every output edges of the origin bus stop, within the limits of the ending conditions maxLength (per candidate) and maxDetours (global).
The recursive function aims at building single candidates. Its parameters are :

- currentEdge: The starting edge for the current exploration step
- dest: The destination node of the exploration
- order: Used to check the current candidate's length against maxLength
- maxLength, maxDetours, detoursSet : Transferred from the calling function (same semantics)
- edgesSet: Used to avoid cycles in a path

The overall principle is to explore the graph until

1. The destination node is reached and the length of the candidate is under maxLength (if maxLength is reached, the candidate is not selected)
2. The maximum number of detours has been reached

Algorithm 1: Different call examples over the getMetric method

```
1: val busLine:Line = Graph.getLine("152", Direction.B)
2:
3: // TimePeriod for which historical data exists
4: var tp:TimePeriod = TimePeriod("2019-01-01
   08:00:00","2021-02-01 00:00:00")
5: var metric:Metric = Speed
6:
7: // Will return all the runs found for line 152 over tp
8: var runs:Set[Run] = getMetric(busLine,tp,metric)
9: runs.foreach.displayMetric()
10:
11: // TimePeriod for real time Data
12: val now:Long = System.currentTimeMillis
13: tp.setPeriodStart(now)
14: tp.setPeriodEnd(now)
15:
16: /** Will return runs corresponding to the last data
   available
17: * in the digitalshadow, comparing with now
18: */
19: runs = getMetric(busLine.getSections.head,tp,metric)
20: runs.foreach.displayMetric()
21:
22: // TimePeriod for future date
23: tp.setPeriodStart("2022-03-19 17:30:00")
24: tp.setPeriodEnd("2022-03-19 17:30:00")
25: metric = TravelTime
26:
27: // Will seamlessly return a predictedRun
28: runs = getMetric(busLine.getinter-station(5),tp,metric)
29: runs.foreach.displayMetric()
```

5 Validation of the predictive model

The DataTime framework relies on its predictive model to allow querying future states, at any scale within the graph. The very core of the predictive model is the compositional prediction (aka. compred) mechanism based on micro-learning over graph edges. This approach allows us to propose what-if scenarios on paths that are not known a priori (e.g., to optimize detours in the case of slow bus lines due to events such as road works, which is a shortest path approximation problem). However, DataTime also support predictions on paths that are known a priori and that therefore do not necessarily require a compositional approach as a macro prediction model could be used. We demonstrate the consistency between the two approaches.

The main advantage of the compositional approach is that predictions are based on the underlying structure of the graph, rather than on macro-observations on it. As shown in our running example describe in subsection 3.4, it yields a large flexibility in predictions, without domain specific knowledge (e.g., in the bus network context, multi-agent simulation requires complex modeling hand-made tuning [18]).

But this method can also have its drawbacks. When an edge measure is missing, a strategy has to be deployed to fill the missing data, and this is highly related to the amount of describing features on the underlying graph. Also, the cost of building many models for each edge could be tremendous, regarding its macro-level counterpart. Finally, and more importantly, as each model bears its own approximations, errors may sum up along the compositional law $\mathcal{C}$, giving a potentially unusable prediction.

5.1 Research Questions

We identify the following research questions:

- RQ1: Does micro-learning ensure satisfying prediction accuracy ?
- RQ2: Are a rich dataset and its associated feature set mandatory to fill missing edge observations in the dataset?
- RQ3: Does the compositional prediction approach compete with the macro predication method (that apply in different contexts)?
- RQ4: Can usable predictions (wrt. quality) be obtained with the compositional prediction strategy for the step-by-step, what-if scenarios?

In the following experiments, we evaluate and discuss these questions in depth, on two sets of data. The first set comes from a synthetic bus network simulation,

Algorithm 2: Detours finding (Scala inspired pseudo-code)

```
/** Call function **/
def detours(orig, dest, maxLength, maxDetours):
val detoursSet = new Set[Set[Edge]]()
/** Get the edges starting from orig **/
val possibleEdges:Set[Edge] = GRAPH.edges.filter(_.orig == orig)
for (e: Edge in possibleEdges) do
    val edgesSet = new Set[Edge]()
    _detours(e, dest, 1, maxLength, maxDetours, edgeSet, detoursSet)
end for
return detoursSet
end detours

/** Recursive function **/
private def _detours(currentEdge, dest, order, maxLength, maxDetours, edgesSet, detoursSet):
if (detoursSet.length < maxDetours and order < maxLength) then
    /** Non duplication check **/
    val vertexInSet:Boolean = if (edgesSet.isEmpty) false
    else edgesSet.map(_.orig == currentEdge.dest).reduce(_||_)
    if (currentEdge.dest == dest and not vertexInSet and edgesSet.add(currentEdge)) then
        /** FINAL CASE **/
        detoursSet.add(edgesSet)
    else if (not edgesSet.contains(currentEdge) and not vertexInSet) then
        /** GREEDY CASE **/
        val possibleEdges:Set[Edge] = GRAPH.edges.filter(_.orig == currentEdge.dest)
        /** This loop should be parallel for better performance **/
        for (e:Edge in possibleEdges) do
            edgesSet.add(currentEdge)
            if (not edgesSet.contains(e)) then
                _detours(e, dest, order + 1, maxLength, maxDetours, edgesSet.clone, detoursSet)
            end if
        end for
    end if
end if
end _detours
```

where timing data is 100% correct by construction. The second set is real data coming from a real bus network: this data being quite imperfect as it is most often the case for real bus networks [17, 24].

5.2 Experimental environment

All experiments were run on a computer equipped with 16 cores 32 thread @ 3500MHz, 64GB DDR4 3200MHz 1TB SSD NVMe, running Ubuntu 20.04 (Budgie flavour). Data were gathered into a normalized data lake built upon Apache Spark 2.4.5/Scala 2.11.12 jobs. All learning tasks were performed using GraalVM 20.2 and Scala SMILE 2.5.3. It took around 70 minutes for overall model training and 2 minutes for each of the following predictions.

5.3 Experiments

5.3.1 Experiment on Synthetic Data

We created a data synthesizer that generates an artificial bus network based on real world features evenly distributed over it using uniform random generation. Using this code, we created an artificial bus network made of 2000 inter-stops (edges) representing 110 different bus lines, each made of 12 to 43 inter-stops. Hence, some inter-stations are shared between bus lines. Note that we do not need to generate nodes as long as nodes features are not used in our model which is edge based (nodes are virtually represented by the fact that edges are connected to each other within bus lines).

We then generated 25 to 150 virtual bus trips for each of those lines, generating a target measure for each edge crossed in the trip. We used a negative exponential based function that takes edges features into account (a typical modeling in bus networks).

Algorithm 3: Example of how to get the best detour on line A2 between A and B for the morning rush hour of Tuesdays during working periods (Scala inspired pseudo-code)

```
1: val line:Line = GRAPH.lines.A2
2: val detourStart:Vertex = A
3: val detourEnd:Vertex = B
4:
5: val originalRoute:Set[Edge] = line
6: .getSubRoute(detourStart,detourEnd)
7:
8: val day = Days.TUESDAY
9: val holidays = Holidays.NONE
10: val period = Periods.MORNING_RUSH_HOUR
11: val metric = Metrics.SPEED
12:
13: val possibleDetours:Set[Set[Edge]] =
14: detours(detourStart, detourEnd, 20, 10)
15: .except(originalRoute)
16:
17: val metrics:Set[Double] = possibleDetours
18: .map(_.predict(metric,TimePeriod(holidays, day, period))
19:
20: val bestDetour:(Set[Edge],Double) = possibleDetours
21: .zip(metrics).sortBy(_._2).last
22:
23: GRAPH.enact(line,bestDetour)
```

$$V_c = (V_0 + V_0'\delta D + V_0''\delta P + V_0'''\delta L) \; e^{-[(\alpha+\alpha'\delta fp)fp+(\beta+\beta'\delta tp)tp+(\gamma+\gamma'\delta fs)fs+(\sigma+\sigma'\delta ts)ts]} \tag{1}$$

Equation 1 represents the commercial speed equation proposed by [9]. Because of the efficacy of this equation, and the fact that it is easy to tune to make it close enough to the reality, we base our data generation equations on it, yielding Equations 2 and 3 .

Table 1 Exposes the variables we use in the equations, and the range of the value they can randomly take. Each coefficient in the equations are manually tweaked in order to obtain values that yield consistent results. Once done, and for each scenario (travel time and risk), we generate a value μ using the corresponding equation. We then apply it as the mean of a normal law for which we generate a random standard deviation σ, corresponding to a random value situated between 2.5 and 15% of the mean μ we generated earlier. We then use this normal law $normal(\mu, \sigma)$ to yield random values for our datasets.

$$TravelTime = 35 * e^{-(0.03*d+0.02*h+0.05*p+0.0002*l+0.002*s+0.001*c+0.0002*g+0.0003*ra+0.007*st+0.01*ts+0.0002*pb-0.0001*pc+0.0001*po+0.0002*ps)} \tag{2}$$

$$Risk = 1 * e^{-(0.00002*d+0.00002*h+0.000035*p+0.000002*l+0.000005*s+0.000003*c+0.000005*g+0.000003*ra+0.000002*st+0.000005*ts+0.000002*pb-0.000001*pc+0.000001*po+0.000002*ps)} \tag{3}$$

Table 1: Variables of our equation used for data generation

Variable	Meaning
d	day of the week $\in [0, 6]$
h	holiday $\in [0, 1]$
p	period in day $\in [0, 2]$ with 0=free, 1=avg traffic, 2=rush hour
l	length of the inter-stop in meters $\in [150, 600]$
s	maximum speed $\in [30, 55]$
c	number of crossings $\in [0, 4]$
g	number of giveways $\in [0, 3]$
ra	number of roundabouts $\in [0, 2]$
st	number of stop signals $\in [0, 1]$
ts	number of traffic signals $\in [0, 3]$
pb	proportion of road shared with bikes $\in [0.0, 1.0]$
pc	proportion of bus corridor $\in [0.0, 1.0]$
po	proportion of one way road $\in [0.0, 1.0]$
ps	proportion of slow zone $\in [0.0, 1.0]$

Doing so we generated two datasets: one for which measure is the travel time in seconds. Its compositional result is obtained by summing the travel time of each edge; another one for which measure is inter-stop reliability (i.e., a real value between 0 and 1, 0 being unreliable and 1 totally reliable), with its compositional rule being made of product of edge reliability.

5.3.2 Experiment with Real Data

We considered the bus transportation system of the French city of Rennes and its metropolitan area (about 500,000 inhabitants) managed by the Keolis Rennes company. Our graph contains 2110 bus stops and 116 bus lines, and the features of road portions are automatically extracted from OpenStreetMap (OSM in the sequel). As experimental candidates, we chose 4 groups of bus lines for which we could gather data all along their path. We tested 13 different bus lines in total. The four groups contain different class of bus lines: urbans, inter-district, metropolitans and express. We considered only real data: no imputation was made in our dataset. We considered only full trips for which all data points were present. We also filtered historical data used to feed the model, dropping invalid speeds/nulls/values and outliers. Using these rules, we gathered 12 month of data between January 2019 and December 2019, for a total of around 15 millions tuples.

5.3.3 Prediction task

We targeted the prediction of the speed of a bus line depending on time (holidays period, day, time of the day) and line type (urban bus line, metropolitan bus line). We considered predictions for existing lines (classical prediction) and non-existing lines (what-if scenarios, where the lines and some inter-stops data are absent in the training dataset).

5.3.4 Micro-learning dataset

The micro learning dataset is made of measures for each inter-stop of the network. We simply identified every distinct inter-stop and added meta data such as holidays, bus line type, OSM data (that is constant through time), period of time in day and week day.

5.3.5 Macro-learning dataset

The macro learning dataset is built using the raw dataset, from which we extract only the overall speed of every complete trip (i.e., trips for which we have data for each inter-stop) of every bus line. We then add the bus line, its type and finally the OSM data aggregated all along the line (i.e., total number of traffic signals, stops, etc.). In order to keep as much data as possible, we did not use traffic and ridership data due to the fact that they cover smaller areas and periods in time.

5.3.6 Learning method and validation methodology

Our models are built using a classical random forest algorithm parameterized for regression and optimized using grid-search (this choice is motivated by its generality and overall performance, after selecting it amongst others: Lasso, OLS, SVR, Cart, bayesian ridge and gradient boosting, testing their performance using a small sample of data. It could be naturally adapted to more specific methods, but this is not our target here). For micro-learning, we build a model of inter-stop speed. We then predict the speed of any trip by the natural compositional prediction $\mathcal{C}_s$ (summing the road lengths divided by the summed travel times, obtained with the predicted speeds). For macro-learning, we build a model of full bus-line speeds (from start to stop). Learning is performed with OpenStreetMap features, and without OpenStreetMap features on every bus lines in order to assess the impact of these features on the model precision. For validation purposes, we have removed all the data of candidates bus lines from the training dataset of each model. To evaluate the classical scenario, we predicted the speed of an entire line with macro and compositional prediction. For the what-if scenario, we measured the compositional prediction quality on non-existing paths in the dataset.

5.4 Evaluation

5.4.1 Micro-learning Accuracy (RQ1)

The experiment we led using artificial data yielded the following results:

Tables 2 and 3 show that predictions for our synthetic datasets, the accuracy of our micro-model on edges is coherent for both speed and reliability prediction. But how would it behave with real, imperfect data that is typical of bus networks?

Accuracy We tested (predicted) ∼12'500 trips for the urban bus lines group, ∼1'250 trips for the inter-district group, ∼5'800 for the metropolitan group and ∼800 trips for the express group. All those trips are distributed over the 6 holidays regime, 7 days and 17 periods in day over the 2019 year. Figure 7 shows the predicted speed of one of the urban bus lines using micro-learning. Each point represents the average predicted speed (purple) and average real speed (dashed) of each inter-station of the line on all the time periods for which data was available. The leftest point of each curve represents the beginning of the bus line, the rightest one being the last stop. Observe that due to the bus line's data deletion from the training dataset, some inter-stops that are specific to this line are totally unknown to the model (orange dots). We also considered metropolitan, express and inter-district bus lines (not displayed)[3]. Visually at first, we can see that micro-learning captures well the daily behaviour of buses, even when the inter-stops are unknown to the model (orange dots). Table 4 sums up the model's prediction errors for each bus line, group of lines (urban, inter-district, metropolitan and express) and all the lines (global). For each model and bus line (resp. group of lines), the table presents Root Mean Squared Error (RMSE) in km/h (eq. 4), Mean Absolute Error (MAE) in km/h (eq. 5) and Mean Absolute Percentage Error (MAPE) (eq. 6) in percent. RMSE emphasizes large residuals (outliers) while MAE and MAPE are less sensitive to residuals and easier to interpret [22].

[3] More material is available in the public deposit https://gitlab.inria.fr/glyan/compred

Table 2: Prediction accuracy on a synthetic dataset for additive composition law (bus line duration)

Prediction type	RMSE	MAE	MAPE	% known inter-stops
Micro	4.78s	0.12s	7.49	69.04
Macro	62.36s	51.96s	3.76	69.04
Compred	25.36s	3.23s	1.75	69.04

Table 3: Prediction accuracy on a synthetic dataset for multiplicative compositional law

Prediction type	RMSE	MAE	MAPE	% known inter-stops
Micro	0.019	0.001	1.50	65.17
Macro	0.073	0.012	7.69	65.17
Compred	0.072	0.006	7.63	65.17

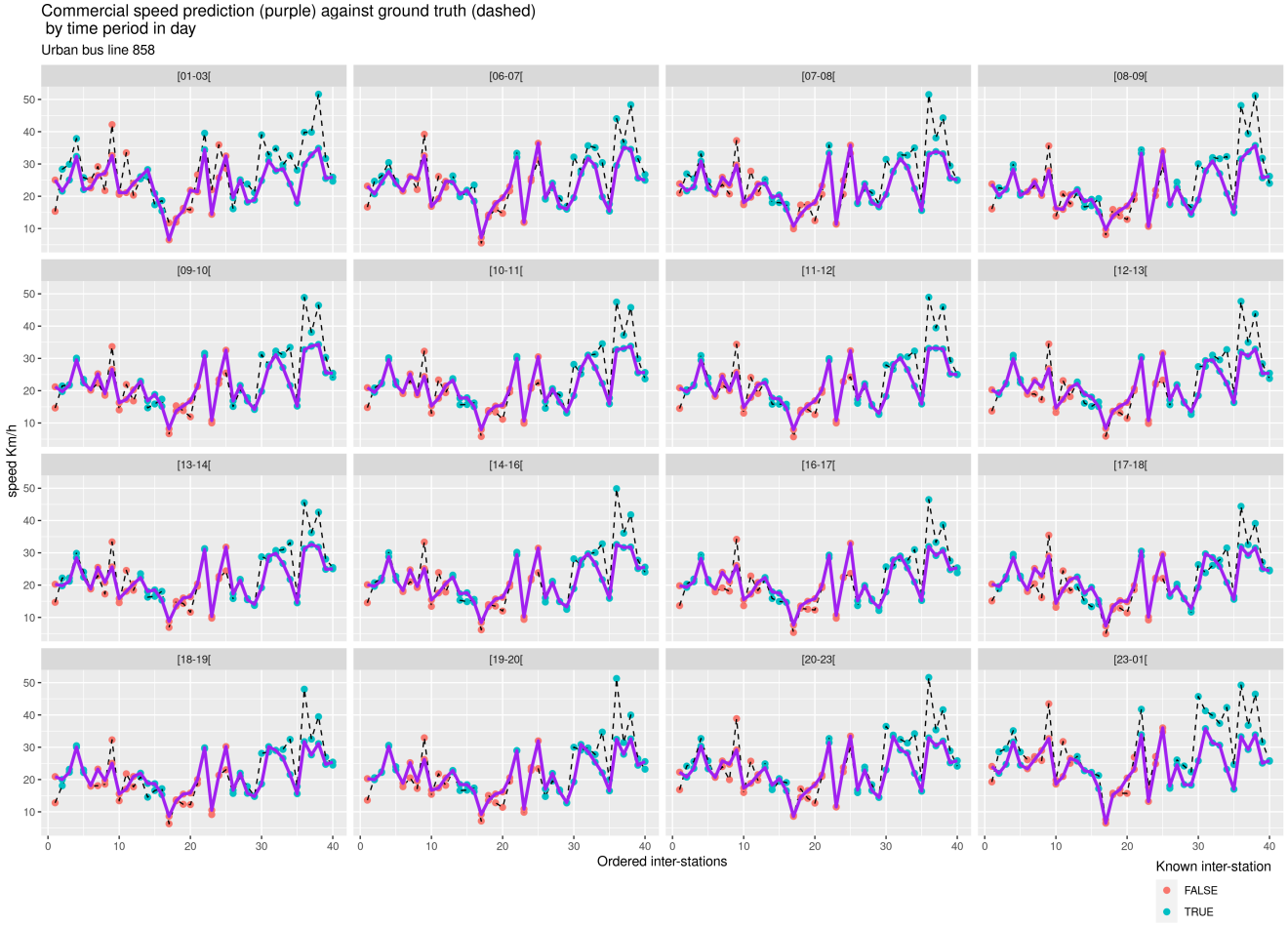

Figure 7: Speed of urban bus line 858 at inter-stations using micro-learning (purple) vs true speed (dashed). Orange inter-stations were not used by any other bus, i.e., are absent from the training set.

Table 4: Results table

Line	Micro known			Micro unknown			Micro global			Compred			Macro		
	RMSE	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**
Urban lines															
858	7.3	5.0	18.2%	6.1	4.5	25.9%	6.9	4.8	22.4%	1.6	1.2	6.2%	2.1	2.0	8.4%
787	5.6	3.9	19.9%	9.1	7.3	44.4%	6.0	4.2	22.2%	1.7	1.3	7.2%	2.2	2.0	9.7%
785	5.9	4.5	22.7%	8.3	6.0	29.2%	7.5	5.4	26.6%	2.6	2.0	9.9%	2.7	2.5	10.4%
889	5.6	4.3	31.1%	8.7	5.9	25.4%	6.3	4.6	30.1%	3.0	2.7	17.4%	2.7	2.6	15.4%
689	6.3	4.6	29.4%	8.2	6.7	53.9%	6.4	4.7	30.7%	3.2	2.8	16.5%	2.4	2.3	12.2%
All urbans	6.3	4.6	25.2%	7.5	5.4	29.4%	6.7	4.8	26.5%	2.6	2.1	12.0%	2.4	2.3	11.5%
Inter-District lines															
696	8.9	6.4	27.1%	9.3	7.4	46.2%	9.0	6.7	33.5%	3.2	2.8	13.7%	4.0	3.8	15.5%
581	6.4	5.1	26.7%	10.3	7.6	31.8%	7.6	5.7	27.9%	3.5	2.8	12.3%	3.9	3.5	13.4%
All inter-Districts	8.8	6.4	27.0%	9.3	7.4	45.9%	9.0	6.7	33.3%	3.2	2.8	13.7%	4.0	3.7	15.5%
Metropolitan lines															
683	5.0	3.8	15.5%	8.4	6.7	26.9%	7.1	5.4	21.8%	3.2	2.5	10.3%	3.1	2.9	9.5%
849	6.3	4.7	37.4%	5.0	4.0	11.9%	6.3	4.7	36.6%	3.2	2.8	12.1%	3.7	3.5	14.3%
532	7.9	5.7	24.2%	12.1	11.9	427%	8.2	6.0	45.4%	4.0	3.3	13.7%	4.8	4.5	15.5%
969	7.2	4.8	25.3%	6.0	4.4	16.1%	7.1	4.8	24.4%	3.6	2.6	9.8%	3.5	3.2	9.9%
All metropolitans	6.7	4.8	25.6%	8.6	6.8	48.3%	7.3	5.4	31.7%	3.5	2.8	11.4%	3.8	3.5	11.9%
Express lines															
711	5.8	4.2	36.6%	4.5	3.6	10.1%	5.6	4.1	32.2%	3.6	2.7	10.3%	7.0	6.9	21.1%
859	6.1	4.2	27.8%	7.9	5.5	21.9%	6.6	4.6	26.1%	7.3	5.6	24.7%	7.0	6.8	20.2%
All express	5.9	4.2	33.4%	6.5	4.6	16.4%	6.1	4.3	29.8%	5.3	3.8	15.6%	7.0	6.8	20.8%
All lines															
All lines	6.6	4.7	25.5%	5.7	7.8	33.7%	7.0	5.0	28.0%	3.0	2.4	12.1%	3.2	2.9	12.2%

Table 5: Models's sensitivity to time features

group	Compred standard dev.	macro standard dev.
urban	1.0	0.5
inter-district	0.7	0.3
metropolitan	0.9	0.4
express	0.9	0.0
global	0.9	0.4

$$RMSE = \sqrt{\frac{1}{N}\sum_{1}^{N}(Y_t - \hat{Y}_t)^2} \quad Y_t = actual, \hat{Y}_t = predicted \quad (4)$$

$$MAE = \frac{\sum_{1}^{N}(Y_t - \hat{Y}_t)}{N} \quad Y_t = actual, \hat{Y}_t = predicted \quad (5)$$

$$MAPE = \frac{1}{N}\sum_{1}^{N}\left|\frac{Y_t - \hat{Y}_t}{Y_t}\right| * 100 \quad Y_t = actual, \hat{Y}_t = predicted \quad (6)$$

RMSE and MAE unit is km/h while MAPE's unit is percentage (difference against truth in percents). If we take a look at micro predictions in Table 4, we observe that prediction errors for known inter-stations (classical prediction) are better than prediction errors of unknown inter-stations (what-if predictions), except for the express bus lines group. Indeed, RMSE, MAE and MAPE can vary from a 1.2 to 2 factor between micro known and unknown. This shows that the features set used for micro learning might need some enhancement so that the modeling of inter-stations has better fidelity. Finally, we observe that global micro predictions precision vary between bus lines groups. However, the urban group gets the best predictions in all aspects when compared to other groups (RMSE: 6.7, MAE: 4.8, MAPE: 26.5%), followed by express (RMSE: 6.1, MAE: 4.3, MAPE: 29.8%), metropolitan (RMSE: 7.3, MAE: 5.4, MAPE: 31.7%) and inter-district (RMSE: 9.0, MAE: 6.7, MAPE: 33.3%) groups. Urban bus lines are the major lines in the network, hence they produce a significant part of the data used for training. This can explain why the models are more precise when they predict speed for this group. We noticed a huge prediction error for metropolitan bus line 532 with resp RMSE, MAE and MAPE at 12.1, 11.9 and 427%. We investigated how can the model perform that bad on some cases and found that the average speed of the only unknown inter-station of the bus line 532 is lower than 5km/h, while its features are alike others inter-stations in the network. The speed calculation business rules used might yield such a low speed, e.g., if the bus has to wait at a bus stop for operational reasons. In other words, inter-stations that have very low average speed with non specific features set is prone to yield important predictions errors (i.e., considered as an outlier). Globally, micro-learning yields variable quality results, with a higher error rate for unknown inter-stops. The features used for predictions probably need enhancement in terms of noise reduction and/or external data addition such as smart-card data, traffic status, etc.

Finally, the quality of the predictions evaluated by the different indicators (RMSE, MAE, MAPE) is considered as satisfying by the Keolis operators considering the ex-nihilo aspect of some edges.

5.4.2 Sensitivity with respect to features (RQ2)

We checked the models sensitivity to OpenStreetMaps features and time features by training models with OSM-less datasets in order to assess how well those features describe the bus network. A global features analysis has been done in the PhD thesis work that yielded this paper [14]. Table 6 shows the results of the predictions made without OSM features, hence using time discretization and built-in features only (period, day, holidays, line type, dwell-time). The values in table 6 (results without OSM features) are colored to compare them with table 4 (results with OSM features): Black for identical results, green for better results and red for worse results.

We observed that, in most of the cases (i.e., individual bus lines and groups), OSM features play an important role for speed prediction for micro, compositional and macro prediction. It seems that micro prediction has to be done using OSM features[4]. Indeed, the error is often 1.5 or more than 2 times larger when compared to results of Table 4. Table 7 shows the speed prediction variation of micro-learning model with and without OSM features. Clearly, the micro-learning model trained with them is way more capable of predicting different speed for different inter-stations, with a standard-deviation of predicted speeds that is nearly three times higher than the micro-learning model trained without OSM features. However, predictions for unknown inter-stations of lines 787 and 683 are way better without OSM features than with it. Moreover, there are cases in which the removing of OSM features delivers better results for compositional and macro prediction, and not by a margin (more than 5%). This is the case for metropolitans bus lines and inter-district bus lines (Compred mostly).

[4] Charts of micro-learning results with and without OSM features are available in the public repository for a more visual feedback

Table 6: Results table of models trained without OpenStreetMaps features

Line	Micro known			Micro unknown			Micro global			Compred			Macro		
	RMSE	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**	**RMSE**	**MAE**	**MAPE**
Urban lines															
858	10.1	7.5	29.6%	8.0	6.3	49.7%	9.3	7.1	37.7%	2.0	1.6	8.6%	2.3	2.2	9.8%
787	7.3	5.6	32.8%	6.5	5.3	31.9%	7.2	5.5	32.7%	2.1	1.7	9.2%	2.3	2.0	9.1%
785	8.0	6.2	34.3%	10.2	7.6	37.7%	9.4	7.1	36.3%	2.2	1.6	7.8%	4.0	3.8	15.5%
889	7.3	6.0	51.5%	8.5	6.5	30.0%	7.5	6.1	47.6%	5.8	5.5	35.0%	4.2	4.1	25.2%
689	8.5	6.5	46.5%	8.8	7.7	76.0%	8.5	6.6	48.1%	3.9	3.7	21.1%	1.5	1.4	6.8%
All urbans	8.6	6.6	40.7%	9.0	6.9	42.9%	8.7	6.7	41.3%	3.9	3.2	18.8%	3.2	2.9	14.8%
Inter-District lines															
696	11.4	8.6	35.2%	12.0	10.1	69.7%	11.6	9.1	46.7%	2.6	2.2	10.8%	3.8	3.5	14.6%
581	8.5	6.9	36.0%	14.7	10.6	36.1%	10.4	7.9	36.0%	2.6	2.1	8.6%	7.4	7.0	26.6%
All inter-Districts	11.3	8.6	35.2%	12.1	10.1	68.9%	11.6	9.1	46.3%	2.6	2.2	10.7%	4.0	3.7	15.1%
Metropolitan lines															
683	9.1	6.9	25.3%	6.1	4.9	20.6%	7.6	5.8	22.7%	2.3	1.7	6.9%	3.2	3.0	9.8%
849	11.3	8.8	107%	8.5	7.4	21.5%	8.7	11.3	104%	3.9	3.5	15.0%	2.0	1.8	6.8%
532	13.8	10.9	49.0%	24.0	23.8	838%	14.6	11.6	90.4%	2.5	1.9	7.8%	3.0	2.8	9.1%
969	14.4	12.0	68.5%	8.5	6.4	20.4%	13.9	11.5	64.1%	4.6	3.9	12.8%	6.4	6.0	18.2%
All metropolitans	12.1	9.4	60.3%	8.3	6.0	65.7%	11.3	8.5	61.8%	3.1	2.4	9.1%	3.7	3.2	10.5%
Express lines															
711	10.0	8.7	69.1%	10.0	9.3	26.1%	10.0	8.8	61.9%	5.3	4.6	15.7%	7.2	7.0	21.7%
859	8.9	7.6	50.4%	9.7	7.9	25.0%	9.1	7.7	43.3%	6.4	5.3	19.6%	7.9	7.6	22.8%
All express	9.6	8.3	62.2%	9.8	8.6	25.5%	9.6	8.4	54.4%	5.7	4.9	17.1%	7.5	7.2	22.1%
All lines															
All lines	9.6	7.3	44.5%	9.1	7.0	48.5%	9.5	7.2	45.7%	3.7	3.0	15.5%	3.6	3.2	13.9%

Table 7: Micro-learning models'sensitivity to OSM features

group	prediction variation
with OSM features	7.5 km/h
without OSM features	2.8 km/h

Impact of the size of the training dataset on the precision of the compositional predictive model: For all measures, the precision obtained for the prediction is considered to be satisfactory from the point of view of the bus operator (ie, Keolis Rennes). All tuples devoted to the training data set are used, which represents about 14.5 million tuples. Although training the predictive model with this amount of data can be done in just over an hour on a standard workstation. It is interesting to explore whether such a large amount of data is necessary to achieve sufficient accuracy. We considered the impact of the reduction of the training dataset on the accuracy of the composition prediction model. Figure 8 shows that a few hundred or even tens of thousands of tuples allow training of a predictive model. It yields compositional prediction results that are close to those that are obtained using the entire dataset. These results can be applied in the context of continuous learning of the model. Thus, a component could periodically compute the optimal training dataset's size and open the way to operate a continuous learning using a sample of the new data only.

Micro-learning on graphs at edge level (e.g., bus-inter-stops level) yields consistent results on synthetic data. The accuracy turns out to be variable on a real dataset. The prediction error tends to be more important when there are no observations on this particular edge available in the learning dataset. These results point the importance of choosing the right features set for the model. However, Figure 7 shows that the micro learning model has a sensitivity to current features, this allows for what-if scenarios, when one wants to predict the behaviour of a new unseen path.

It seems that the addition of OSM features in the training datasets helps the models to link the time features and speeds to the network topology, thus improving the prediction precision. On the other hand, an effort has yet to be made to explain the reasons of the enhancement observed on some bus lines' predictions when removing OSM features.

5.4.3 Compositional prediction (Compred) efficiency in comparison with macro prediction (RQ3)

Comparing compositional prediction results (Compred) with Macro prediction, raised the following remarks:

For synthetic data, Tables 2 and 3 show that the compositional prediction (Compred) performs at least as well as macro prediction (i.e., over whole line), and quite often much better. Thus, we can hope that this trend will be the same for real world data.

For urban group, Compred MAPE and RMSE differences are only of 0.5% and 0.2 km/h while Compred's MAE is 0.2 km/h better than macro's. That is to say, macro and Compred are nearly on par when it comes to predict urban bus speed. In other groups, compositional prediction performs slightly better than macro predictions, with MAE, RMSE and MAPE lower for Compred than macro by around 2.5 to +25% depending on the measure we compare.

This suggests that compositional prediction seems to be more capable of catching fine grain variations (i.e., variation on inter-stations) than macro does, as shown in Table 5. This table shows the average predicted speed variation (standard deviation in km/h) of bus lines over time (holidays, days, period).

We observed that Compred model seems to be at least twice as sensitive to time features as macro model. Thereby, it confirms the claim we made in section 5.4.1

Model prediction errors according to the path size (line size): Although the compred approach captures local variation more effectively, it is worth exploring wether it can be affected by the aggregation of the errors of each sub-prediction. It is then convenient to ensure that the compositional prediction error of an additive metric (e.g., commercial speed) do not grow significantly as the number of sub-predictions (inter-stations in a path) increases.

In Figure 9 we chose a line from the ones with the highest number of inter-stations and then randomly extracted sub-lines composed of consecutive inter-stations. The results indicate that the compred approach has a homogeneous prediction error depending on the path size. Indeed, the error does not increase with the number of aggregations. This observation strengthens the use of compred in a context of bus networks. However, variations are more visible for the smallest sub-lines. Indeed, the smaller the line, the greater the impact of a large error in one of the sub-predictions.

To predict a measure on a path, we applied compositional prediction (here, a weighted sum of the predicted measure at the edge level). The quality obtained is satisfying with respect to the macrolearning approach both on artificial and real-world data. Indeed, our method reaches state of the art performance if we compare with [28]. However, real world data results are not as good as artificial ones. We interpret this as follows: micro-learning is done for all edges, and glitches in observations may perturb all these levels. Thus, the composition may aggregate all these errors. Conversely, in macro-learning fewer models are computed which is less error-prone. But again, macro-learning does not allow

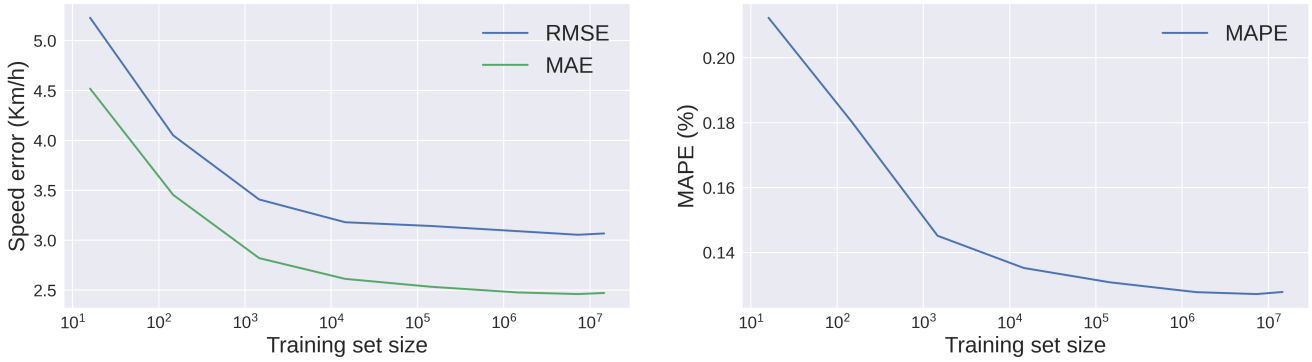

Figure 8: Impact of the size of the training dataset on the precision of the compred model (RMSE, MAE, MAPE)

Figure 9: Evolution of the error (RMSE, MAE, MAPE) of compositional predictions depending on the number of inter-stations. Each sub-line is extracted from consecutive inter-stations of line 858.

the prediction of an unknown path of any length, while compositional prediction can.

5.4.4 Quality of compositional prediction for step-by-step, what-if scenarios (RQ4)

The compositional prediction approach can be used to address what-if scenarios where decisions must be made step by step and based on local properties. In such a scenario, the quality of the micro-prediction for each inter-station is crucial.

In subsection 5.4.1 we evaluated the micro-learning accuracy on real data, these results have been validated and considered sufficiently accurate by Keolis to be used to practice experiments in the context of what-if scenarios. The operators leverage the framework for what-if scenarios, equipping them in activities such as creating new bus lines, step-by-step, using compositional prediction made on the inter-stations. Although this is not the initial objective, it is possible to leverage the DataTime framework to develop tools that address this kind of scenarios, automatically and without any human interaction.

6 Discussion

6.1 Applicability of the DataTime framework

The DataTime framework as been applied to IPTS but the approach is not limited to this domain. As we describe in 1 the DataTime framework is domain agnostic and can be derived for any application. It is suited for domains having massive graph structured-data with a spatial dimension that evolves slowly over time. In addition, the predictive model relies on compositional prediction and require that the metric of interest can be obtained on each edge, and then aggregated following a compositional law. Such a law can be as simple as an addition, as in our example.

6.2 Lessons learned of the application of the DataTime framework in practice

6.2.1 Technical considerations

In practice, real world data is dirty [21]. In our case, we gathered data on a 6 months period between early

July 2018 and late January 2019. The data collected is based on the bus network AVL system which contains 116 bus lines. We made some data quality analysis of this data in order to build data-cleaning processes if needed. Table 8 shows the summary of the raw data. As one can see, the outliers in the dataset are quite preposterous: -996 km/h and 130 km/h are impossible speeds for buses. Moreover, readings for which bus speed was lower than 1 km/h or higher than the legal speed limit of 70 km/h (which respectively are low speed limit we defined and legal maximum speed for buses in France), can represent up to an important 5.4% of the dataset.

The main dataset came along with a twin that contains more reliable data (built over technologies that are more reliable than the main dataset), nonetheless at a coarser time grain (20 seconds instead of 1 second accuracy). Hence, we curated the data by using both filtering (removing outliers) and a specific merge-join techniques. We managed to reduce the error rate to 0% while removing every outlier. We published a previous work that explains in depth the issues of these datasets and how we solved them [15].

This kind of data quality issue is recurrent in data science, the data must be cleaned up to a certain extent, after which additional cleaning is no longer helpful. The difficulty is to identify when to stop it. This has been discussed in [15].

6.2.2 Performance and scalability

We fed our implementation of DataTime using Apache Spark v3.1.1. It was behaving as a datasink, making it possible to query a datalake containing more than 2 years of filtered data, totalizing nearly 40 GB. As an example, the primo-execution of a query like the one visible at lines 1-9 in Algo.1, which consists of querying over all the trips of a bus line for a 2 years period, takes around 40 seconds on a computer equipped with a middle end 8 cores x86 CPU. If one executes this query a second time, the result is almost instantaneous provided the last request results were kept in memory. In a nutshell, the bigger the period and the longer the path, the slower the querying will be. In other words, The interfacing resp. performance and design of the `Digital Shadow` services are paramount for querying to be efficient.

The predictors, that are able of continuous learning (by automatically training new predictive models when needed), must be trained before being able to predict. The training time depends on many factors but there are 4 of them that will have a significant impact. Those are the type of prediction model, the size of the training dataset, the number of features and the hyper-parameters tuning. The latter can be the worst one if, e.g., one uses grid search to find the best set of hyper-paremeters (using `optimize=true` in the method `train(...)`). We decided to train models with a training dataset that contains 1 or 2 month of data, with a set of 8 features and the default set of hyper-parameters. These 8 features have been selected thanks to the state of the art in public transportation literature [2, 4, 9, 17–19, 26, 27] and the business knowledge of Keolis agents. The training of such a model takes no longer than 10 minutes on a middle end computer (8 cores, 16 GB RAM). That is to say the continuous training if the predictive models eventually turn dull would be done in the same time range. This is thrilling knowing that a single model is able to yield predictions for the whole system.

Finally, our implementation is scalable in these ways:

1. The digital shadow can rely on scalable databases, hence the storage and querying can be distributed amongst different machines if needed.
2. The implementation of the framework can be used on any machine and does not need a tremendous amount of computing power to analyze data or predict data: the data collection for analysis relies on the digital shadow's ability to scale, and the predictive models training time is short even on a single machine.

6.2.3 Impact at Keolis

Industrial fields that are not computer-science focused often face an issue that we could call "the data overwhelming problem". This consists of having more and more complex information systems that generate more and more data, with a small team of IT engineers who are already too busy keeping the information system healthy. Although the company's data is valuable, its exploitation is hampered by operational constraints and the difficulty of exploiting it without appropriate tools. In companies that rely on stable and well known technologies like standard relational databases and spreadsheets to analyze them, such an amount of data yields the impossibility to study large samples of data. Specifically if those companies core workforce is made of domain experts for which computers are tools and services providers only. As an example the bus network of Rennes generates nearly 2GB of data per month for the sole AVL system, and more than 10GB if we consider all the stakeholders of the bus network information system. Thus, for the domain experts of the bus network who are provided low to middle end computers, the analysis of large samples of data is quite a challenge

Table 8: datasets properties

Population	Average speed	Speed standard deviation	Minimum speed	Maximum speed	Speed error rate
16793293	20.31 km/h	11.6 km/h	-996 km/h	130 km/h	5.4%

when it is not merely out of reach. Thereby, a framework such as DataTime favors the data centralizing with a focus on scalability for data analysis. It makes it possible for domain experts to integrate past, present and future within a traditional information system containing a priori models. Moreover, DataTime could be even more user friendly by the providing of future DSLs, for, e.g., facilitating the integration of new data sources, the edition of line topologies or make data analysis more fluid, etc. In addition, DataTime allowed us to highlight characteristics of the bus networks that were yet not visible for the operator. Indeed, the use of predictive models allowed the analysis of feature importance and to do sensitivity tests for, e.g., speed and vehicle engine type. It appeared that the electrical buses that were test running on the bus line 12 for a few months were systematically slower than their combustion counterparts, which was later explained by their higher gravity center, due to the presence of the traction batteries on the buses roofs.

6.3 Threat to the validity of the predictive model validation

As internal threats, we acquired the real dataset ourselves using our own code, which may be prone to errors. But this data are used in production by the Keolis company, and has been controlled using business rules many times. Also, we used only complete data. No imputation was done. Our code uses a high-level language and state-of-the art libraries for data extraction and machine learning. As construct validity, we choose to use Random forests as ensemble methods model for regression. The choice we made was based on performance amongst a set of models tested on a small sample of data from which random forests performed better (in terms of results and computing time). However, we assumed that the models would behave in an analogous manner with a wider dataset, whereas there is a small risk that this is not the case. Finally, the scope of the validation of the predictive model section is to assess the quality of compositional prediction for composite objects in complex environment against traditional macro approach, we then considered the machine-learning model selection as a secondary issue, thus we probably could have different results with other models. Yet, choosing better models for compositional prediction is another issue that probably needs a dedicated work. As an external threat, our dataset, time frame and targets selection might have characteristics that could prevent generalization beyond these lines or Rennes Metropolis. On the other side, we gathered 15M tuples over a year from a large city common bus transportation system. The chosen lines are typical and of different kind (urban, inter-district, metropolitan and express).

The compositional prediction approach enables "what if" scenarios such as prediction on paths (lines) with edges (inter-stations) that are unknown to the model. We address such scenarios in our validation by testing our models on lines that are not in the training dataset. Nevertheless, we presented other scenarios that are not in the validation such as step by step scenarios (eg, creating new lines). This type of scenario can be assimilated to a shortest path problem for which we propose a simple fast search algorithm. Still, the greedy search is not optimal and state-of-the-art algorithms such as the Bellman-Ford algorithm, the Floyd-Warshall algorithm or the Dijkstra algorithm could be used. The validation and the search for optimal algorithm to this application (taking into account execution time, performance) can be the subject of a future work.

7 Related work

There are three main ways for obtaining predictive models for transportation networks: build an analytical model (e.g., using mathematical modeling), get it from simulations (e.g., multi-actors), or through machine learning. This holds for any kind of transportation networks, be it water, electricity or buses, but in this section we mostly consider those related to IPTS.

7.1 Analytical models

First, one can try to build an analytical model that, when adequately configured, produces quality predictions for e.g., bus speed in bus corridors (constraint environment).

Fernandez and Valenzuela [9] proposed an efficient analytical model to predict bus commercial speed anywhere on a bus network, for any kind of bus line. They upgraded a state of the art function based on exponential decay to take in account more influencing paramet-

ers such as dwell time at bus stop, passenger density or even time periods or bus technology. Their model must be precisely parameterized by tweaking weights in order to produce satisfying result regarding the reality. Thus, the use this model requires the gathering of a lot of data and domain knowledge in order to obtain satisfying results. In the same way, Valencia and Fernandez [27] developed a similar approach dedicated to bus corridors (bus lanes). Their work presents similar characteristics, pros and cons, as the model described before. Analytical models are built by and for specific issues. In these particular contexts they can be very powerful when adequately tweaked. The other side of the coin is that these models are very static, thus non-applicable to others issues without a total rethinking of their behaviour. Moreover, each model is tweaked to fit a specific situation, making its portability to other instances of the same problem difficult.

7.2 Simulation models

Various scale simulations can be used to mimic an environment and then observe how e.g., a new traffic-lights system impacts the bus travel time at a crossroads (fine grain), or simulate the traffic flow on a whole city (coarse grain).

Simulations are dynamic models that aim at providing estimations obtained from a simulated environment. The closer the environment is to the actual environment, the better the estimations are expected to be. This kind of tool is usually dedicated to a specific task at a given granularity. Indeed, simulations are usually classified in 3 different categories [4]:

- Macroscopic: Simulation of a whole city (or even more), emulating traffic flow dynamics (inspired by fluids physics);
- Mesoscopic: Simulation of a district, involving explicit treatments of intersections;
- Microscopic: Simulation of a crossroad or a few roads using multi-agents simulation with complex rules and interactions among agents.

However, these kinds of simulations ask for a lot of resources to be modelled and ran properly: The cities and agents must be manually modeled, and the computing of those simulations often involves multiple CPUs and GPUs. On the other hand, DataTime may yield quality predictions and can be deployed by a few engineers, provided they have access to accurate data.

7.3 Machine learning

Machine learning can be leveraged using different methods to predict e.g., travel time of bus lines or bus stop arrival time. Machine learning models are mostly used for travel time prediction in the literature: Altinkaya and Zontul [2] reviewed the computational models used in Urban Bus Arrival Time Prediction as of 2013. Their work covers the use of historical data and statistical models, time-series, regressions, neural networks and hybrids models. They explain that the existence of that many models in this field of research is due to the fact that predicting travel time of buses is a complex task for which no model in particular has proven its superiority yet. However, they claim that hybrid models (e.g., combine neural networks with Kalman filters) are on the roll. Moreover, they add that predictions might be better if one splits the datasets into sub-datasets in which data represents similar conditions, restricting the models prediction scope. Thereby, our micro-learning approach can be considered as a divide and conquer method following this idea.

Mendes-Moreira and Barachi [19] proposed a prediction model for networks by predicting sub parts of the networks and re-conciliate the aggregated predictions of the sub-parts with the path they are part of. They do this using a method they called Reconciliation For Regression (R4R) by weighting each sub-prediction using a constraint least square algorithm. Their results show that they reach state of the art performance for bus travel time prediction. However, it is unclear on how far from the reality their model perform without MAPE. One could also raise the following statement: the added complexity of R4R is questionable because it shows that it seems to never offer a better improvement than 3% in prediction precision when compared to other models, including simple ones such as Multivariate Linear Regression (MLR). Our method does not correct data on aggregation, yet the prediction of our predictive system are satisfying. However, the enhancing of the data quality and the concept of prediction error reconciliation must be considered when building a predictive system based on aggregated prediction.

7.4 Digital twins

Bordeleau et al. [5] suggested that models at runtime are key for implementing digital twins. Our work feeds this claim in addition to adressing the following open issues, raised by their work: 1) Models and Data, and 2) Architectural Framework for Digital Twins for the specific case of spatio-temporal models. Kirchof et al. [13] worked on the interconnectivity of digital twins, their

related information system and the cyber physical system they are the virtual image of. Their work could be used to enhance the inter-connectivity of our digital twin and digital shadow, diminishing the endeavour the designers would provide to efficiently implement DataTime.

7.5 Integrating a priori models with models learnt from data

Combemale et al. [6] proposed the conceptual models and data (MODA) framework. They aim at providing a reference for model-driven and data-driven modelling issues. Their work proposes a data-centric and model-driven approach to integrate heterogeneous models and data into a single framework for the entire life-cyle of socio-technical systems. DataTime can be considered as a practical implementation fitting the MODA conceptual framework.

Hartmann et al. [10,11] worked on temporal graphs to analyze data in spatio-temporal dimensions, which embed historical analysis and predictions. Their tool, Greycat, includes a scalable graph-oriented data model that allows the building and analysis of multiple parallel worlds (forks of graphs) in a single framework. Their model is meant to be used for fast evolving networks such as smart-grids, cyber-physical systems or IoT systems that yield a lot of data and that can physically evolve quickly. The strengths of their model are the following: their model is totally and quickly scalable with a reasonable resource needs; they embed machine learning seamlessly in order to predict events on the graph and even build what if scenarios by forking graphs, yielding new instances with inherited properties. Moreover, they took a look at the interest of applying the "divide and conquer" paradigm in order to make predictions over the smaller parts of the graph (Nodes). Their findings are that for complex data models that are made of an aggregation of smaller parts, machine learning models that are trained using fine-grained data outperform models trained with coarse-grained data (i.e. parts or whole graph). Greycat has been successfully tested on a smart-grid application in Luxembourg, and is an interesting way of thinking about the interactions between data and models at large scale on evolving environments. DataTime clearly reuses the same graph-based, micro-learning strategy as GreyCat, while being optimized towards slowly evolving graphs that are typical of IPTS, and bringing a full integration of the temporal dimension (past, present, future) on top of the existing a priori model of the network.

8 Conclusion and Perspectives

In this paper, we propose a framework called DataTime to implement models at runtime which capture the state of a socio-technical system according to the dimensions of both time and space. Space is modeled as a slowly evolving directed graph where both nodes and edges bear local and independent states (i.e., values of properties of interest). DataTime provides a unifying interface to query the past, present and future (predicted) states of such socio-technical system, hiding the complexity and scalability issues of dealing with huge time series and machine learning. We applied our framework in the context of an urban transportation system, and concretely deployed and evaluated it on the Intelligent Public Transportation System of the city of Rennes (France). We also evaluated the compositional prediction approach, that offers flexibility on the prediction scale and opens the way to simulation scenarios on paths not known a priori while offering at least comparable and often better results than the classical approach (in the case of macro-prediction on a priori determined paths). DataTime makes it possible for domain experts to integrate past, present and future within a traditional information system containing a priori models. Still up to now, using DataTime requires experts to work at the level of the chosen programming language, here Scala, which is seldom known to the IT department of an IPTS operator such as Keolis. We thus plan to make it easier to use the DataTime framework by providing a set of DSL embodying its main abstractions, thus facilitating the integration of new data sources, the edition of the network topology, or make data analysis more fluid. As future work, we envision applying our framework to other application domains that bear structural and temporal similarities with IPTS, such as smart grids, water abduction systems, or supply chains. As long as they fit the directed graph abstraction at the core of DataTime, and as long as global properties of interest could be obtained by composing atomic values associated to edges, we do not foresee any issue in specializing DataTime towards these systems.

In addition, we have found that in the case of additive metrics, the prediction errors remain homogeneous and do not tend to increase significantly as the size of the path increases (in the order of magnitude of a bus line). However, it should be interesting to reproduce the experiment on other types of metrics (multiplicative composition law ...) as well as on networks having a size of an order of magnitude larger (electrical network ...). In order to avoid that the prediction errors are ag-

gregated by composition, hybrid strategies combining a macro and micro approach could be deployed.

Finally, the current approach considers offline strategical/long term (involving human intervention) reconfiguration only. i.e., in our industrial application, reconfiguration are not done while the buses are running. It is only done offline, for example during a weekend or during a night. A further work would be to leverage tactical/short term dynamic reconfiguration (that could be done with or without human supervision) for e.g., bus clogging due to an unforeseen event. In such a scenario, the system should seek a way to quickly solve the situation and restore a minimal bus service, i.e., enhancing the resilience of the system.

References

1. Aloquili, O., Elbanna, A., Al-Azizi, A.: Automatic vehicle location tracking system based on GIS environment. IET Software **3**(4), 255 (2009). DOI 10.1049/iet-sen.2008.0048
2. Altinkaya, M., Zontul, M.: Urban Bus Arrival Time Prediction: A Review of Computational Models **2**(4), 7 (2013)
3. Amirat, H., Lagraa, N., Fournier-Viger, P., Ouinten, Y.: Myroute: A graph-dependency based model for real-time route prediction. JCM **12**, 668 (2017)
4. Barceló, J., Casas, J., García, D., Perarnau, J.: A METHODOLOGICAL APPROACH COMBINING MACRO, MESO AND MICRO SIMULATION MODELS FOR TRANSPORTATION ANALYSYS p. 24 (2005)
5. Bordeleau, F., Combemale, B., Eramo, R., Van Den Brand, M., Wimmer, M.: Towards Model-Driven Digital Twin Engineering: Current Opportunities and Future Challenges. In: ICSMM 2020 - International Conference on Systems Modelling and Management. Bergen, Norway (2020). URL `https://hal.inria.fr/hal-02946949`
6. Combemale, B., Kienzle, J.A., Mussbacher, G., Ali, H., Amyot, D., Bagherzadeh, M., Batot, E., Bencomo, N., Benni, B., Bruel, J.M., Cabot, J., Cheng, B.H.C., Collet, P., Engels, G., Heinrich, R., Jezequel, J.M., Koziolek, A., Mosser, S., Reussner, R., Sahraoui, H., Saini, R., Sallou, J., Stinckwich, S., Syriani, E., Wimmer, M.: A Hitchhiker's Guide to Model-Driven Engineering for Data-Centric Systems. IEEE Software (2020). DOI 10.1109/MS.2020.2995125
7. Cortés, C.E., Gibson, J., Gschwender, A., Munizaga, M., Zúñiga, M.: Commercial bus speed diagnosis based on GPS-monitored data. Transportation Research Part C: Emerging Technologies **19**(4), 695–707 (2011). DOI 10.1016/j.trc.2010.12.008
8. Courtois, X., Dobruszkes, F.: L'(in)efficacité des trams et bus à Bruxelles, une analyse désagrégée. Brussels Studies. La revue scientifique électronique pour les recherches sur Bruxelles / Het elektronisch wetenschappelijk tijdschrift voor onderzoek over Brussel / The e-journal for academic research on Brussels (2008). DOI 10.4000/brussels.603
9. Fernandez, R., Valenzuela, E.: A model to predict bus commercial speed. Traffic Engineering & Control **44**(2) (2003)
10. Hartmann, T., Fouquet, F., Moawad, A., Rouvoy, R., Le Traon, Y.: GreyCat: Efficient what-if analytics for data in motion at scale. Information Systems **83**, 101–117 (2019). DOI 10.1016/j.is.2019.03.004
11. Hartmann, T., Moawad, A., Fouquet, F., Traon, Y.L.: The Next Evolution of MDE: A Seamless Integration of Machine Learning into Domain Modeling. Software & Systems Modeling volume p. 17 (2019). DOI 10.1007/s10270-017-0600-2
12. Hug, T., Lindner, M., Bruck, P.A.: Microlearning: Emerging concepts, practices and technologies after e-learning. Proceedings of Microlearning **5**(3), 74 (2005)
13. Kirchhof, J.C., Michael, J., Rumpe, B., Varga, S., Wortmann, A.: Model-driven Digital Twin Construction: Synthesizing the Integration of Cyber-Physical Systems with Their Information Systems p. 12 (2020)
14. Lyan, G.: Urban mobility : leveraging machine learning and data masses for the building of simulators. Theses, Université Rennes 1 (2021). URL `https://tel.archives-ouvertes.fr/tel-03520672`
15. Lyan, G., Gross-Amblard, D., Jezequel, J.M., Malinowski, S.: Impact of Data Cleansing for Urban Bus Commercial Speed Prediction. SN Computer Science **3**(1), 82 (2022). DOI 10.1007/s42979-021-00966-1. URL `https://link.springer.com/10.1007/s42979-021-00966-1`
16. Lyan, G., Jézéquel, J.M., Gross-Amblard, D., Combemale, B.: Datatime: a framework to smoothly integrate past, present and future into models. In: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 134–144 (2021). DOI 10.1109/MODELS50736.2021.00022
17. Ma, X., Chen, X.: Public Transportation Big Data Mining and Analysis. In: Data-Driven Solutions to Transportation Problems, pp. 175–200. Elsevier (2019). DOI 10.1016/B978-0-12-817026-7.00007-2
18. Matsumoto, T., Sakakibara, K., Tamaki, H.: Bus line optimization using multi-agent simulation model of urban traffic behavior of inhabitants applying branch and bound techniques. pp. 234–239. IEEE (2015). DOI 10.1109/SICE.2015.7285551
19. Mendes-Moreira, J., Baratchi, M.: Reconciling Predictions in the Regression Setting: An Application to Bus Travel Time Prediction. In: M.R. Berthold, A. Feelders, G. Krempl (eds.) Advances in Intelligent Data Analysis XVIII, vol. 12080, pp. 313–325. Springer International Publishing, Cham (2020). DOI 10.1007/978-3-030-44584-3_25. Series Title: Lecture Notes in Computer Science
20. Morin, B., Barais, O., Jézéquel, J.M., Fleurey, F., Solberg, A.: Models at Runtime to Support Dynamic Adaptation. Computer pp. 46–53 (2009). URL `https://hal.inria.fr/inria-00477529`
21. Ndez, M.A.H., Stolfo, S.J.: Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem p. 29 (1998)
22. Pontius, R.G., Thontteh, O., Chen, H.: Components of information for multiple resolution comparison between maps that share a real variable. Environmental and Ecological Statistics **15**(2), 111–142 (2008). DOI 10.1007/s10651-007-0043-y
23. Riter, S., McCoy, J.: Automatic Vehicle Location - An Overview. IEEE Transactions on vehicular technology (1977)
24. Robinson, S., Narayanan, B., Toh, N., Pereira, F.: Methods for pre-processing smartcard data to improve data

quality. Transportation Research Part C: Emerging Technologies **49**, 43–58 (2014). DOI 10.1016/j.trc.2014.10.006
25. Taskar, B., Wong, M.F., Abbeel, P., Koller, D.: Link prediction in relational data. Advances in neural information processing systems **16**, 659–666 (2003)
26. Treethidtaphat, Wichai, Pattara-Atikom, W., Khaimook, S.: Bus arrival time prediction at any distance of bus route using deep neural network model. International Conference On Intelligent Transportation (2017)
27. Valencia, A., Fernandez, R.: A method to calculate commercial speed on bus corridors. Traffic Engineering & Control p. 8 (2012)
28. Yidan, S., Guiyuan, J., Siew-Kei, L., Shicheng, C., Peilan, H.: Bus Travel Speed Prediction using Attention Network of Heterogeneous Correlation Features. Society for Industrial and Applied Mathematics, Philadelphia, PA (2019). DOI 10.1137/1.9781611975673