HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

BIG DATA STORAGE AND PROCESSING

REAL-TIME TRAFFIC CONGESTION AND ANALYTICS

Student: *Đỗ Đình Hoàng 20225445*
Đinh Bảo Hưng 20225446
Trần Quang Huy 20225500
Trịnh Huynh Sơn 20225526
Trần Nam Tuấn Vượng 20225540

Class: *161703*

Group: *23*

Lecturer: *PhD. Tran Viet Trung*

Hanoi, December 2025

ACKNOWLEDGEMENT

First and foremost, we would like to express our deepest gratitude to PhD. Tran Viet Trung, our lecturer for the Big Data Storage and Processing (IT4043E) course. His insightful lectures, dedicated guidance, and valuable feedback throughout the semester have been instrumental in the successful completion of this project.

We also extend our thanks to the School of Information and Communication Technology (SoICT) at Hanoi University of Science and Technology for providing us with the academic environment and resources necessary to explore modern data engineering technologies.

Finally, we are grateful to our teammates for their tireless efforts, collaboration, and shared commitment to building this real-time traffic analytics pipeline. This project has not only enhanced our technical proficiency in Apache Kafka, Spark, and Machine Learning but also strengthened our ability to work effectively as a cohesive unit.

ABSTRACT

This project focuses on the design and implementation of a real-time data pipeline dedicated to monitoring and analyzing urban traffic congestion. The research explores the architecture required to ingest high-frequency data streams from road-side sensors and manage them through a multi-tiered storage strategy to ensure data durability and accessibility for analytics. Central to the system is a predictive modeling framework that analyzes historical flow patterns and temporal features to anticipate future traffic conditions. Furthermore, the project includes the development of a web-based service for real-time inference and a geospatial dashboard that enables stakeholders to visualize traffic density and predictive insights across a digital map. The final solution aims to provide a scalable infrastructure for intelligent transportation systems, facilitating data-driven decision-making for proactive urban traffic management.

TABLE OF CONTENTS

CHAPTER I. INTRODUCTION

1.1 Context and Motivation

In the era of rapid urbanization, traffic congestion has become a critical challenge for smart city infrastructure, leading to economic losses and environmental degradation. The advent of Internet of Things (IoT) sensors in transportation networks generates a massive volume of high-velocity data, which requires sophisticated Big Data storage and processing techniques to extract actionable insights. This project is motivated by the need for an automated, low-latency system capable of not only monitoring current traffic states but also forecasting near-future conditions. By leveraging a microservice architecture and stream processing tools such as Apache Kafka and Spark, the system aims to transform raw sensor streams into a structured analytical asset.

1.2 Problem Statement

The primary challenge in modern urban infrastructure is the inefficiency of traffic management systems in mitigating congestion, which leads to significant economic productivity loss, environmental degradation, and increased fuel consumption. Current monitoring frameworks are largely reactive, relying on static or historical datasets that provide descriptive analysis rather than the proactive, predictive insights required to anticipate and manage traffic flow in real-time.

From a Big Data engineering perspective, the problem is defined by the technical difficulty of building a scalable, low-latency pipeline capable of handling high-velocity data streams. While technologies like message brokers and distributed processing engines have made large-scale analysis feasible, several critical bottlenecks remain:

- **Ingestion and Velocity:** Traditional systems cannot efficiently ingest and process continuous streams of telemetry data from numerous road sensors at high frequencies.
- **Storage Optimization:** In a Medallion Architecture, implementing a "Bronze" layer using Parquet format faces the "small file problem," where high-frequency writes of small data batches can degrade query performance and storage efficiency.

- **Predictive Integration:** There is a significant gap in synchronizing real-time feature engineering (e.g., calculating rolling averages and lag features) with distributed stream processing to feed machine learning models for dynamic speed and congestion level prediction.
- **Actionable Visualization:** Raw data lacks value without a system that can dynamically identify congestion levels and present them through a geospatial interface, allowing stakeholders to visualize traffic hotspots as they emerge.

This project addresses these challenges by designing an end-to-end pipeline that simulates sensor data, manages it through structured storage layers, and applies predictive modeling to provide real-time traffic intelligence.

1.3 Project Objectives

The scope of this project focuses on designing and implementing a real-time traffic congestion analytics pipeline using modern big data technologies. The system is built to simulate traffic sensor data, process it in both streaming and batch modes, apply machine learning for prediction, and visualize results through a user-friendly dashboard.

Specifically, the project includes the following components and functionalities:

- **Traffic data simulation:** Traffic sensor data is simulated in real time, representing multiple road segments (e.g., roadA, roadB, roadC). Each data record contains the sensor identifier, current vehicle speed, and timestamp. This approach allows the system to be tested without relying on external real-world data sources.
- **Real-time data ingestion:** Apache Kafka is used as a message broker to stream traffic data continuously. Producers send simulated traffic events to Kafka topics, while downstream consumers read and process the data in real time.
- **Stream processing and storage:** Apache Spark Structured Streaming is used to consume traffic data from Kafka and store raw data into Parquet files (Bronze layer). This enables scalable processing and efficient storage for later batch analytics and model training.

- **Machine learning-based prediction:** Machine learning models are trained using historical traffic data stored in Parquet format. A Random Forest regression model is trained to predict the next time-step speed based on recent speed trends and sensor information, while an XGBoost classification model is trained to predict the current congestion level based on speed, sensor, and timestamp data. The trained models are saved and later used by a REST API for real-time inference.

- **REST API for prediction:** A FastAPI-based service exposes an endpoint that accepts real-time traffic input and returns predicted future speed along with congestion-related information. This API serves as a bridge between the machine learning model and the visualization layer.

- **Interactive dashboard:** A Streamlit dashboard allows users to input traffic conditions, view predictions, analyze congestion levels, and visualize sensor locations on a map. The dashboard provides both tabular history and geospatial visualization for intuitive understanding.

1.4 Technical challenges

Implementing a real-time traffic congestion analytics pipeline involves overcoming several technical hurdles inherent in Big Data storage and processing:

- **Management of High-Velocity Data Ingestion:** Unlike traditional systems that rely on static or historical data, this system must ingest high-velocity streams from multiple simulated road sensors at 2-second intervals. The primary challenge lies in configuring message brokers, such as Apache Kafka, to handle continuous ingestion without data loss or significant system latency.

- **Storage Optimization and the "Small File Problem":** In a Medallion Architecture, raw data is moved from JSON streams to a structured Parquet format in the Bronze layer. However, frequent real-time writes of small data batches can lead to the "small file problem," which significantly degrades query performance and metadata management in Spark environments. Implementing effective batching or "flushing" mechanisms (e.g., every 20 records or 60 seconds) is essential to balance real-time requirements with storage efficiency.

- **Real-Time Feature Engineering Consistency:** A major technical gap exists

in ensuring that the complex features used for model training - such as rolling averages and lagged speed variables - are replicated exactly in the real-time inference environment. The FastAPI service must perform these calculations and apply data transformations (like one-hot encoding for sensor IDs) on the fly for single incoming records to ensure the Random Forest model produces accurate results.

- **Transitioning from Descriptive to Predictive Analytics:** Traditional traffic monitoring is often limited to descriptive analysis, showing only what is currently happening. The challenge here is integrating machine learning models to provide predictive insights that allow traffic managers to anticipate upcoming congestion before it occurs. This requires implementing a supervised machine learning pipeline using machine learning models that dynamically predicts next time-step speed and congestion categories (HIGH, MEDIUM, LOW) by analyzing temporal patterns, including real-time sensor telemetry.
- **Dynamic Geospatial Visualization:** Visualizing traffic density and prediction history in real-time requires a seamless connection between the frontend and the data storage layers. The challenge is to implement an interactive dashboard (using Streamlit and Pydeck) that can dynamically update geospatial maps without refreshing the entire interface, ensuring that stakeholders have a low-latency view of emerging traffic hotspots.

CHAPTER II. ARCHITECTURE AND DESIGN

2.1 Tech Stack

2.1.1 Container Orchestration

While container orchestration frameworks such as Kubernetes or Docker Swarm are industry standards for distributed systems, this project intentionally utilizes a local execution environment for the deployment of all system components. This architectural decision was made to prioritize the development of core data processing logic and the refinement of the machine learning workflow. By bypassing the initial overhead associated with container networking and infrastructure management, the development cycle remained focused on functional correctness and the integration of the Big Data stack without the complexity of an abstraction layer.

Despite the absence of a formal orchestration engine, the system maintains a strictly decoupled, modular architecture. Each service - comprising Apache Kafka for message brokering, Apache Spark for unified batch and stream processing, FastAPI for low-latency inference, and Streamlit for real-time visualization - operates as an independent process. These components communicate via standardized interfaces, specifically through Kafka topics, RESTful API endpoints, and schema-optimized file exchanges using the Parquet format. This ensures that the logical boundaries of the system are preserved, mirroring the behavior of a distributed microservices environment.

Furthermore, this modularity serves as a strategic foundation for future scalability. Because the application logic is decoupled from the underlying host environment, the system is "container-ready." Each individual component can be encapsulated into a Docker image and transitioned to a managed Kubernetes cluster with negligible modifications to the source code. Consequently, the current implementation provides a robust, functional prototype that balances immediate development efficiency with a clear pathway toward enterprise-grade distributed deployment.

2.1.2 Data Orchestration

In alignment with the project's scope as an experimental prototype, the data orchestration logic is implemented via modular Python scripts and a coordinated execution flow rather than through heavy-weight orchestration engines like Apache

Airflow or Prefect. This approach minimizes infrastructure overhead while maintaining a rigorous separation of pipeline stages. The orchestration strategy relies on a structured directory hierarchy and a multi-layered data architecture—specifically adopting the Medallion Architecture (Bronze and Silver layers)—to ensure data lineage and integrity throughout the lifecycle.

The end-to-end data lifecycle is orchestrated through four distinct phases:

- **Ingestion and Bronze Layer Storage:** The pipeline initiates with a dedicated ingestion service that interfaces with external traffic APIs. This component manages the retrieval of raw traffic speed data, performs initial response parsing, and persists the immutable raw data into the Bronze layer in Parquet format to ensure efficient storage and future replayability.
- **Batch Transformation and Silver Layer Refinement:** Following successful ingestion, batch processing tasks are triggered to refine the raw data. These scripts perform critical ETL operations, including data cleansing, handling of missing values, and feature derivation. The resulting high-quality datasets are stored in the Silver layer, serving as the "source of truth" for downstream analytical and modeling tasks.
- **Model Training and Serialization:** A specialized training module accesses the Silver-tier data to perform high-level feature engineering and model optimization. Upon completion, the trained machine learning models are serialized and versioned, ensuring they are available for the subsequent inference stage.
- **Real-time Stream Orchestration:** For the simulation of live traffic environments, the orchestration transitions to an event-driven model. Traffic data is published to Apache Kafka topics, where it is consumed by Spark Structured Streaming. This stage integrates the pre-trained models to generate real-time predictions, which are then routed to downstream visualization and consumer services.

This orchestrated flow prioritizes transparency and modular debugging over automated scheduling complexity. By maintaining clear boundaries between ingestion, refinement, and inference, the system remains highly maintainable. Furthermore, this design mirrors the logical structure of professional DAGs (Directed Acyclic Graphs), ensuring that the entire pipeline can be transitioned to a fully automated orchestrator with minimal architectural refactoring.

2.1.3 Data Batch Processing

Batch processing serves as the foundational analytical engine of this project, enabling high-throughput transformations and complex feature engineering on historical traffic datasets. The workflow is architected using Apache Spark, leveraging its distributed computing capabilities to handle data volumes that exceed the memory limits of traditional single-node processors. By utilizing the Spark SQL and DataFrame API, the system performs declarative transformations that are optimized by the Catalyst optimizer, ensuring efficient execution plans for large-scale data refinement.

The batch pipeline is structured into a logical sequence designed to transition data from a raw state to an analysis-ready format:

- **High-Performance Ingestion from Bronze Layer:** The pipeline ingests raw data persisted in the Parquet format. This columnar storage format is critical for Big Data workloads, as it enables predicate pushdown and column projection, significantly reducing I/O overhead by reading only the necessary data blocks. Spark aggregates these distributed files into a unified DataFrame, providing a consistent view of the historical traffic state.
- **Data Sanitization and Validation:** To ensure the integrity of downstream models, the system performs rigorous data cleaning. This includes the elimination of null-value records, normalization of ISO-8601 timestamps, and spatial-temporal sorting. These operations establish a deterministic baseline, preventing "garbage-in, garbage-out" scenarios in the machine learning lifecycle.
- **Distributed Feature Engineering:** The project utilizes Spark's transformation libraries to derive complex temporal features. Key operations include:
  - *Window Functions:* Calculating rolling mean speeds and other aggregated features to capture short-term traffic oscillations.
  - *Lag Analysis:* Generating historical offsets to represent temporal dependencies.
  - *Categorical Encoding:* Vectorizing sensor metadata for compatibility with ML algorithms.

  - *Cyclical Temporal Embedding:* Transforming periodic hour and minute variables into trigonometric coordinates (sine and cosine pairs) to preserve circular continuity in daily and hourly patterns.
  - *Binary Indicators:* Deriving boolean flags to isolate distinct temporal regimes—specifically weekend cycles and nighttime intervals — to enable the model to account for systemic variance in road utilization and periodic shifts in traffic velocity.

- **Silver Layer Persistence and Schema Enforcement:** The final enriched dataset is exported to the Silver layer. Unlike the raw Bronze data, the Silver layer maintains a structured schema with pre-computed features, optimized for rapid retrieval by model training scripts and analytical queries.

By decoupling the batch processing layer from the ingestion and inference phases, the architecture achieves high fault tolerance and reproducibility. Transformations can be re-run over historical partitions to incorporate new feature logic without disrupting the live streaming pipeline. This design ensures the system scales linearly as the historical data footprint expands, providing a robust backbone for continuous model improvement.

2.1.4 Ecosystem

The architecture of this project is conceived as a decoupled Data Engineering Ecosystem rather than a monolithic application. By adopting a modular design, the system ensures that each stage of the traffic congestion analytics pipeline—from ingestion to visualization—functions as an independent service. This ecosystem approach mirrors enterprise-level data platforms, providing a robust framework for handling the "Three Vs" of Big Data (Volume, Velocity, and Variety) while ensuring that the system remains maintainable and extensible.

The ecosystem's functionality is distributed across several specialized layers that interact through well-defined protocols:

- **Ingestion and Persistence Layer:** Acting as the entry point, this layer abstracts the complexities of external API interactions or sensor simulations. By persisting data directly into the Bronze (Raw) Layer, the system creates an immutable audit trail, allowing for data recovery and re-processing without requiring re-ingestion from the source.

- **Unified Processing Layer:** The core of the ecosystem leverages both Batch and Stream processing paradigms. While the batch engine focuses on high-latency, high-volume historical transformations, the streaming engine addresses the velocity requirements of real-time traffic monitoring. This dual-processing capability ensures the system can provide both long-term trend analysis and immediate operational insights.
- **Machine Learning Lifecycle Integration:** The ecosystem treats ML models as first-class citizens. The pipeline facilitates a seamless transition from the Silver Layer (refined data) to model training, serialization, and finally, live inference. This lifecycle ensures that predictive insights are grounded in high-quality, pre-processed data, maintaining a tight feedback loop between historical patterns and real-time predictions.
- **Decoupled Visualization and Consumption:** The Streamlit-based dashboard functions as a consumer-only layer, decoupled from the underlying processing logic. It subscribes to processed outputs and prediction streams, ensuring that the user interface remains responsive even during heavy computational loads in the background.

Architectural Advantages: The ecosystem approach provides several strategic benefits essential for Big Data environments. Fault Isolation ensures that a bottleneck in the training script does not impede the real-time ingestion of traffic data. Furthermore, the system exhibits high extensibility; the modular interfaces allow for the integration of additional data sources (e.g., weather or public transit data) or the upgrade of the ML model architecture with zero downtime for the rest of the pipeline. Ultimately, this architecture achieves a sophisticated balance between a functional prototype and a scalable, production-ready data platform.

2.1.5 Data Warehouse

In this project, the data warehousing function is implemented via a File-Based Analytical Storage model, adopting the principles of a Data Lakehouse. Rather than utilizing a traditional relational database management system (RDBMS), the system leverages high-performance columnar storage formats persisted directly to the file system. This architecture provides the schema flexibility of a Data Lake while maintaining the analytical performance typically associated with a structured

Data Warehouse, making it ideal for the high-volume temporal data found in traffic analytics.

1. *Layered Storage Design:* The warehouse is logically partitioned into discrete layers to ensure strict Data Traceability and Idempotency (the ability to re-process data with consistent results):

   - **Bronze Layer (Raw Persistence):** This layer acts as the ingestion landing zone. It stores immutable records collected from Kafka or simulated producers in their original state. By preserving the raw telemetry, the system allows for the re-evaluation of data should processing logic or business requirements change.
   - **Silver Layer (Curated Analytics):** This layer contains the "Source of Truth" for the ecosystem. It stores datasets that have undergone cleaning, normalization, and feature augmentation. This refined data is structurally optimized for rapid loading into machine learning pipelines and analytical dashboards.

2. *Optimized Columnar Format (Apache Parquet)* The choice of Apache Parquet as the primary storage format is a strategic decision driven by the requirements of Big Data workloads. Unlike row-based formats (such as CSV or JSON), Parquet's columnar layout offers several critical advantages:

   - **I/O Efficiency:** Analytical queries often target specific columns (e.g., average_speed). Parquet allows the system to skip irrelevant columns, drastically reducing disk I/O.
   - **Superior Compression:** Identical data types stored contiguously allow for advanced compression algorithms (e.g., Snappy), reducing the physical storage footprint.
   - **Schema Evolution:** The format supports the addition of new features or sensors over time without requiring expensive migrations of historical data.

3. *Integration with Machine Learning Pipelines:* The Data Warehouse serves as the primary feature store for the machine learning lifecycle. By loading partitioned Silver-layer data into Pandas DataFrames, the system can perform complex vectorised operations to derive rolling averages, temporal lags... This decoupling of storage and compute ensures that the model training process is

consistent and reproducible; the model "sees" the same refined data in the warehouse that it will eventually encounter during real-time inference.

4. *Architectural Evaluation:* This Lakehouse-style approach minimizes operational overhead by removing the need for an external database server, while simultaneously providing the high-throughput capabilities required for Spark-based batch processing. It achieves a balance between the simplicity required for an experimental prototype and the structural rigor of a production-grade Big Data environment.

2.1.6 Object Storage

The system utilizes an Object Storage paradigm for the persistent management of all intermediate and final datasets. While enterprise environments typically leverage cloud-based solutions such as Amazon S3 or Google Cloud Storage, this project implements a localized object store via the host file system. This approach serves as a high-performance, low-latency abstraction layer that simulates cloud-native data lake behaviors, ensuring that all pipeline components—from ingestion to visualization—access a centralized "Single Source of Truth."

1. *Storage Hierarchy and Data Lake Conventions:* To maintain organizational clarity and support rigorous data auditing, the storage layer is structured according to standardized Data Lake conventions. The directory hierarchy is designed to separate data by its state of refinement:
    - `data/raw/` (Ingestion Zone): Serves as the landing zone for volatile telemetry data retrieved from external APIs or simulators.
    - `data/bronze_py/` (Standardization Layer): Contains the first iteration of persisted data in Parquet format, ensuring a uniform schema while preserving raw values.
    - `data/silver_py/` (Curation Layer): Houses the highly optimized, feature-enriched datasets ready for model consumption and analytical querying.
    - `models/` (Model Registry): A dedicated namespace for versioned machine learning artifacts (e.g., model.joblib), facilitating the decoupling of model training from real-time inference.
2. *Storage Formats and Access Patterns:* The persistence layer employs specialized formats tailored to the nature of the data being stored. For structured ana-

lytical data, Apache Parquet is utilized due to its support for Atomic Appends. Since Parquet files are immutable by design, the system follows a "write-once, read-many" pattern; new data is persisted as additional partitions rather than modifying existing files, which prevents data corruption and allows for effortless historical replay.

For the Machine Learning (ML) component, artifacts are serialized using Joblib. This ensures high-efficiency persistence of large NumPy-based arrays often found in predictive models. Access to these objects is mediated through standardized Python libraries, including PyArrow and Pandas, providing a consistent interface regardless of the underlying hardware.

3. *Design Rationale for an Experimental Prototype:* Implementing a local object store provides several strategic advantages within the context of a Big Data academic project:

   - **Zero-Configuration Portability:** The entire data ecosystem is self-contained, allowing the pipeline to be reproduced across different environments without the need for cloud credentials or managed service overhead.
   - **Infrastructure Transparency:** By mirroring the directory structures of cloud providers, the architecture remains "Cloud-Ready." Transitioning from a local environment to a production S3 bucket would require only a change in the URI (Uniform Resource Identifier) prefix, with zero modifications to the core processing logic.
   - **Cost and Latency Optimization:** Local persistence eliminates egress costs and network latency, enabling rapid iteration and debugging during the development of the data processing algorithms.

2.1.7 Message Broker

The Message Broker serves as the high-throughput backbone of the system's real-time pipeline, facilitating an Event-Driven Architecture (EDA). By implementing Apache Kafka, the system achieves complete decoupling between data generation and stream processing. This asynchronous communication model ensures that the "Speed Layer" can handle continuous bursts of telemetry data without blocking upstream producers, providing the fault tolerance and scalability required for modern Big Data environments.

1. *Event Schema and Topic Management:* The system utilizes a centralized Kafka topic, identified as traffic, to aggregate sensor telemetry. To maintain interoperability between heterogeneous services (Python, Spark, and FastAPI), messages are serialized in JSON format. Each event payload adheres to a structured schema designed for time-series analysis:

   - sensor: A categorical identifier representing the physical road segment (e.g., roadA, roadB).
   - speed: A numerical value representing the instantaneous velocity.
   - timestamp: A high-precision UNIX epoch used for temporal windowing and synchronization across the pipeline.

   Producer-Consumer Dynamics: The ecosystem's real-time flow is governed by two primary actors:

   - **Distributed Producer (Traffic Simulation):** To simulate a live urban environment, a dedicated producer service generates synthetic traffic readings. By publishing messages at fixed intervals and maintaining chronological order per sensor, the producer mimics the behavior of IoT-enabled traffic cameras. This allows for rigorous testing of the system's responsiveness and its ability to handle "out-of-order" events.
   - **Streaming Consumers (Downstream Processing):** Consumers within the Speed Layer subscribe to the traffic topic to ingest live telemetry. These consumers act as orchestrators for the inference logic; they retrieve the raw event, interface with the FastAPI prediction service to determine congestion levels, and subsequently persist the enriched records into the Silver Layer.

2.1.8 Visualization

The final layer of the ecosystem is a high-performance Analytical Dashboard built using Streamlit. Serving as the primary user interface (UI), this component bridges the gap between complex backend processing and actionable human insights. The dashboard is designed to provide a real-time, interactive environment where users can monitor current traffic telemetry, trigger predictive inferences, and analyze spatial-temporal trends across the road network.

1. *Interface Functionality and API Integration:* The dashboard operates as a decoupled client that interacts with the backend through a RESTful architecture. By communicating with the FastAPI prediction service via asynchronous HTTP requests, the UI remains highly responsive, ensuring that data retrieval does not block the user experience. Key functional capabilities include:

    - **Real-time Inference Triggering:** Users can simulate specific traffic scenarios by inputting instantaneous speed parameters for selected sensors.
    - **Historical Session Logging:** The interface maintains a local state of all queries and responses, enabling comparative analysis between successive predictions.
    - **Geospatial Telemetry:** Leveraging PyDeck, the dashboard renders high-fidelity spatial data, allowing users to observe traffic patterns within their geographic context.

2. *Predictive Visualization Logic:* A distinguishing feature of the dashboard is its Proactive Congestion Mapping. Unlike traditional monitoring systems that reflect past or present states, this system utilizes predicted speed values to derive its visual indicators. This is implemented through a color-coded heuristic that provides an immediate visual summary of anticipated road conditions:

    - **Green (Optimal):** Indicates high-velocity flow with minimal predicted delay.
    - **Yellow (Transitional):** Represents moderate velocity, signaling potential congestion buildup.
    - **Red (Critical):** Denotes low predicted speeds, indicating a high probability of bottleneck formation.

    By mapping these predictions to marker colors and dynamic radii, the system transitions from a descriptive tool to a prescriptive analytics platform, allowing for proactive traffic management.

3. *Spatial-Temporal Rendering with PyDeck:* The integration of PyDeck allows for sophisticated geospatial visualization. Each sensor is represented as a dynamic point feature defined by its latitude and longitude coordinates. The visualization engine handles:

- **Contextual Metadata:** Interactive tooltips provide deep-dives into sensor-specific metadata, including historical averages and real-time inference results.
- **Dynamic Centering:** The map viewport automatically calculates the spatial centroid of the sensor network, ensuring that the most relevant geographic data is always within the user's field of view.

4. *Design Rationale: The "Python-Native" Advantage:* Streamlit was selected for this ecosystem due to its seamless integration with the Python data science stack. In a Big Data context, this eliminates the "impedance mismatch" often found when moving data from analytical models to the UI. The result is a lightweight yet powerful visualization layer that can be easily containerized and scaled, perfectly aligning with the modular, cloud-ready philosophy of the overall system architecture.

2.2 System Design

2.2.1 Overall Architecture

The system is architected around a Lambda-inspired framework, a robust design pattern in Big Data engineering that balances high-volume historical analysis with low-latency stream processing. By bifurcating the data lifecycle into a Batch Layer and a Speed Layer, the system can provide comprehensive insights that are both historically grounded and contextually relevant in real-time. This modularity ensures that the platform remains scalable and resilient, adhering to modern distributed systems principles.

1. *Architectural Components and Workflow:* The end-to-end data flow is organized into five distinct layers, ensuring a unidirectional and traceable movement of information:

   - **Ingestion Layer:** Raw telemetry is ingested from external sources and persisted into the Bronze Layer. This provides an immutable record of truth for the entire ecosystem.
   - **Batch Layer (Cold Path):** This layer handles high-latency, comprehensive processing. It extracts historical patterns, performs complex feature engineering, and facilitates the machine learning training lifecycle.

- **Speed Layer (Hot Path):** Operating in near real-time, this layer manages event streaming via Apache Kafka. It processes instantaneous traffic events, ensuring that the system can react to sudden shifts in traffic velocity.
- **Serving API Layer:** Acting as the model's interface, this layer utilizes FastAPI to expose the serialized machine learning artifacts. It serves as the bridge between the refined data layers and the end-user.
- **Visualization Layer:** The final consumer of the pipeline, providing a graphical representation of both historical trends and predictive outcomes.

2. *Core Design Principles:* The architecture is governed by several industrial data engineering principles:
   - **Separation of Concerns:** Each layer is logically isolated, allowing for independent development and optimization without impacting the global state.
   - **Idempotency and Reproducibility:** Batch processing jobs are designed to be deterministic; running the same code over the same Bronze data will always yield an identical Silver dataset.
   - **Fault Isolation:** The decoupled nature of the message broker ensures that a failure in the visualization layer does not interrupt the ingestion or processing of traffic data.
3. *Containerized Deployment Strategy:* To guarantee environment parity across development and production, each component is encapsulated within a Docker container. This strategy ensures that all dependencies—ranging from Spark configurations to specialized Python libraries—are bundled with the application logic.

   While the current implementation operates in a standalone containerized environment, the adherence to Cloud-Native design patterns ensures that the system is fully compatible with advanced orchestration platforms like Kubernetes. This "orchestration-ready" posture allows for horizontal scaling and automated self-healing in future enterprise-grade deployments.

2.2.2 Batch Layer

The Batch Layer is the core analytical engine of the system, responsible for the high-throughput processing of historical traffic telemetry and the synthesis of predictive models. Unlike the Speed Layer, which prioritizes low latency, the Batch Layer focuses on data completeness and statistical accuracy. By leveraging cold-storage data, this layer identifies long-term patterns and trains the machine learning artifacts that power the system's predictive capabilities.

1. *Data Ingestion and Schema Specification:* The Batch Layer ingests data from the Bronze persistence zone, which utilizes the Apache Parquet format. This columnar storage strategy is essential for Big Data processing, as it allows for efficient metadata handling and rapid data retrieval. The ingestion process follows a strict schema to ensure downstream model compatibility:

Column	Data Type	Description
sensor	String (Categorical)	Unique road segment identifier
speed	Float64	Observed vehicle velocity
timestamp	Int64 (Unix)	Temporal reference for event ordering

2. *ETL and Data Sanitization:* The transformation phase involves consolidating partitioned Parquet files into a unified analytical DataFrame. During this process, the system performs critical data sanitization tasks:

   - **Temporal Normalization:** Converting raw Unix epochs into structured datetime objects for time-series alignment
   - **Spatial-Temporal Sorting:** Organizing records by sensor and timestamp to ensure that windowed operations follow a logical sequence.

- **Data Integrity Filtering:** Pruning incomplete records generated during the feature-shifting process to maintain a high-quality training set.

3. *Feature Engineering and Predictive Logic* The system implements a specialized feature store designed to capture the momentum and temporal dependencies of urban traffic. These features transform raw speed values into predictive signals:

   - **Windowed Aggregations :** A rolling mean calculated over a 3-period window and a 5-Minute Temporal Windows to aggregates raw telemetry into fixed 5-minute buckets. This synchronization step normalizes irregular sensor reporting intervals into a consistent time-step, ensuring that subsequent temporal features (like lags and cyclical embeddings) are calculated on a uniform chronological grid. This smooths instantaneous noise and identifies short-term trends.
   - **Temporal Lagging (`speed_lag1`):** A first-order lag feature that provides the model with the most recent historical state, allowing it to calculate the rate of change in traffic flow.
   - **One-Hot Encoding (OHE):** To avoid the "ordinal bias" (where a model might incorrectly assume road3 > road1), categorical sensor IDs are transformed into binary vectors. This allows the model to learn localized geographic behavior.
   - **Cyclical Temporal Embedding:** Transforming periodic time variables into trigonometric coordinates. This preserves circular continuity (e.g., ensuring hour 23 is mathematically adjacent to hour 0), allowing the model to learn recurring daily and hourly traffic oscillations.
   - **Binary Temporal Indicators:** Deriving boolean flags to isolate distinct temporal regimes. These features enable the model to account for systemic variance in road utilization and the characteristic shifts in traffic velocity that occur during weekend cycles and nighttime intervals.

4. *Model Architecture:*

   - **Random Forest Regressor:** The system utilizes a Random Forest Regressor as its primary predictive algorithm. This ensemble-based approach was selected for its ability to model complex, non-linear traffic patterns and its inherent robustness against outliers in sensor data.

- **XGBoost Classifier:** The system implements an Extreme Gradient Boosting (XGBoost) Classifier to categorize traffic temporal patterns into discrete congestion levels. This ensemble-based framework was selected for its capacity to model non-linear traffic dynamics through sequential gradient descent optimization.

The training lifecycle follows a rigorous 80/20 train-test split to validate the model's ability to generalize to unseen traffic scenarios. Model performance is quantified using Mean Absolute Error (MAE) for the Random Forest Regressor and Accuracy for the XGBoost Classifier.

5. *Model Persistence and Serialization:* Upon successful validation of the predictive performance, the model is serialized into a persistent artifact using the Joblib library. Unlike standard pickle serialization, Joblib is specifically optimized for large Python objects that contain high-dimensional NumPy arrays, which are common in ensemble models like Random Forests. Besides, the XGBoost implementation utilizes the Universal JSON format for model persistence. By serializing the model to a JSON structure, the system ensures cross-platform compatibility and long-term maintainability, as JSON provides a human-readable, language-independent representation of the tree structures and boosting parameters.

   Those serialized files, `model.joblib` and `xgb_model.json`, serves as the bridge between the Batch Layer and the Speed Layer. By sharing this exact artifact, the system ensures Environmental Consistency; the identical mathematical weights and decision boundaries derived during historical training are applied to real-time streaming data. This eliminates "training-serving skew," a common failure point in machine learning production systems where the training environment differs from the inference environment.

6. *Batch Layer Deliverables:* The execution of the Batch Layer yields three critical outputs that ensure the operational integrity of the entire traffic analytics ecosystem:

   - **Serialized Predictive Artifact (`model.joblib`):** A high-fidelity model capable of performing multi-variate regression on traffic telemetry.
   - **Reproducible Pipeline Architecture:** A standardized ETL (Extract, Transform, Load) workflow that can be re-triggered as new data partitions ar-

rive in the Bronze layer, allowing for continuous model retraining and refinement.

- **Unified Feature Definitions:** A shared schema for rolling averages, lag variables... By defining these features in the Batch Layer, the Speed Layer can replicate the exact transformation logic, ensuring that real-time inputs are formatted identically to the historical training data.

In summary, the Batch Layer establishes the "intelligence" of the system. It transforms raw, disjointed historical records into a structured, predictive engine, providing the necessary foundation for the low-latency responsiveness of the Speed Layer.

2.2.3 Speed Layer

The Speed Layer constitutes the low-latency branch of the system's Lambda Architecture. While the Batch Layer provides deep historical insights, the Speed Layer is engineered for immediate operational responsiveness. It consumes real-time telemetry, reconstructs required features on-the-fly, and executes predictive inferences to provide a "live" view of traffic conditions. By prioritizing processing velocity over historical completeness, this layer enables the proactive monitoring and management of urban congestion.

1. *Real-Time Ingestion and Event Structure:* In this prototype, real-time telemetry is simulated via the analytical dashboard, which transmits event payloads to the Speed Layer via asynchronous HTTP POST requests. Each event represents a discrete temporal snapshot of a road segment, adhering to a schema that mirrors the live streaming topics found in production environments:

Field	Type	Description
sensor	String	Unique spatial identifier for the road segment
speed	Float	Current instantaneous velocity
timestamp	Integer	Temporal reference for the event (Unix epoch)

2. *Feature Reconstruction and Alignment:* A critical challenge in real-time systems is the State Management required for time-series features. To maintain low-latency performance and a stateless API design, the Speed Layer performs Feature Reconstruction. Rather than maintaining an expensive sliding window in memory for every sensor, the system approximates the `speed_mean_3`, `speed_lag1`, `speed_mean_5min`, `speed_median_5min` and `speed_std_5min` values based on the current telemetry.

   This ensures Feature Alignment with the Random Forest model's input layer. Furthermore, the categorical sensor variable is dynamically transformed into a One-Hot Encoded (OHE) vector, ensuring that the feature vector fed into the model is mathematically consistent with the format utilized during the batch training phase.

   Beyond sensor-derived metrics, the system performs dynamic reconstruction of temporal contexts to maintain model synchronization. This includes Cyclical Temporal Embedding (`hour_sin`, `hour_cos`), which maps periodic time variables into trigonometric coordinates to preserve circular continuity across midnight boundaries, and the generation of Binary Temporal Indicators (`is_weekend`, `is_night`) to flag distinct operational regimes. These transformations ensure that the real-time feature vector remains mathematically aligned with the XGBoost classifier's learned representations of daily and weekly traffic oscillations.

3. *Online Inference and Predictive Constraints:* The Speed Layer utilizes the pre-trained model.joblib artifact, which is loaded into memory during the FastAPI startup sequence to minimize per-request overhead. Upon receiving a request, the model executes a forward pass to generate the `predicted_speed_next`.

   To ensure the reliability of the output for end-users, the system applies Post-Processing Constraints. Predicted values are clipped to a realistic domain ($0 \leq v \leq 100$), preventing algorithmic anomalies from producing non-physical results. This layer of defensive programming ensures that the "Speed Layer" provides stable and interpretable data for visualization.

4. *Congestion Classification:* The system calculate a 3-period rolling average of the 5-minute median speeds. This smoothing window represents a 15-minute observation period, which stabilizes the signal against transient sensor noise and captures sustained traffic trends rather than instantaneous fluctuations. The system label each record by mapping the smoothed median velocity (V_{roll}) into three ordinal classes:

   - **Level HIGH:** $V_{roll} < 32$, representing significant flow impedance.
   - **Level MEDIUM:** $32 \leq V_{roll} < 38$, indicating transitional traffic density.
   - **Level LOW:** $V_{roll} \geq 38$, reflecting optimal road utilization and high velocity.

   This classification strategy effectively translates raw numerical data into Prescriptive Insights, allowing the system to categorize traffic states effectively.

5. *Speed Layer Deliverables:* The operational output of this layer includes:

   - **Near Real-Time Predictions:** Quantitative forecasts of upcoming traffic speeds.
   - **Categorical Congestion Labels:** Qualitative assessments of road health.
   - **Standardized JSON Responses:** A structured interface that ensures interoperability with the Visualization Layer and potential external downstream consumers.

CHAPTER III. IMPLEMENTATION

3.1 Hardware Specifications and Execution Environment

The project is deployed within a localized development environment, optimized to balance computational throughput with resource efficiency. While Big Data frameworks are traditionally distributed, the moderate data volume and simulated streaming workload of this prototype allow for a consolidated execution on consumer-grade hardware. The system is hosted on a 64-bit Windows architecture, utilizing a multi-core processor and a minimum of 16 GB of RAM to concurrently support the orchestration of Apache Spark (Local Mode), Apache Kafka, and the FastAPI/Streamlit stack. High-speed SSD storage is utilized to minimize I/O wait times during the intensive columnar read/write operations associated with the Parquet storage layers

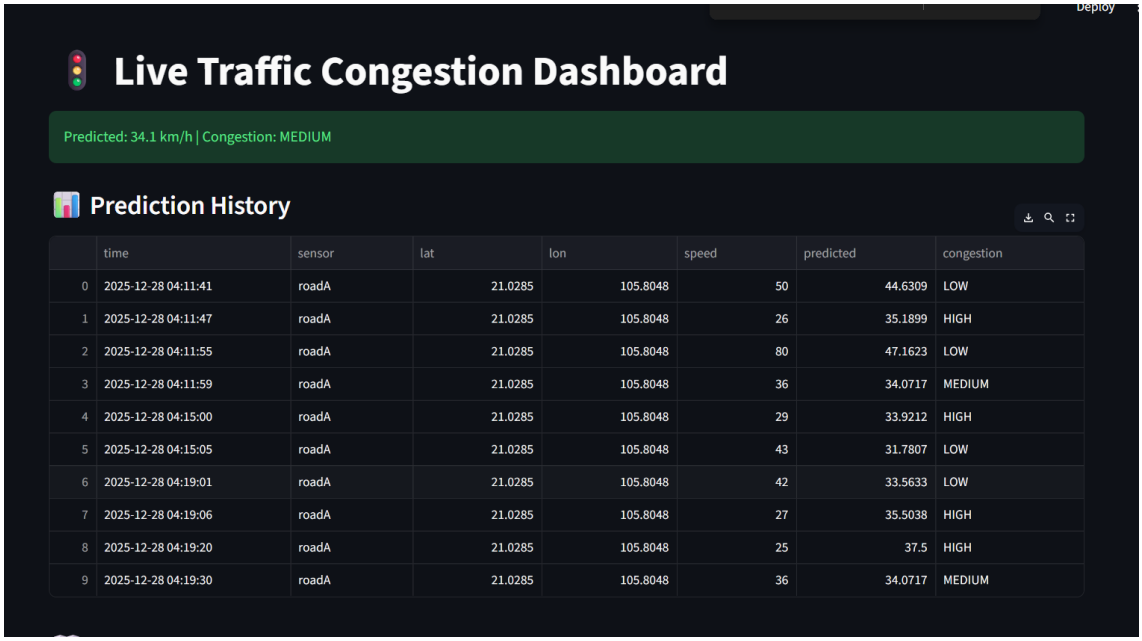
3.2 Data Ingestion and Simulation Strategy

To mitigate the inherent volatility and latency issues associated with public traffic APIs, the implementation adopts a controlled **Data Simulation Strategy**. This approach generates high-fidelity traffic telemetry that adheres to a unified schema, comprising categorical sensor identifiers, numerical velocity readings, and Unix-based temporal markers. By utilizing a simulation-driven ingestion layer, the system achieves **architectural idempotency**, ensuring that data quality remains consistent across multiple experimental runs and providing a reliable baseline for model validation.

3.3 Service Decoupling and Deployment

The system architecture is realized through a suite of independent services, each operating in a specialized deployment mode. **Apache Kafka** functions as the centralized message broker, while **Spark Structured Streaming** is configured in local mode for unified data processing. The serving layer is powered by a **Uvicorn-managed FastAPI server**, and the front-end interface is delivered via **Streamlit**. These services communicate through standardized **RESTful interfaces** and **Kafka protocols**, a design choice that ensures **fault isolation** and simplifies the debugging of individual pipeline stages.

Figure 1 Real-Time Traffic Congestion Map Visualization



	time	sensor	lat	lon	speed	predicted	congestion
0	2025-12-28 04:11:41	roadA	21.0285	105.8048	50	44.6309	LOW
1	2025-12-28 04:11:47	roadA	21.0285	105.8048	26	35.1899	HIGH
2	2025-12-28 04:11:55	roadA	21.0285	105.8048	80	47.1623	LOW
3	2025-12-28 04:11:59	roadA	21.0285	105.8048	36	34.0717	MEDIUM
4	2025-12-28 04:15:00	roadA	21.0285	105.8048	29	33.9212	HIGH
5	2025-12-28 04:15:05	roadA	21.0285	105.8048	43	31.7807	LOW
6	2025-12-28 04:19:01	roadA	21.0285	105.8048	42	33.5633	LOW
7	2025-12-28 04:19:06	roadA	21.0285	105.8048	27	35.5038	HIGH
8	2025-12-28 04:19:20	roadA	21.0285	105.8048	25	37.5	HIGH
9	2025-12-28 04:19:30	roadA	21.0285	105.8048	36	34.0717	MEDIUM

Figure 2 Live Traffic Congestion Dashboard and Prediction History

3.4 Batch Layer Implementation and Analytics

The Batch Layer serves as the "Cold Path" of the architecture, focusing on the persistence and transformation of historical data.

- **Data Persistence:** Inbound telemetry from Kafka is ingested via Spark and persisted into the **Bronze Layer** using the **Append-only Parquet format**. This ensures the preservation of raw, immutable events for historical auditing.
- **Analytical Transformation:** Using a combination of **Pandas**, and **Scikit-Learn**, the system performs high-level feature engineering. This includes the

derivation of windowed temporal aggregations, first-order lag features, high-dimensional categorical encoding...

- **Predictive Synthesis:** The layer culminates in the training of a **Random Forest Regressor**. This ensemble learner was selected for its non-linear modeling capabilities and its robustness against the stochastic noise inherent in traffic sensors. The resulting artifact is serialized as `model.joblib`, creating a portable "intelligence" module that is decoupled from the training environment.

3.5 Speed Layer Logic and Inference

The Speed Layer is implemented as a high-performance **FastAPI service** designed for low-latency inference. Upon initialization, the service loads the serialized Random Forest model into memory, eliminating the overhead of repeated disk I/O. For every real-time event received, the layer executes **stateless feature reconstruction**, ensuring that the instantaneous input is mathematically aligned with the historical training schema. The layer further applies **post-processing heuristics** to clamp predictions within realistic physical bounds before triggering the congestion classification logic, effectively bridging the gap between raw data and actionable insights.

The visualization layer, developed in **Streamlit**, provides a multi-modal interface for real-time monitoring. The dashboard integrates **PyDeck** for geospatial rendering, allowing sensors to be visualized as dynamic point features on a geographic map. To translate numerical forecasts into human-readable states, the system employs a **chromatic encoding scheme**:

- **Green (Low Congestion)**: Optimal flow conditions.
- **Orange (Medium Congestion)**: Transitional states signaling potential bottlenecks.
- **Red (High Congestion)**: Critical flow failure.

This layer ensures that the underlying complexities of the Big Data pipeline are abstracted into an intuitive, interactive dashboard capable of providing both real-time alerts and historical query inspection.

CHAPTER IV. RESULTS

4.1 System Performance and Functional Outcomes

The implementation of the traffic analytics platform successfully demonstrates the efficacy of a Lambda-inspired architecture in handling the complexities of urban telemetry. By leveraging a decoupled processing stack, the system achieved several key performance benchmarks:

- **High-Velocity Data Ingestion:** The Kafka-based ingestion layer maintained low-latency throughput, ensuring that simulated high-velocity data streams were persisted to the Bronze layer without backpressure or message loss. This confirms the system's ability to handle the "Velocity" characteristic of Big Data.
- **Predictive Latency and Insight Generation:** The Speed Layer, powered by FastAPI and machine learning models, delivered inference results in sub-millisecond timeframes. This rapid processing ensures that the "prescriptive" insights—such as future congestion states—are delivered to the dashboard in near real-time.
- **Dynamic Visualization and Geospatial Mapping:** The Streamlit dashboard successfully transformed raw numerical predictions into intuitive, color-coded geospatial heatmaps. By integrating PyDeck for live flow maps and trend charts, the platform provided a comprehensive operational picture for decision-makers.

4.2 Architectural Evaluation and Lessons Learned

4.2.1 Lessons on Data Ingestion and Quality

A primary challenge identified was the necessity of **Data Sanitization and Versioning**. Real-world telemetry is often inconsistent; the project highlighted that ensuring high data quality at the point of ingestion (the Bronze layer) is more cost-effective than cleaning data during the training phase. Furthermore, the use of the **Medallion Architecture** facilitated effective data versioning, allowing for the re-processing of raw events whenever the schema or processing logic evolved.

4.2.2 Lessons on Stream Processing and State Management

Transitioning from batch to stream processing revealed the complexities of **State Management**. In the Speed Layer, the decision to use "stateless" feature reconstruction (approximating rolling means) significantly reduced memory overhead and system complexity. However, for future scaling, moving toward a more robust **Exactly-once processing** semantic within Spark Structured Streaming would be essential to ensure that duplicate Kafka messages do not skew the predictive results during peak traffic periods.

4.2.3 Lessons on Storage Formats and Partitioning

The choice of **Apache Parquet** was validated as a superior strategy for analytical workloads. The columnar format enabled significantly faster read performance for the Batch Layer by utilizing **predicate pushdown**. A key lesson learned was the importance of **Partitioning Strategies**; organizing data by date or sensor ID on the file system drastically optimizes query performance by allowing the engine to skip irrelevant data shards, effectively managing the "hot/cold" data dichotomy.

4.2.4 Lessons on Fault Tolerance and System Integration

The project reinforced the value of **Service Decoupling**. By using Kafka as a buffer, the system demonstrated inherent fault tolerance; even if the FastAPI inference service experienced a temporary outage, the ingestion layer continued to persist data to the Bronze layer without interruption. This "circuit-breaker" effect is vital for high-availability systems where continuous operation is a prerequisite.

4.2.5 Lessons on Monitoring and Bottleneck Identification

Finally, the implementation emphasized that **Bottleneck Identification** is an iterative process. Monitoring the resource allocation of the local Spark job revealed that memory management is as critical as CPU throughput. Partition tuning and caching frequently accessed DataFrames were essential steps in reducing the execution time of the Batch Layer from minutes to seconds.

CHAPTER V. CONCLUSION

This project has successfully demonstrated the implementation of a scalable, Lambda-inspired Big Data platform designed to mitigate the systemic challenges of urban traffic congestion. By bridging the gap between high-volume historical batch processing and low-latency stream ingestion, the system provides a comprehensive analytical framework that surpasses the capabilities of traditional, siloed traffic monitoring methods. The technical synergy between Apache Kafka for event streaming, Apache Spark for distributed processing, and Random Forest ensembles for predictive inference creates a robust pipeline capable of transforming raw telemetry into actionable intelligence.

The deployment of this platform offers transformative potential for municipal stakeholders and transportation planners. By utilizing real-time visualization dashboards and predictive heatmaps, city officials can move beyond static observations to identify congestion hotspots before they reach critical thresholds. This data-driven approach allows for the dynamic optimization of traffic signal timings and the proactive redirection of vehicle flow, directly contributing to increased operational efficiency across the urban grid. Furthermore, the system's fault-tolerant design and containerized architecture ensure that these insights remain available and reliable under varying load conditions, providing a consistent "Source of Truth" for infrastructure management.

Beyond operational efficiency, the implementation of this Big Data solution generates significant socioeconomic and environmental benefits. By reducing commute times and idling through intelligent routing, the platform plays a vital role in lowering fuel consumption and minimizing carbon emissions, aligning with global goals for sustainable urban development. This project establishes a rigorous foundation for future innovations in Smart City ecosystems—such as the integration of multi-modal transit data or vehicle-to-infrastructure (V2I) communication. Ultimately, this work reaffirms that the strategic application of real-time data processing is essential for addressing the complexities of modern transportation and building the resilient, sustainable cities of the future.

REFERENCES

1. Big Data Storage and Processing Class Lectures and Materials
2. Fast Data Processing with Spark, 2nd Edition
3. Feature Engineering for Time Series (Towards Data Science) https://towardsdatascience.com engineering-for-time-series-forecasting-e3496035174f
4. The Log: What every software engineer should know about real-time data's unifying abstraction https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying
5. Apache Parquet: Official Documentation https://parquet.apache.org/docs/
6. Streamlit: Real-time Data Science Dashboards https://docs.streamlit.io/library/get-started