University of California, Berkeley
Coleman Fung Institute for Engineering Leadership
Capstone Project Report - Spring 2024

Exploring Low-Code Approaches to Digital Twins with Traffic Prediction for Smart City Applications

Project ID: 24

Nozomu Kitamura[*]
Civil and Environmental Engineering

Qingyang Hu[*]
Electrical Engineering and Computer Sciences

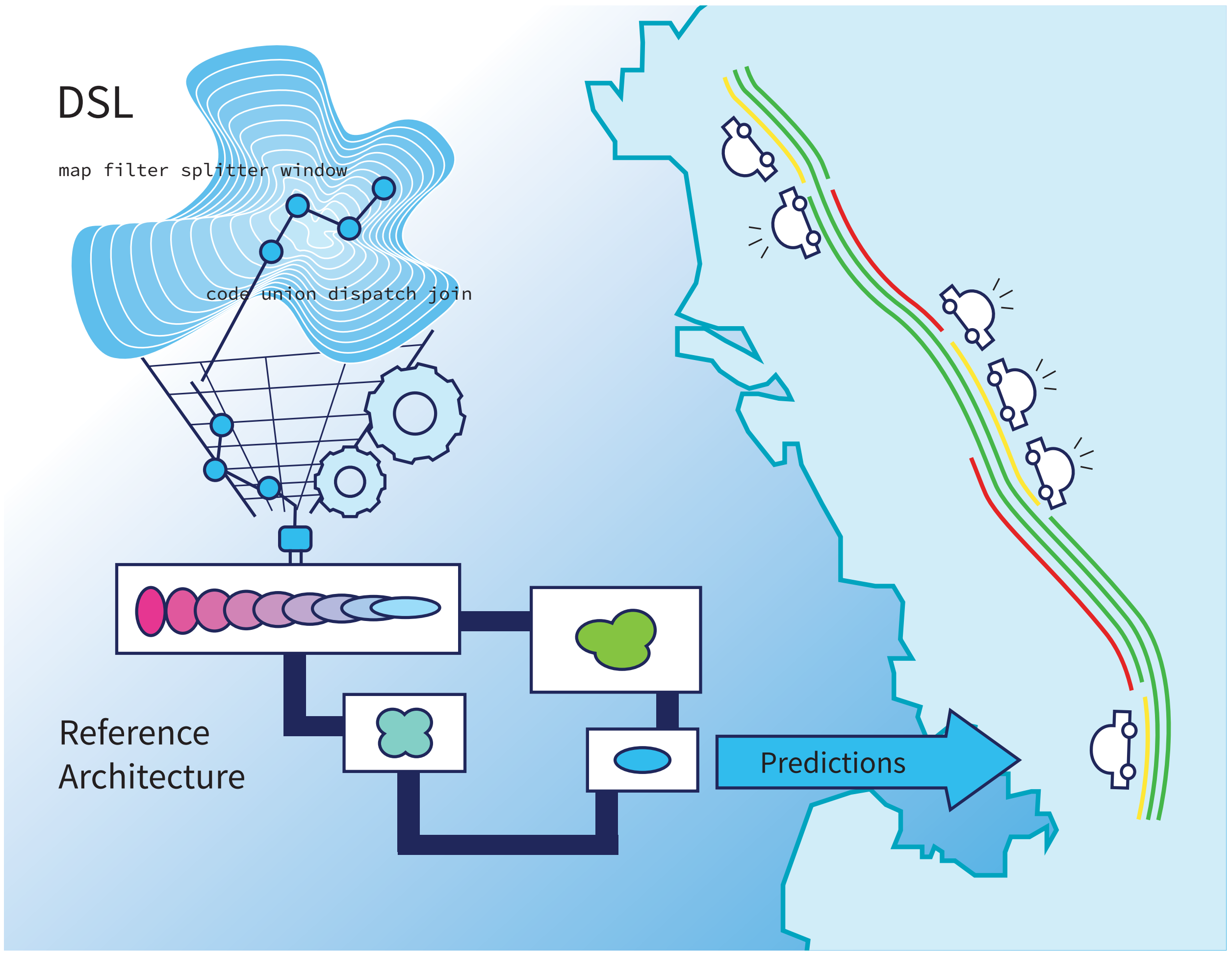
Jhan-Shuo (Jeff) Liu[*]
Electrical Engineering and Computer Sciences

Advised By

JD Margulici
Novavia Solutions

Gabriel Gomes
Mechanical Engineering

[*]All authors contributed equally to this work.



Preface: This illustration provides a high-level view of our low-code analysis for our traffic model on I-880.

Contents

Executive Summary

This project introduces a low-code framework that enables the development of digital twins without deep expertise in data engineering, software engineering, and cloud infrastructure, leveraging advancements in IoT technology. Specifically, we developed a traffic forecasting model for a specific segment of the I-880 freeway, integrating diverse real-time data sources such as speed sensors, weather conditions, and traffic event information. This model can predict traffic flow 10 minutes into the future based on real-time data, with the predictions visualized on an intuitive dashboard. Furthermore, the project explored the potential of the low-code framework by simplifying the development process application with a reference architecture and domain-specific language (DSL). This low-code framework simplifies the digital twin development process by automatically generating application code from the DSL, utilizing the reference architecture we explored. Future work will focus on implementing the low-code framework's functionality, including converting DSL into executable code and creating user-friendly management tools. In addition, efforts will be made to enhance the accuracy of our predictive models through more detailed redefinition of event severity, improved feature engineering and selection processes, and the exploration of suitable deep-learning architectures for time series forecasting. The reference architecture will also need updates to incorporate components supporting continuous improvement, locally or in the cloud. These efforts aim to make digital twin development more accessible to a broader audience and enable the creation of more accurate and practical predictive models.

Chapter 1

Business Analysis

The IoT market in the U.S. is expected to increase from approximately \$118.2 billion in 2023 to about \$553.9 billion in 2030 at a compound annual growth rate (CAGR) of 24.7%. This forecast underscores the rapid expansion of the IoT market [1]. As the IoT market evolves, IoT networks facilitate digital twins, which simulate real-world systems and processes using virtual models. This advancement enhances the monitoring and forecasting of physical operations across multiple sectors. However, creating digital twins requires a profound understanding of complex data engineering, software engineering, and cloud infrastructure. Furthermore, data scientists currently spend about 80% of their time building data pipelines for data analysis, leaving only 20% for analysis and other substantive tasks [2].

To explore the use of low-code frameworks in developing a digital twin of full-stack applications for our methodology, we have conducted extensive business studies and literature reviews. We have delved into essential elements for a real-time traffic forecasting and analysis model in section 1.1. This includes identifying crucial inputs, selecting optimal machine learning models, and finding effective ways to convert geographic locations into a linear space. In section 1.2, we investigated current approaches using cloud-based technologies to understand the feasibility of a real-time prediction system and how digital twin development is currently performed. To aid decision-making, we have examined innovative ways to present traffic flow analysis, including predictions up to 10 minutes into the future, via an intuitive dashboard in section 1.3. Additionally, we explored the potential of low-code approaches by reviewing different platforms and our own low-code framework, Anaximander, in section 1.4. These analysis collectively enhance the functionality and accessibility of our digital twin system. By integrating real-time traffic data, cloud-based processing, and intuitive dashboards through low-code platforms, we ensure a comprehensive and user-friendly digital twin solution that adapts dynamically to evolving user needs and technological advancements.

1.1 Traffic Analysis

Traffic Congestion Factors

The Cambridge report outlines current conditions, trends, and countermeasures for traffic congestion in the United States, focusing on traffic flow-affecting factors [3]. It details seven significant causes, including physical bottlenecks, traffic accidents, construction zones, weather, traffic control devices, special events, and normal traffic volume fluctuations. These factors have similarities and differences with other machine learning-based traffic flow forecasting research. Appropriately considering these key factors affecting traffic flow when forecasting traffic flow using machine learning can provide valuable insights to achieve more practical forecasts grounded in reality.

Traffic Flow Prediction Models

[4] illustrates a machine learning-based traffic forecasting package that considers weather, construction sites, accidents, and special events. They integrate actual traffic data with external factors and apply decision trees and Markov models to offer a practical tool for predicting traffic congestion. Meanwhile, [5] discusses the accuracy of machine learning methods such as Random Forest, KNN, XGBoost, and GradientBoost, with the highest accuracy reaching 92%, indicating the potential for improvement. Additionally, [6] presents a neural network-based speed prediction algorithm that uses current traffic information. These studies demonstrate the effectiveness of combining real-time data with external factors and applying machine learning and deep learning techniques to traffic flow forecasting, providing essential insights for building more realistic traffic management models.

Road Segmentation

In forecasting traffic flows, it is not realistic to uniformly forecast the entire roadway since, in reality, only a portion of the roadway is often very congested. Therefore, the guidelines in the Highway Capacity Manual provide a method for forecasting traffic flow that does not uniformly forecast the entire roadway but analyzes specific segments separately, such as merging points, turnouts, lane numbers, and sections with different speed limits [7]. This approach makes our project more realistic and detailed traffic flow forecasts.

1.2 Cloud-Based Technologies

Streaming Data

Creating a function to periodically collect data and pass it as streaming data to our pipeline is straightforward. However, [8] highlights the challenge of streaming data and proposes a solution. The issue is that data or messages might not be delivered to our system on time due to network connectivity or the iterative characteristics of data collection. He suggests defining a time window to discard any out-of-order data delayed beyond this window. This consideration is crucial when replaying data for simulation, addressing delay messages, and managing these situations in our online streaming prediction.

Event-Driven Applications

For smart city applications, updates often come in the form of events from diverse sources. Hence, event-driven development is a popular choice for these applications. [9] suggests that traffic monitoring, which involves gathering information from numerous sensors, could utilize this method. Although no realized applications were provided, a publisher/subscriber system for event communication and data mining for traffic pattern insights was recommended.

Digital Twins for Smart Cities

Digital Twins replicate physical world signals or data in the digital realm. For transportation, analyzing speed data from freeway sensors allows transportation departments to make more informed and timely decisions [10]. Thus, the digital twins concept involves collecting ample real-world data, extracting useful information, and making data-driven decisions across various fields.

1.3 Decision Support Systems

Dashboard for Data-driven Decision Making

Smart city applications assist users in making data-driven decisions. Therefore, a dashboard that visualizes and summarizes data can be beneficial. Traffic Analytics Dashboard (TA-Dash) is an interactive dashboard for visualizing urban traffic patterns over time and space [11]. The usefulness of TA-Dash is demonstrated through showcased by illustrating how it can analyze, predict, and visually represent the effects of special events on traffic. This insight underscores the need for a dashboard to visualize model predictions on geometric maps and line plots, enabling even non-expert users to understand the results.

1.4 Low-Code Solutions

Low-Code Platforms

Low-code platforms have gained popularity, enabling individuals with minimal coding expertise to develop applications swiftly. [12] provides a comprehensive comparison of existing low-code platforms, such as Google App Maker and Salesforce. They observed that these platforms share similar design models but lack built-in AI support, advanced business insights reporting, and support for event-driven applications.

Anaximander Framework

The Anaximander framework is a Python library that combines object-oriented programming with data science tools. The framework is designed to provide concise declarations of data, metadata, and transformation pipelines. Anaximander also automates the setup, configuration, and management of infrastructure, software, systems, and the resources and services required for data access in a cloud environment. This allows developers to focus on business logic and data science-related issues. In addition, the Anaximander framework supports event-driven applications through the use of low-code technologies [13]. It models data sources with Python code, enabling developers to manage them as traditional Python objects through its object-oriented design. This facilitates expressive programming and allows for quick testing of new ideas and accelerated innovation, although it has not yet been applied in real-world applications.

Chapter 2

Methodology and Results

Based on the business analysis, this project explores the potential of using low-code frameworks to simplify the development of digital twins, by constructing a real-time traffic flow prediction on the I-880 freeway. Utilizing data from various sources, including speed sensors, weather conditions, and traffic events, we've developed a model capable of forecasting traffic conditions 10 minutes into the future, with intuitive dashboards. From the model, we created a cloud-based reference architecture and Domain Specific Languages to illustrate a concept for our low-code framework.

Our methodology involves collecting and integrating data from multiple sources, as suggested in our previous analysis, to provide a comprehensive view of factors influencing traffic flow. We apply advanced machine learning techniques, such as Linear Regression and Neural Networks, to predict traffic patterns, and leverage cloud computing for efficient data processing and model deployment. The creation of the reference architecture and domain-specific language (DSL), through decomposing our implementation, aims to provide a proof of concept for our low-code framework, serving a blueprint for the future development. This would make the development process more accessible, allowing for the rapid creation of digital twin applications without extensive software engineering expertise.

The results of our project demonstrate the effectiveness of our approach in forecasting traffic flow, in both costs and time. Additionally, our exploration into low-code development with our reference architecture and DSL showcases the potential for broader application in general purpose IoT applications, making complex data engineering and software development tasks more approachable by eliminating the need to understand complex data transformation and cloud infrastructure.

This chapter outlines the details of our methodology, from traffic modeling and cloud computing to reference architecture and DSL design, and discusses our results to understand their significance for similar developments. Through our work, we contribute to the growing field of digital twin technology, highlighting the importance of integrating IoT data with low-code frameworks in solving real-world challenges.

2.1 Methodology

2.1.1 Dataset

Based on our business analysis, we identified weather, traffic events, and traffic speed as key factors for predicting traffic flow. Thus, we collected datasets from the following sources:

- Traffic events: 511 SF Bay: `511.org`
- Weather: OpenWeatherMap: `openweathermap.org`
- Traffic speed: Caltrans Performance Measurement System (PeMS): `pems.dot.ca.gov`

For weather and traffic event data, obtaining data is straightforward because these sources provide APIs for direct access. Consequently, we can retrieve data at our desired frequency, which is every 5 minutes. However, PeMS does not permit frequent data crawling without explicit permission. As a result, we opted to download PeMS daily data instead of at 5-minute intervals. To compensate for this, we introduced a data replay logic at a later stage to simulate real-time streaming by replaying past data at normal or accelerated speeds.

2.1.2 Traffic Modeling

At the start of the project, we gathered data from various sources, merged it, and assigned a section ID to each segment of I-880 for which predictions were to be made. This consolidated dataset was then subjected to exploratory data analysis (EDA) to assess its structure, identify missing values, and detect outliers. We performed data cleaning by eliminating irrelevant columns and data points through this process. The analysis used data on current traffic speeds, weather conditions, incidents, calendar dates, and times. We applied one-hot encoding to the categorical variables, such as weather conditions, traffic incidents, calendar dates, and times, and standardized the numerical data, such as current traffic speeds. In our feature engineering phase, we identified the most critical severity for events that occurred within a 60-minute window before the forecast time and within a 10-mile radius of the midpoint of the predicted section. We then introduced a new feature called the event severity score, which was calculated using the following formula:

$$\text{score} = \text{severity} \times \left(e^{-\text{time}} + e^{-\text{distance}}\right) \qquad (2.1)$$

For weather and traffic event data, where there were many types, and the individual impact was unclear, related categories were consolidated. Additionally, Principal Component Analysis (PCA) was applied to the entire dataset to reduce the dimensionality of the features. These preprocessing techniques aimed to enhance the efficiency of model training and the accuracy of forecasts. In constructing the traffic flow forecasting model, linear regression, random forest, LightGBM, and a neural network MLP (Multilayer Perceptron) were compared. The model with the lowest MSE was chosen as the final model.

2.1.3 Cloud Computing

Our pipeline utilizes the Google Cloud Platforms. We aimed to

1. Establish a production-grade data pipeline, incorporating data collection, processing, inference, and visualization.
2. Introduce data replay capabilities for a faster review of historical changes.
3. Support streaming data to observe real-time changes.

Our focus lies on utilizing FaaS (Function as a Service) for project implementation and deployment to scale down running and maintenance costs. We exchange real-time data via Pub/Sub, a publisher-subscriber system, while data storage is handled by BigQuery, a data warehouse. Our implementation consists of the following components:

1. Data Ingestors/Replayer: We will employ Cloud Function to ingest data from the datasets or to replay data from BigQuery. These processes are triggered using Cloud Scheduler based on specified inputs.

2. Preprocessor: We will preprocess the data using the feature engineering techniques detailed in the previous section, facilitated by Dataflow.

3. Inference: Depending on the scale of our model, the inference will run on either Dataflow or Cloud Function.

4. Dashboard: Dashboards will be deployed via Cloud Run which we will discuss in a subsequent section.

2.1.4 Dashboard

To enhance the visualization of predictive traffic flow and streamline development efforts, we have chosen Grafana, a robust visualization tool, to present our model predictions. Grafana offers a wide range of plot types, including time series, bar charts, heatmaps, and geomaps, making it a versatile choice. Additionally, its support for various plugins allows seamless integration with different databases. In our implementation, we incorporated the BigQuery plugin to facilitate connection with our dataset, empowering Grafana to generate insightful visualizations.

The first section of our dashboard focuses on time series data for individual segments. This includes the actual observed speeds sourced from PeMS data alongside our predictive values. To enrich the analysis, we've integrated event scores onto these graphs. These visualizations enable us not only to compare predicted and actual speeds but also to examine any correlation between event scores and actual speed fluctuations.

The second segment of the dashboard is dedicated to showcasing traffic congestion levels on a geographical map. This provides users with a comprehensive view of congestion across different segments at specific timestamps. Each segment is represented by an arrow on the map, with colors indicating the severity of congestion. As congestion escalates, the color of the arrow transitions from green to orange, red, and eventually purple, providing a clear indication of worsening traffic conditions.

2.1.5 Low-code approach

In order to illustrate the concept of our low-code framework, we simplified our implementation. This transformed our product into a reference architecture and a domain-specific language (DSL). Our low-code framework design is supported by the reference architecture, which functions as the fundamental backbone. In this proposed design, the low-code framework will be responsible for generating the application code using our reference architecture.

Our DSL has been formulated to combat the issues associated with traditional approaches such as SQL, which grapple with stateful transformations, late or out-of-order data, and trade-offs between cost, correctness, and latency. The DSL will be specified in pseudocode, using Python as the designated language.

Our core objectives in this methodology were to incorporate the separation of the data model and control flow, ensure the proficient use of decorators, and define stateful and stateless transformations in a more straightforward and efficient manner.

2.2 Results

2.2.1 Traffic Modeling: Accuracy

The initial analysis used the current speed as the forecasted speed for 10 minutes later, serving as a baseline. This method computed the Root Mean Squared Error (RMSE), with a lower RMSE indicating better performance. Subsequently, the results of the comparative analysis using linear regression, random forest, LightGBM, and MLP neural networks, which are utilized in other studies mentioned in the literature review, revealed that the MLP model exhibited superior performance, as indicated in the table 2.1. Model training and accuracy evaluation were conducted using five-fold cross-validation and RMSE, and hyperparameter optimization was carried out using random search. The MLP model achieved an 11.8% reduction in RMSE compared to the baseline. Furthermore, the MLP model reduced the RMSE by 7.8 % compared to a fundamental linear regression benchmark, offering more realistic forecasts than the other evaluated models[1].

Model Type	RMSE
Baseline	2.80
Linear Regression	2.68
Random Forest	2.62
LightGBM	2.51
MLP	2.47

Table 2.1: Comparison of RMSE using 5-Fold Validation Across Various Models

2.2.2 Cloud Computing

Through the use of serverless compute technologies, our implementation on the Google Cloud Platform supports streaming data processing, facilitated by the use of PubSub and Dataflow. This functionality means that our pipeline can be evaluated against real-time data, enabling users to conduct analyses based on real-time inputs rather than relying solely on downloaded data. Specifically, our pipeline can generate real-time predictions of traffic speeds for the next 10 minutes based on current data.

Additionally, due to the adoption of FaaS, our implementation requires minimal maintenance once deployed and hence, incurs minimal costs. As Google manages the fundamental environment for our implementation, we eliminate the need to handle the complex setup of operating systems and environment. Furthermore, Google will only bill us for the time resources are in use, thereby reducing wastage of funds on idle resources. On average, our implementation incurs approximately \$0.50 in costs per day [2], comparing to hundreds of dollars per day with traditional server-based approaches.

[1]Code to train our models is available at `https://github.com/BayAreaCloudCity/trainning`.

[2]Code of our GCP implementation, along with instructions on how to deploy it, is available at `https://github.com/BayAreaCloudCity/gcp`.

2.2.3 Dashboard

Our dashboard is shown below in Figure 2.1, which provides a time-series view on specific data for each segment, and a map-view on overall congestion level for all segments given a specific time.

In examining the time series panel, two noteworthy observations emerge. Firstly, our predictions consistently lag behind the actual speed, for about 10 minutes. This discrepancy arises because our model forecasts the speed 10 minutes ahead based mostly on the current speed, essentially employing a baseline approach. This allows us to assess the accuracy of our model in comparison to this baseline. Secondly, the event score appears disconnected from instances of speed drops. Potential explanations for this misalignment include inaccuracies in the formula used to calculate the event score or discrepancies in the event data itself. Upon investigation, we discovered instances where event data from 511.org may be sourced from alternative channels, resulting in multiple events sharing identical creation times. This suggests a potential need for refining the event data collection process to ensure accuracy. Moreover, even when creation times are intended to be accurate, delays between the occurrence of an event and its reporting may still impact the data reliability.

Turning to the map panel, a notable disparity in congestion levels between peak traffic hours and midnight is evident. Users have the flexibility to adjust the time settings to observe distinct patterns [3].

2.2.4 Reference Architecture

By simplifying our implementation, we have created a reference architecture, as shown below, for general-purpose digital-twin applications. Based on our business analysis, our reference architecture reflects our implementation by dividing the product into several components: data ingestors, data replayers, feature pipeline, training pipeline, inference pipeline, and business insights. We distinguished between functionality and technology by representing the required

[3]Our dashboard can be accessed at `https://grafana-et73rt2k6a-uw.a.run.app/`.

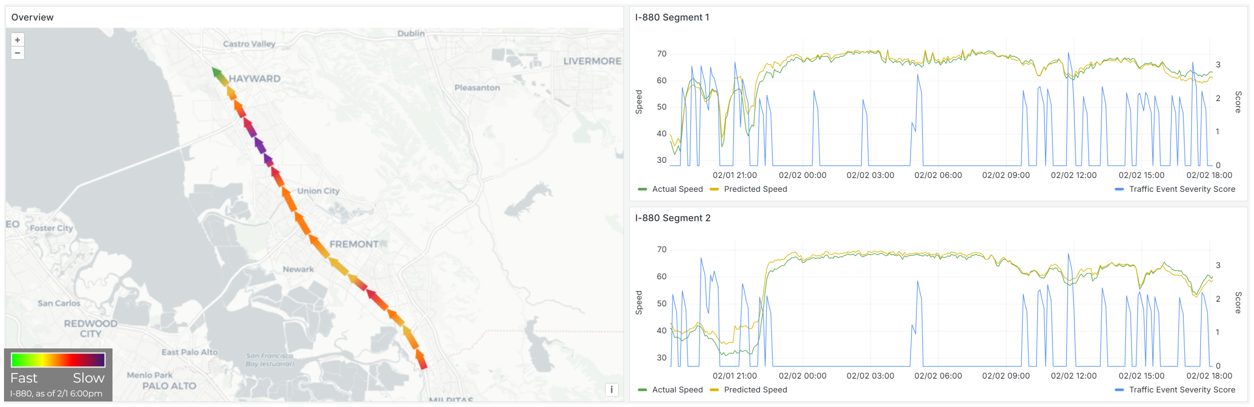


Figure 2.1: Our dashboards for I-880. The left side shows overall speed at the current time, and the right side shows historical data with their predictions over the past 24 hours.

technology within the boxes. Our low-code framework will mirror this design when generating the application code.

2.2.5 Domain Specific Languages

The DSL design comprises two main components. Firstly, we conceptualize data as assets organized into typed streams, allowing for easier transformation through framework built-in functions. Secondly, our DSL incorporates default stream operations to enhance operational efficiency.

For typed streams, we categorize streams into sample, event, session, transition, and journal, as detailed in Table 2.2.

We have also define many types of stream operations, including operations within a stream and among multiple streams. Our design assumes that all data types can be accommodated within typed streams, and data transformation can be executed seamlessly using our operation design. Operations within a single stream include

- `map`: This function applies a transformation to each record individually, ensuring a one-to-one correspondence between input and output records.
- `filter`: This function selects records based on specific criteria, effectively filtering out records that do not meet these criteria.
- `splitter`: This function divides each record into multiple parts, facilitating the processing of each part separately.
- `window`: This function selects records that fall within a specified time frame, often used for analyzing trends over time.
- `groupby`: This function groups records based on a common key and applies aggregate functions (like sum, average, etc.) to each group, summarizing or combining data in meaningful ways.

Operations among multiple streams include

- `union`: This function combines multiple data streams into a single stream without merging their content based on keys or conditions, simply appending one stream to another.
- `dispatch`: This function routes records from a single input stream to multiple output streams, distributing the data based on specified criteria or conditions.
- `join`: This function merges two or more streams into a single stream, aligning records based on shared keys and/or timestamps, which allows for correlating data across different sources.

Figure 2.3 is an illustration of our PeMS data transformation process. Using our DSL design, we ini-

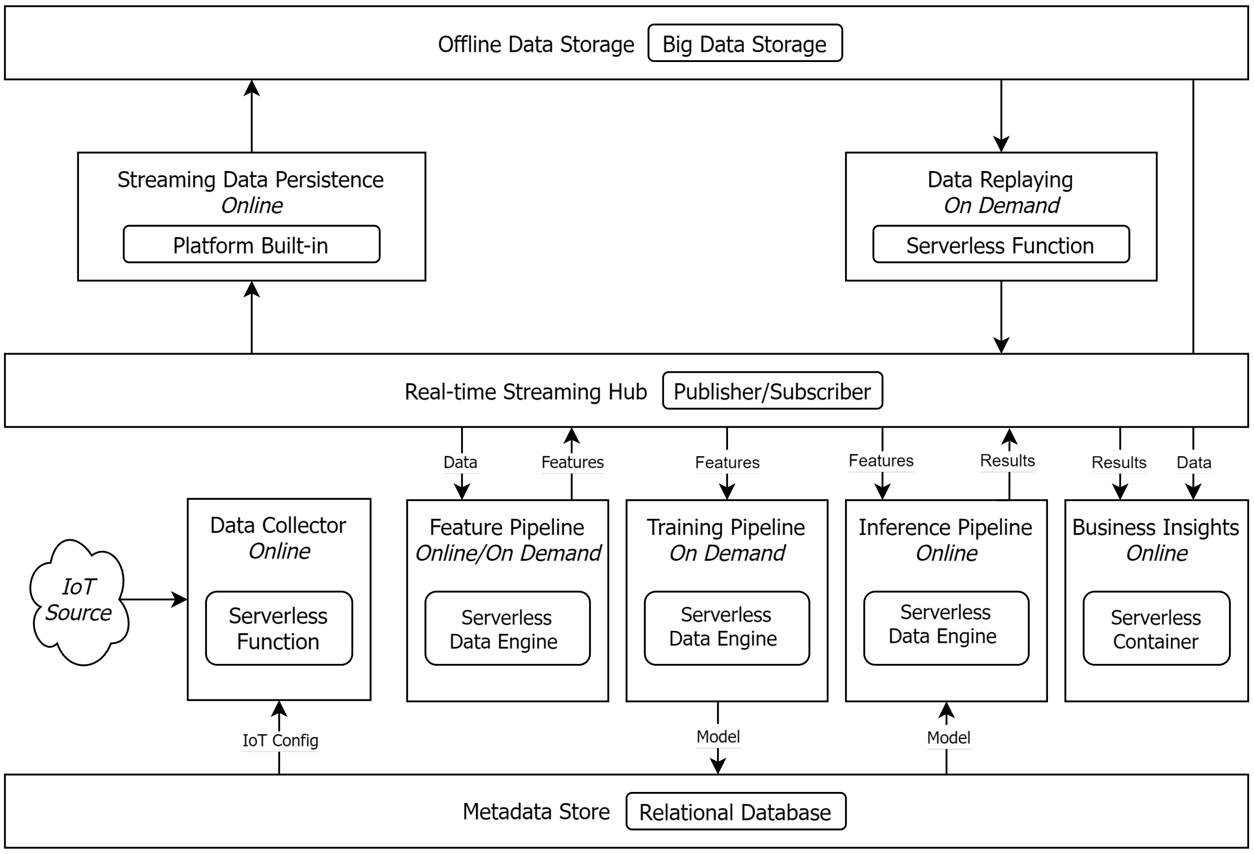


Figure 2.2: Our reference architecture to be used for future IoT projects.

tially define the data models for both input and output data. Subsequently, we specify how the input data model is transformed into the output model, utilizing a combination of predefined functions and custom aggregation functions. This process shares similarities with Dataflow in terms of overall structure. The transformation involves mapping station observation data to segments, followed by grouping by `segment_id` to calculate aggregated speeds using the `get_pems_features` function. Additionally, we incorporate windowing functions into this transformation.

Unlike the original Dataflow (Apache Beam) implementation shown in Figure 2.4, which unions multiple streams first and then applies transformations, our DSL takes a different approach. We transform multiple streams separately first and then combine them. However, the key advantage of our DSL lies in its utilization of data models, eliminating the need for users to manually extract data from storage. Our framework handles this task seamlessly, simplifying the process of building data pipelines for users. By merely defining the data models and model transformations, users can effortlessly construct data pipelines, underscoring the usability and practicality of our system.

Overall, our implementation serves as evidence of the practicality of a low-code framework. If users can easily write code of this nature, the framework could seamlessly translate it into a reference architecture and generate corresponding components within a cloud environment. We envision this level of abstrac-

Feature Type	Characteristics	Examples
Sample	Near-periodic Timestamps	Teamperature Measurements
Event	Non-periodic Timestamps Typed or untyped events	Alerts from monitoring systems
Session	Start and end times Non-overlapping intervals	Machine production runs
Transition	Time-partitioning	Parking space availability
Journal	Strictly periodic timestamps sliding or tumbling window	Daily sales reports

Table 2.2: Feature types with their characteristics and examples.

```python
class SegmentPeMsJournal(nx.Journal):
    segment_id: int = nx.key()
    timestamp: datetime = nx.timestamp()
    aggregated_speed: Speed = nx.data(ge=0)

    @nx.source
    def from_sample(cls):
        PeMSSample.map(aggregated_speed, field="station_id",  new_field="segment_ids") \
        .splitter(field="segment_ids", new_field="segment_id") \
        .group_by_key(key="segment_id") \
        .agg(get_pems_feature) \
        .sliding_windows(window_size, window_period)
```

Figure 2.3: DSL Pseudocode in Python to process aggregated data for each segment

tion as accessible for individuals with a basic coding proficiency but lacking extensive software and data engineering backgrounds, enabling them to develop data-intensive applications [4].

2.3 Future Work

2.3.1 Low-code Framework

While our project has proposed a viable method for implementing a low-code framework through our Domain-Specific Language (DSL) and reference architecture, the actual development within Anaximander–the low-code framework we are developing–remains a work in progress. To fully realize the potential of our work for Anaximander, we will need to establish a transformation process from our DSL to executable Python code. Additionally, it will be necessary to interconnect different components using Python code and develop tools that enable users to easily manage them. Ultimately, the outcome will be a comprehensive library in Python. Users can incorporate this library into their projects and write DSL code in Python as usual. Our framework will then automatically generate executable Python code and deploy it to a cloud environment following our reference architecture.

[4]Our complete design of DSL can be accessed at `https://github.com/BayAreaCloudCity/low-code`.

```python
pems: PCollection = (
    pipeline
    | "PeMS: Read" >> io.ReadFromBigQuery(
        query=get_table_query(pems_table, start, end, window_size),
         ↪  use_standard_sql=True)
    | "PeMS: Map to Segments" >> ParDo(PeMSTransformDoFn(segments))
    | 'PeMS: Window' >> WindowInto(SlidingWindow(window_size, window_period)
)
...
result: PCollection = (({
        'bay_area_511_event': bay_area_511_event, 'weather': weather, 'pems': pems})
    | 'Merge by Segment' >> CoGroupByKey()
    | 'Feature Transform' >> ParDo(SegmentFeatureTransformDoFn(segments,
    ↪  metadata_version))
    | 'Discard Buffer' >> Filter(lambda row: start.timestamp() <= row['timestamp'] <
    ↪  end.timestamp()))

class SegmentFeatureTransformDoFn(DoFn):
    def process(self, element, window=DoFn.WindowParam):
        segment_id, data = element
        t = window.end
        features = \
            self.get_event_features(data['bay_area_511_event'], segment_id, t) + \
            self.get_pems_feature(data['pems'], segment_id) + \
            self.get_weather_features(data['weather']) + \
            self.get_time_features(t)
        # Use get_pems_feature to aggregate station observed speeds into the segment
        ↪  speed
```

Figure 2.4: Dataflow code in Python to process aggregated data for each segment

2.3.2 Model Improvements and Data Correctness

The accuracy is crucial to the usefulness of this application. In order to improve the accuracy, future efforts will focus on getting more high-quality data or enhancing the model. Model enhancements will require a more detailed redefinition of event severity, improved feature engineering and selection, and the exploration of deep learning architectures designed explicitly for time-series forecasting. The most significant events are identified within a 60-minute window preceding the forecast time and within a 10-mile radius of the forecast interval's midpoint. However, a finer temporal and spatial refinement of the event severity definition could provide a more nuanced understanding of the relationship between events and velocity. Furthermore, the accuracy can be improved by adding and examining the correlation between different speeds (fast, medium, and slow) and forecasts. Moreover, the model can reduce complexity and prevent overfitting by eliminating less relevant ones. Additionally, it will investigate deep learning architectures and techniques, including RNNs suitable for time-series analysis, enhanced versions like LSTM and GRU, and the statistical time series model SARIMA. These improvements will be implemented incrementally in stages to enhance the accuracy and reliability of our forecasting models.

2.3.3 Continuous Training Using Streaming Data

Continuous improvement of models is another crucial aspect to consider for many business purposes, due to the constantly changing trends in users' behaviors and needs. To incorporate this concept, our reference architecture will need updates to include components that allow for such continuous improvements, whether locally or in the cloud. This could involve, for example, the addition of a model evaluation component and a model store. Additional logic may also be required, such as the continuous evaluation of model performance, complemented by dashboards that visualize relevant error metrics. Moreover, our Domain-Specific Language (DSL) would need updates to simplify processes for users. For instance, within our DSL, users could specify the frequency and dataset size necessary for model training, eliminating the need for writing complex scheduler logics.

2.4 Conclusion

Our project has comprehensively explored the challenge of digital-twin development involving IoT devices, specifically predicting traffic flow. We have gathered, analyzed, and modeled data from diverse sources, applying state-of-the-art machine learning techniques and leveraging cloud computing technologies to achieve our objectives. Through our efforts, we have not only developed a robust model for traffic flow prediction but also created a scalable, cost-effective infrastructure on the Google Cloud Platform, which stands as a reference implementation to the potential of low-code frameworks in handling big data and real-time analytics.

Our innovative approach in integrating a low-code framework and a Domain-Specific Language (DSL) creates an innovative strategy to democratize technology, making it accessible to people without software engineering expertise, such as domain experts and data scientists. Our project aligns with our vision to allow users to perform advanced analytics and machine learning without the need for deep technical knowledge.

References

[1] F. B. Insights, "U.s. internet of things (iot) market size, growth, forecast, 2030," *Market Research Report*, p. 110, 2023.

[2] G. Trotino, "Building data pipelines: Capabilities, benefits, and challenges," https://www.k2view.com/blog/what-is-a-data-pipeline/#Clean-Data-Wanted, 2021.

[3] C. Systematics, "Traffic congestion and reliability: Trends and advanced strategies for congestion mitigation," United States. Federal Highway Administration, Tech. Rep., 2005.

[4] J. K. Garrett, J. Ma, H. Mahmassani, M. Neuner, R. Sanchez *et al.*, "Integrated modeling for road condition prediction phase 3 project report," United States. Federal Highway Administration. Office of Operations, Tech. Rep., 2020.

[5] N. Zafar and I. Ul Haq, "Traffic congestion prediction based on estimated time of arrival," *PloS one*, vol. 15, no. 12, p. e0238200, 2020.

[6] J. Park, D. Li, Y. L. Murphey, J. Kristinsson, R. McGee, M. Kuang, and T. Phillips, "Real time vehicle speed prediction using a neural network traffic model," in *The 2011 International Joint Conference on Neural Networks.* IEEE, 2011, pp. 2991–2996.

[7] B. Aghdashi. Freeway facilities – hcm segmentation process. University of Florida, Transportation Institute. [Online]. Available: https://www.youtube.com/watch?v=3A9SPRCnUHs

[8] J. S. Damji, B. Wenig, T. Das, and D. Lee, *"Learning Spark".* O'Reilly Media, Inc., 2020.

[9] A. Hinze, K. Sachs, and A. Buchmann, "Event-based applications and enabling technologies," in *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*, 2009, pp. 1–15.

[10] C. Hu, W. Fan, E. Zeng, Z. Hang, F. Wang, L. Qi, and M. Z. A. Bhuiyan, "Digital twin-assisted real-time traffic data prediction method for 5g-enabled internet of vehicles," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 4, pp. 2811–2819, 2021.

[11] N. Tempelmeier, A. Sander, U. Feuerhake, M. Löhdefink, and E. Demidova, "Ta-dash: an interactive dashboard for spatial-temporal traffic analytics," in *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*, 2020, pp. 409–412.

[12] A. Sahay, A. Indamutsa, D. Di Ruscio, and A. Pierantonio, "Supporting the understanding and comparison of low-code development platforms," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).* IEEE, 2020, pp. 171–178.

[13] J. Margulici, "Anaximander: The rapid application development framework for data-intensive python." 2022.