Building a Scalable 3-Tier Data Lakehouse for Mobility Analysis in Spain

Big Data Engineering Project

María López Hernández **Joan Sánchez Verdú**
Fernando Blanco Membrives

January 7, 2026

Abstract

Abstract. The rapid digitalization of urban transport has generated massive volumes of mobility data, rendering traditional analytical workflows obsolete. This paper presents a scalable 3-tier Data Lakehouse architecture designed to analyze large-scale Spanish mobility data. By integrating high-volume Origin-Destination matrices from MITMA with INE socio-economic indicators, the system provides a robust foundation for urban analytics. The infrastructure leverages a serverless stack—utilizing *DuckDB* for vectorized processing and *DuckLake* for ACID transactions—orchestrated via Apache Airflow. We validate the architecture through three analytical use cases: temporal pattern clustering, infrastructure gap detection via gravity models, and functional zoning classification, demonstrating a cost-effective solution for data-driven transport planning.

1 Introduction

The availability of high-resolution mobility data from the Spanish Ministry of Transport (MITMA) has opened new avenues for urban planning. However, the sheer volume of these datasets (comprising billions of daily trip records) presents significant engineering challenges that exceed the capabilities of traditional analytical tools. Transport experts often lack the scalable infrastructure required to transform this raw big data into actionable insights.

This project addresses this gap by implementing a Data Lakehouse architecture. By combining the cost-efficiency of data lakes with the transactional integrity of data warehouses, we propose a solution built on DuckDB and DuckLake. This infrastructure allows for the efficient processing of large-scale mobility matrices, enriching them with socio-economic context without the overhead of enterprise systems.

1.1 Objectives

The primary goal is to build a robust, reproducible data platform. The specific objectives are:

- **Scalable Architecture:** Implement a 3-tier Lakehouse (Bronze, Silver, Gold) that decouples storage (S3) from compute (AWS Batch) to optimize costs and performance.
- **Data Integration:** Automate ELT pipelines to clean and fuse heterogeneous sources, including mobility flows, census demographics, and vector geometries.
- **Analytical Value:** Demonstrate the system's utility through advanced use cases, such as gravity models for infrastructure analysis and functional zoning classification.

2 Data Sources

The data lakehouse architecture is designed to ingest and integrate public domain data exclusively, ensuring reproducibility and open access. The following subsections detail the specific datasets and sources selected to build the information system.

2.1 MITMA Open Data

The core mobility dataset is sourced from the Spanish Ministry of Transport, Mobility and Urban Agenda (MITMA [1]), specifically the *Estudios Básicos de Movilidad*. This dataset provides high-resolution insights into the daily movements of residents derived from mobile network big data. From the MITMA the following files have been used:

- ***(202301-202312)_ Viajes_ municipios.tar*:** These files, located in the MITMA web at *estudios_ basicos/por-municipios/viajes/meses-completos/*, contain the information of the mobility from an origin to a destination municipality, per day, per hour, for the year 2023.
- ***nombres_ municipios.csv*:** This file, located in the MITMA web at *zonificacion/zonificacion_ municipios/*, contains the name of each municipality.
- ***relacion_ ine_ zonificacionMitma.csv*:** This file, located in the MITMA web at *zonificacion/*, cointains the mapping between the MITMA codes and the INE codes for the municipalities.
- ***poblacion.csv*:** This file, located in the MITMA web at *zonificacion/*, cointains the population of each municipality per year.
- ***zonificacion_ municipios.shp*:** This ESRI Shapefile (along with its *.dbf*, *.shx*, and *.prj* components), located in the MITMA web at *zonificacion/zonificacion_ municipios/*, contains the official geospatial geometries (polygons) for the municipal study zones.

2.2 INE Demographics and Economics

To contextualize mobility flows and enable the gravity model analysis, mobility data is enriched with official statistics from the Spanish National Statistics Institute (INE [2]). The statistics used from INE are the following:

- ***ine_ rent_ municipalities.csv*:** This file, located in the INE web at *`https://www.ine.es/jaxiT3/files/t/es/csv_bd/`*, contains the rent information per year and municipality.

2.3 Open Data

To categorize mobility patterns by day type (e.g., distinguishing standard workdays from holidays), specific calendar data is integrated from Open Data portals.

- ***calendario_ laboral.csv*:** This file, sourced from the Madrid City Council Open Data Portal (*Datos Abiertos Madrid* [4]) at *`https://datos.madrid.es/`*, contains the official working calendar. It provides the classification of dates (working days, weekends, and holidays) from 2013 to 2026 (included).

3 Lakehouse Architecture

The system follows the Medallion Architecture pattern, decoupled into storage and compute layers to ensure scalability.

3.1 Technology Stack

The platform employs a modular architecture designed to decouple storage from compute, leveraging the following core components:

- **Compute Engine (DuckDB [5]):** A high-performance, in-process SQL OLAP engine. It handles vectorized data processing and supports out-of-core execution, allowing for the analysis of datasets larger than available RAM.
- **Elastic Compute (AWS Batch [6]):** Provides an ephemeral, scalable cloud compute layer. It offloads heavy transformation tasks to on-demand instances, optimizing costs through the use of Spot market resources.
- **Lakehouse Management (DuckLake [7]):** A transaction manager that enables ACID properties and snapshot isolation over object storage, ensuring data consistency during concurrent operations.
- **Cloud Storage & Catalog (Neon + S3):** A decoupled storage design where **AWS S3** holds the physical data files (Parquet/CSV) and **Neon** (Serverless Postgres) acts as the centralized metadata catalog and locking mechanism.
- **Orchestration (Apache Airflow [8]):** Manages the end-to-end data lifecycle via Directed Acyclic Graphs (DAGs), handling complex dependencies, scheduling, and automated retries.

3.2 Data Layering Strategy

The platform follows a "Medallion" Lakehouse architecture (Bronze → Silver → Gold). This section serves as the technical reference for the data models, detailing schema definitions, data types (DuckDB dialect), and transformation logic.

3.2.1 Bronze Layer (Raw Ingestion)

The Bronze layer acts as the immutable staging area. Data is ingested from external sources (MITMA, INE, Madrid Open Data) in its native format. To prevent pipeline failures caused by unexpected data types in the source files, strict typing is deferred to the Silver layer; consequently, almost all columns in this layer are typed as `VARCHAR`, preserving the original formatting (e.g., decimal commas, date formats).

In addition, to ensure auditability and lineage, every table in this layer includes two system columns:

- `ingestion_timestamp` (`TIMESTAMP`): The UTC timestamp of when the record was inserted.
- `source_url` (`VARCHAR`): The specific file URI or API endpoint origin.

Below is the complete dictionary of tables persisted in the Bronze Schema:

1. Mobility Data (High Volume)

`bronze_mobility_data`

Description: Stores the raw daily origin-destination matrices. This table is physically partitioned by the `fecha` column.

- `fecha` (`VARCHAR`): Date of the trip (Format: YYYYMMDD).
- `periodo` (`VARCHAR`): Hourly interval (00-23).
- `origen` (`VARCHAR`): Source MITMA Zone ID.
- `destino` (`VARCHAR`): Destination MITMA Zone ID.
- `distancia` (`VARCHAR`): Distance category (e.g., "005-010").
- `actividad_origen` (`VARCHAR`): Imputed purpose at origin (e.g., "casa").
- `actividad_destino` (`VARCHAR`): Imputed purpose at destination.
- `estudio_origen_posible` (`VARCHAR`): Auxiliary study flag.
- `estudio_destino_posible` (`VARCHAR`): Auxiliary study flag.
- `residencia` (`VARCHAR`): Zone ID of the traveler's residence.
- `renta` (`VARCHAR`): Income decile of the traveler.
- `edad` (`VARCHAR`): Age group interval.
- `sexo` (`VARCHAR`): Gender (1/2).
- `viajes` (`VARCHAR`): The expansion factor/number of trips (String with comma decimals).
- `viajes_km` (`VARCHAR`): Total kilometers traveled.

2. Spatial & Reference Data

`bronze_geo_municipalities`

Description: Ingested from ESRI Shapefiles. This is the only table containing a native spatial type upon ingestion.

- `ID` (`VARCHAR`): The MITMA Zone Code.
- `geom` (`GEOMETRY`): The binary geometry object (Polygon/MultiPolygon).

`bronze_zoning_municipalities`

Description: Provides the mapping between numerical IDs and text names.

- `column0` (`VARCHAR`): Index column from raw CSV.
- `ID` (`VARCHAR`): MITMA Zone Code.
- `name` (`VARCHAR`): Official Municipality Name.
- `filename` (`VARCHAR`): Name of the source file.

`bronze_mapping_ine_mitma`

Description: A crosswalk table linking Transport Ministry codes (MITMA) with Statistics Institute codes (INE).

- `seccion_ine` (`VARCHAR`): Census Section ID.
- `distrito_ine` (`VARCHAR`): Census District ID.
- `municipio_ine` (`VARCHAR`): INE Municipality Code.
- `distrito_mitma` (`VARCHAR`): Transport District ID.
- `municipio_mitma` (`VARCHAR`): Transport Municipality ID.

- `gau_mitma` (`VARCHAR`): Great Urban Area (GAU) code.
- `filename` (`VARCHAR`): Name of the source file.

3. Socio-Economic & Temporal Data

`bronze_population_municipalities`

Description: Raw population counts. Note the lack of headers in the source file.

- `column0` (`VARCHAR`): Zone Code.
- `column1` (`VARCHAR`): Population count.
- `filename` (`VARCHAR`): Name of the source file.

`bronze_ine_rent_municipalities`

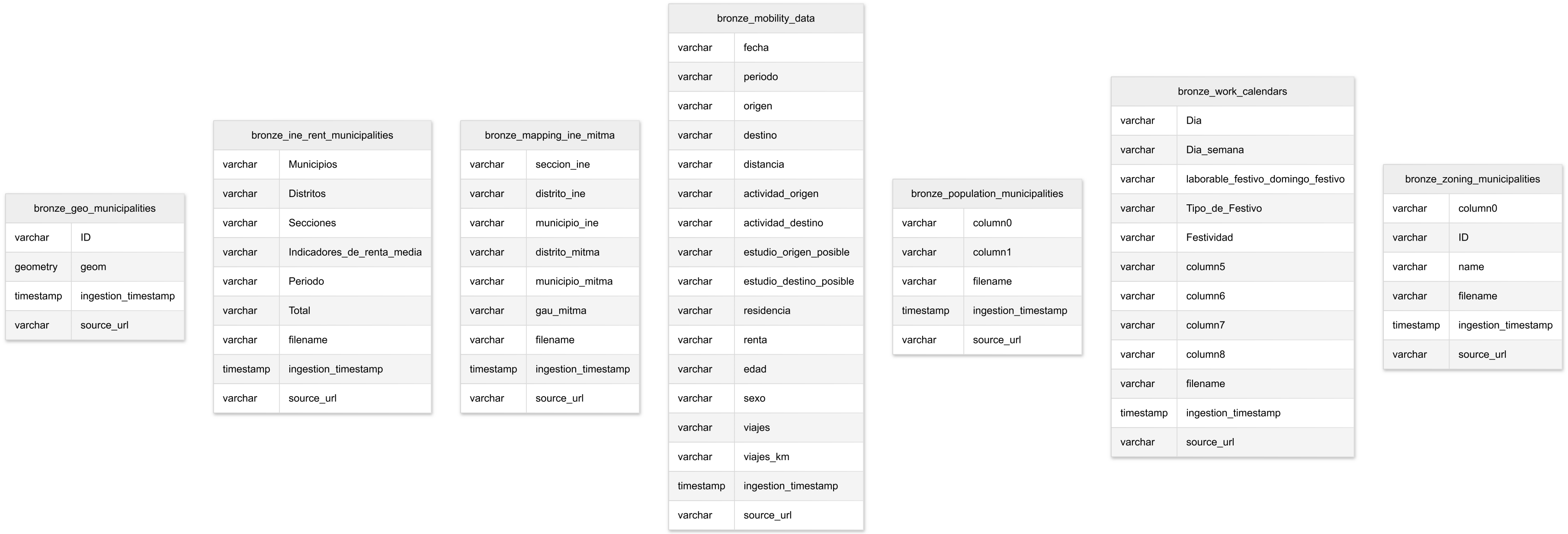
Description: Economic indicators from INE.

- `Municipios` (`VARCHAR`): Composite string (Code + Name).
- `Distritos` (`VARCHAR`): District ID (if applicable).
- `Secciones` (`VARCHAR`): Section ID (if applicable).
- `Indicadores_de_renta_media` (`VARCHAR`): The name of the metric.
- `Periodo` (`VARCHAR`): Reference year.
- `Total` (`VARCHAR`): The numeric value (contains dots for thousands).
- `filename` (`VARCHAR`): Name of the source file.

`bronze_work_calendars`

Description: Calendar data for holiday identification. Extra columns (column5-8) represent empty fields often found in loosely structured CSVs.

- `Dia` (`VARCHAR`): Date string (DD/MM/YYYY).
- `Dia_semana` (`VARCHAR`): Name of the weekday.
- `laborable_festivo_domingo_festivo` (`VARCHAR`): Workday status.
- `Tipo_de_Festivo` (`VARCHAR`): Flag for National/Local holidays.
- `Festividad` (`VARCHAR`): Name of the holiday.
- `column5`, `column6`, `column7`, `column8` (`VARCHAR`): Parsing artifacts from raw CSV.
- `filename` (`VARCHAR`): Name of the source file.



bronze_geo_municipalities	
varchar	ID
geometry	geom
timestamp	ingestion_timestamp
varchar	source_url

bronze_ine_rent_municipalities	
varchar	Municipios
varchar	Distritos
varchar	Secciones
varchar	Indicadores_de_renta_media
varchar	Periodo
varchar	Total
varchar	filename
timestamp	ingestion_timestamp
varchar	source_url

bronze_mapping_ine_mitma	
varchar	seccion_ine
varchar	distrito_ine
varchar	municipio_ine
varchar	distrito_mitma
varchar	municipio_mitma
varchar	gau_mitma
varchar	filename
timestamp	ingestion_timestamp
varchar	source_url

bronze_mobility_data	
varchar	fecha
varchar	periodo
varchar	origen
varchar	destino
varchar	distancia
varchar	actividad_origen
varchar	actividad_destino
varchar	estudio_origen_posible
varchar	estudio_destino_posible
varchar	residencia
varchar	renta
varchar	edad
varchar	sexo
varchar	viajes
varchar	viajes_km
timestamp	ingestion_timestamp
varchar	source_url

bronze_population_municipalities	
varchar	column0
varchar	column1
varchar	filename
timestamp	ingestion_timestamp
varchar	source_url

bronze_work_calendars	
varchar	Dia
varchar	Dia_semana
varchar	laborable_festivo_domingo_festivo
varchar	Tipo_de_Festivo
varchar	Festividad
varchar	column5
varchar	column6
varchar	column7
varchar	column8
varchar	filename
timestamp	ingestion_timestamp
varchar	source_url

bronze_zoning_municipalities	
varchar	column0
varchar	ID
varchar	name
varchar	filename
timestamp	ingestion_timestamp
varchar	source_url

Figure 1: Entity Diagram of the Bronze Layer. One table for each file.

3.2.2 Silver Layer (Cleaned & Integrated)

The Silver layer implements a Star Schema design. In this stage, data is cast to strict types, cleaned of formatting artifacts (e.g., removing thousands separators), and enriched with spatial calculations. All tables include a `processed_at` (`TIMESTAMP`) column for versioning.

1. Dimension Tables (Context)

`silver.dim_zones`
Purpose: The central spatial authority table.
Transformation: Joins `geo`, `zoning`, and `mapping` tables. Generates a Surrogate Key.

- `zone_id` (`BIGINT`): Sequential integer generated via `ROW_NUMBER()`.
- `mitma_code` (`VARCHAR`): Original Transport ID.
- `ine_code` (`VARCHAR`): National Statistics ID (Cleaned).
- `zone_name` (`VARCHAR`): Human-readable name.
- `polygon` (`GEOMETRY`): Boundary for spatial joins.
- `centroid` (`GEOMETRY`): Calculated center (`ST_Centroid`) for distance logic.

`silver.dim_zone_distances`
Purpose: Pre-computed Distance Matrix ($N \times N$).
Transformation: Cartesian product of `dim_zones`.

- `origin_zone_id` (`BIGINT`): FK ref `dim_zones`.
- `destination_zone_id` (`BIGINT`): FK ref `dim_zones`.
- `dist_km` (`DOUBLE`): Euclidean distance in km. **Logic:** `GREATEST(0.5, dist)` ensures no zero-distances to prevent division errors in Gravity Models.

`silver.dim_zone_holidays`
Purpose: Temporal context for mobility patterns.
Transformation: Filters `work_calendars` for "National Holidays" in 2023 and explodes them to all zones via Cross Join.

- `zone_id` (`BIGINT`): FK ref `dim_zones`.
- `holiday_date` (`DATE`): The specific date of the holiday.

2. Metric Tables (Auxiliary Data)

`silver.metric_population`
Grain: One row per Zone per Year.

- `zone_id` (`BIGINT`): FK ref `dim_zones`.
- `population` (`BIGINT`): Cast from `column1`. **Cleaning:** Regex filter `[a-zA-Z]` removes header rows from raw CSV.
- `year` (`INTEGER`): Reference year (2023).

`silver.metric_ine_rent`
Grain: One row per Zone per Year.

- `zone_id` (`BIGINT`): FK ref `dim_zones`.

- `income_per_capita` (`DOUBLE`): Cast from `Total`. **Cleaning:** Removes dots (".") and filters only "Renta neta media por persona".
- `year` (`INTEGER`): Reference year.

3. Fact Table (Core Mobility)

`silver.fact_mobility`

Purpose: The transactional heart of the Lakehouse.
Grain: One row per trip flow (Origin $\rightarrow$ Dest $\rightarrow$ Time).
Transformation: Converts string dates/hours into Timezoned Timestamps. Filters out invalid zones via Inner Joins to `dim_zones`.

- `period` (`TIMESTAMP WITH TIME ZONE`): **Logic:** `fecha` + `periodo` converted to "Europe/Madrid".
- `partition_date` (`DATE`): Partition Key derived from `fecha`.
- `origin_zone_id` (`BIGINT`): FK ref `dim_zones`.
- `destination_zone_id` (`BIGINT`): FK ref `dim_zones`.
- `trips` (`DOUBLE`): Number of trips cast to double precision.

4. Data Observability

`silver.data_quality_log`

Purpose: Metadata registry for pipeline health.

- `check_timestamp` (`TIMESTAMP`): Audit time.
- `table_name` (`VARCHAR`): Target of the check.
- `metric_name` (`VARCHAR`): e.g., "null_rate", "row_count".
- `metric_value` (`DOUBLE`): The result of the check.
- `notes` (`VARCHAR`): Context or warnings.

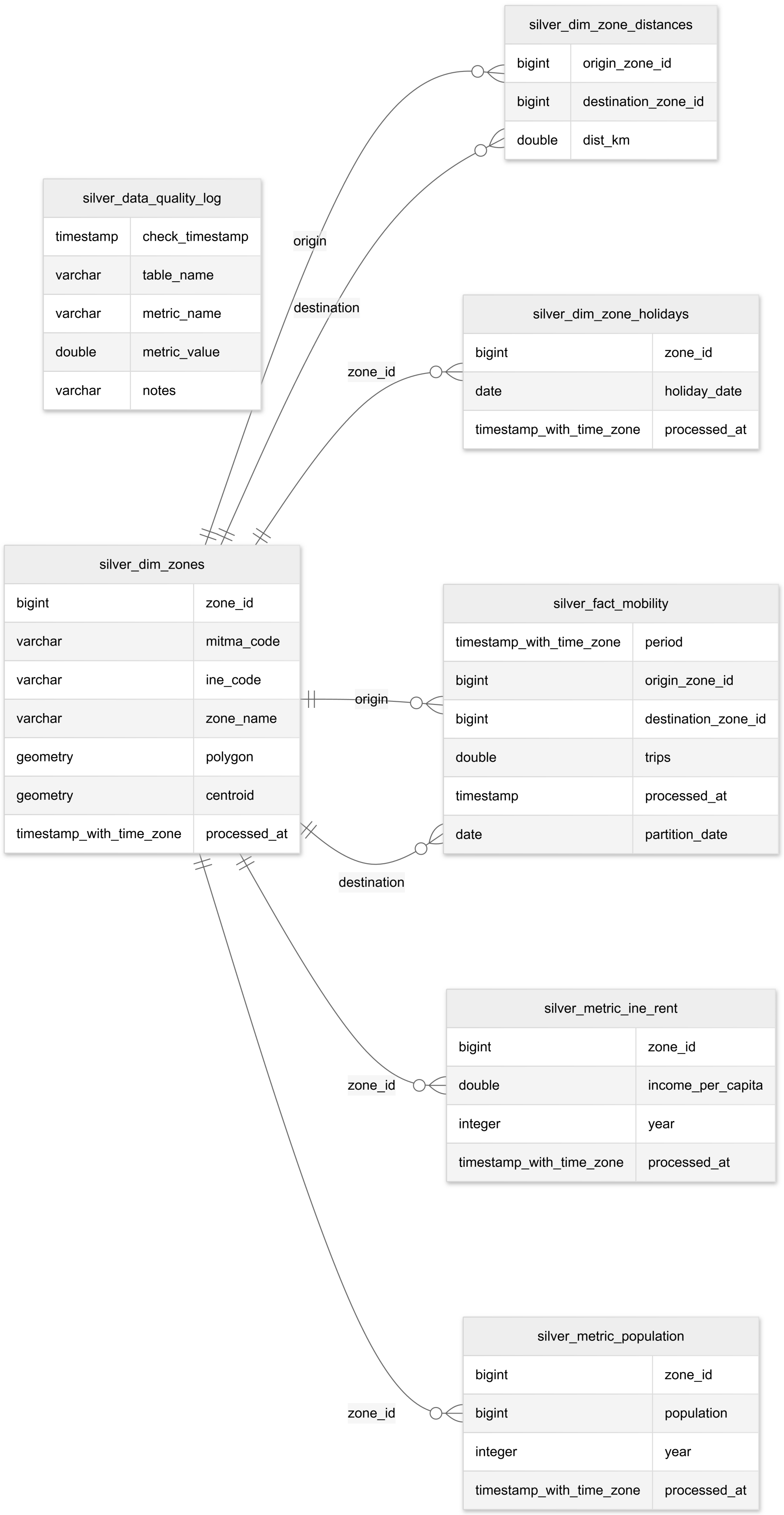


Figure 2: Entity-Relationship Diagram (ERD) of the Silver Layer. The schema follows a Star Schema design, centering on the `fact_mobility` table linked to spatial (`dim_zones`) and temporal dimensions.

3.2.3 Gold Layer (Data Mart)

The Gold layer is the final destination for analytics, designed to answer specific business questions. Unlike the normalized Silver layer, these tables are denormalized and pre-aggregated to optimize query performance for visualization tools (Kepler.gl, Plotly).

Below is the technical definition of the three core analytical products persisted in this layer:

1. Temporal Analysis (Use Case 1)

`gold.typical_day_patterns`

Purpose: Stores the centroids of the mobility profiles identified via Unsupervised Learning (K-Means). It reduces billions of raw records into lightweight hourly curves representing standard behaviors (e.g., "Workday" vs "Holiday"). *Grain:* One row per Cluster per Hour.

- `cluster_id` (`INTEGER`): The label assigned by the K-Means algorithm (0, 1, or 2).
- `hour` (`INTEGER`): Hour of the day (0-23).
- `avg_trips` (`DOUBLE`): The average normalized volume of trips for this specific hour/-cluster combination. Used for plotting demand curves.
- `processed_at` (`TIMESTAMP`): Audit timestamp.

2. Infrastructure Gap Analysis (Use Case 2)

`gold.infrastructure_gaps`

Purpose: The output of the Gravity Model. This table joins mobility flows with socio-economic dimensions to quantify the disparity between theoretical potential and actual usage. *Grain:* One row per Origin-Destination pair.

- `org_zone_id` (`BIGINT`): Origin Zone FK.
- `dest_zone_id` (`BIGINT`): Destination Zone FK.
- `total_population` (`BIGINT`): Population at origin (Driver of demand).
- `rent` (`DOUBLE`): Income per capita at destination (Attractor).
- `total_trips` (`DOUBLE`): The actual observed mobility volume from Silver.
- `dist_km` (`DOUBLE`): Distance impedance between zones.
- `mismatch_ratio` (`DOUBLE`): The key analytical KPI calculated as $ActualTrips/PotentialScore$. A value $\ll 1.0$ indicates an infrastructure deficit.

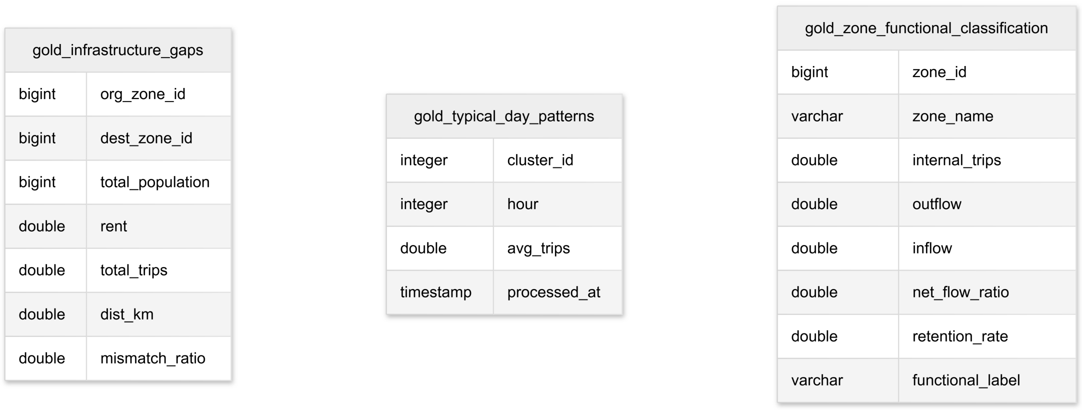
3. Functional Classification (Use Case 3)

`gold.zone_functional_classification`

Purpose: A semantic layer that profiles every municipality based on its role in the metropolitan network (Importer vs. Exporter of commuters). *Grain:* One row per Zone.

- `zone_id` (`BIGINT`): Id of the zone.
- `zone_name` (`VARCHAR`): Human-readable name for map tooltips.
- `internal_trips` (`DOUBLE`): Count of trips starting and ending in the same zone.
- `outflow` (`DOUBLE`): Count of trips leaving the zone.
- `inflow` (`DOUBLE`): Count of trips entering the zone.

- `net_flow_ratio` (DOUBLE): Calculated metric: $(In - Out)/Total$. Positive values indicate “Activity Hubs”; negative values indicate “Residential/Bedroom Communities”.
- `retention_rate` (DOUBLE): Calculated metric: $Internal/(Out + Internal)$. Indicates self-sufficiency.
- `functional_label` (VARCHAR): The final categorical classification derived from the decision tree logic (e.g., “Self-Sustaining Cell”, “Activity Hub”).



gold_infrastructure_gaps	
bigint	org_zone_id
bigint	dest_zone_id
bigint	total_population
double	rent
double	total_trips
double	dist_km
double	mismatch_ratio

gold_typical_day_patterns	
integer	cluster_id
integer	hour
double	avg_trips
timestamp	processed_at

gold_zone_functional_classification	
bigint	zone_id
varchar	zone_name
double	internal_trips
double	outflow
double	inflow
double	net_flow_ratio
double	retention_rate
varchar	functional_label

Figure 3: Gold Layer Schema. These tables represent the final analytical products, denormalized and enriched with derived metrics (e.g., mismatch ratios, functional labels) ready for visualization.

4 Implementation Methodology

The operational logic of the Data Lakehouse is implemented through a modular, automated workflow orchestrated by Apache Airflow [8]. The system adopts a decoupled architecture where distinct Directed Acyclic Graphs (DAGs) handle specific stages of the data lifecycle based on their update frequency and functional scope.

This section details the engineering strategies employed to ensure scalable data ingestion, atomic processing of daily batches, and the automated generation of analytical models. The implementation prioritizes robustness through strict transaction management, task isolation, and the separation of heavy computational ELT from on-demand reporting queries.

4.1 Orchestration Strategy

To manage the dependencies between the Bronze, Silver, and Gold layers efficiently, the system is architected into a multi-DAG ecosystem. This design allows for independent scaling and maintenance of static dimensions versus high-volume mobility facts.

1. **Infrastructure & Dimensions DAG:** Handles the ingestion of low-frequency static data (INE demographics, MITMA zoning, Calendars). It establishes the schema foundations and Bronze/Silver dimension tables.
2. **Mobility Ingestion DAG:** A parameterized worker pipeline designed for high-volume processing. It accepts date ranges to ingest, clean, and transform the daily mobility files (MITMA OD Matrices) from Bronze to Silver using atomic tasks.
3. **Gold Generations and consultings DAGs:** These pipelines (DAGs 31–33) consolidate the logic into unified flows. Triggered on-demand with custom parameters (e.g., specific

spatial polygons or date ranges), they first materialize the aggregated Gold tables using DuckDB and immediately generate the final visual assets (Kepler.gl maps, Matplotlib charts), uploading them to S3 in a single execution context.

4.2 Infrastructure & Dimensions (DAG 1)

The foundational pipeline, `infrastructure_and_dimensions`, is responsible for establishing the Lakehouse schema and ingesting low-velocity reference data. Unlike the daily mobility processing, this DAG is designed to be executed on-demand or annually, as zoning definitions and census statistics change infrequently.

Before data movement begins, the pipeline ensures the environment is consistent. The task called `create_schemas` connects to the Neon catalog to verify the existence of the logical namespaces (*bronze, silver, gold*). Simultaneously, the `create_stats_table` task initializes the `data_quality_log` table in the Silver layer, which is a prerequisite for audit logging in all subsequent pipelines.

To optimize total execution time, all external data extraction tasks are configured to run in parallel. These tasks use the Airflow *@task* notation arguments to specify 3 retries with a gap of 1 minute in case they fail. This phase handles heterogenous data formats through specialized ingestion logic:

- **Spatial Ingestion:** The `br_ingest_geo_data` task utilizes DuckDB's `spatial` extension. It performs an HTTP HEAD request to validate the existence of the remote Shapefiles (.shp, .shx, .dbf) before executing `ST_Read` to ingest the vector geometry directly into the Bronze layer.
- **Statistical Ingestion:** Tasks targeting INE data (e.g., `br_ingest_ine_rent`) implement specific handling for legacy formats, forcing `ISO-8859-1` encoding and tab separators to prevent parsing errors common in Spanish government datasets.
- **Dictionaries:** Tasks for zoning and mapping ingest standard CSVs via DuckDB's `read_csv_auto`, adding audit columns (`source_url`, `ingestion_timestamp`) on the fly.

The dependency structure in the Silver layer follows a *Funnel* pattern (Figure 4). The system enforces a hard dependency on the creation of the master dimension table before populating satellite metric tables.

1. **Convergence Point (`dim_zones`):** This task acts as the synchronization barrier. It waits for the completion of three specific Bronze tasks—Geometry, Zoning names, and INE Mapping. Once available, it performs a 3-way join to construct the master `dim_zones` table, generating the surrogate `zone_id`.
2. **Satellite Expansion:** Once the master dimension exists, the downstream tasks called `metric_population`, `metric_ine_rent`, `dim_zone_distance_matrix` and `dim_zone_holidays` trigger in parallel. Each task performs a join between its respective raw Bronze source and the newly created `dim_zones` to assign the correct surrogate keys to the statistical data.

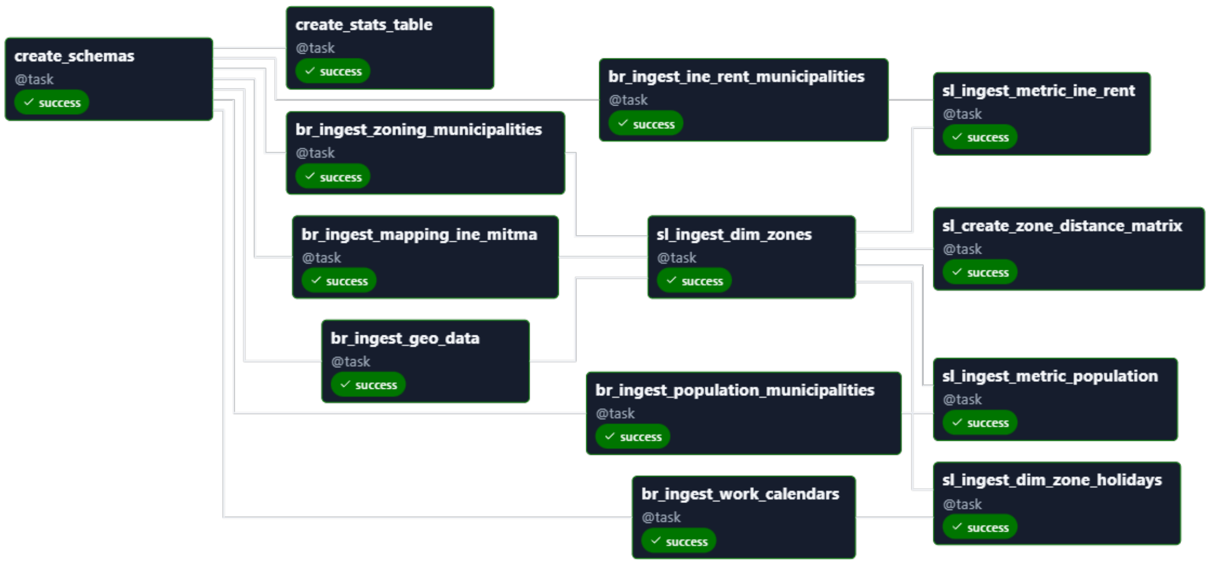


Figure 4: Dependency graph of the Infrastructure DAG.

4.3 Mobility Ingestion (DAG 2)

The second pipeline, `mobility_ingestion`, constitutes the core factory of the Lakehouse. It is responsible for the daily ingestion and transformation of the high-volume Origin-Destination matrices. Given that the source data is partitioned by day (one compressed CSV per day), this DAG utilizes Airflow's Dynamic Task Mapping to scale horizontally, processing date ranges supplied via run-time parameters.

Unlike the static infrastructure DAG, this workflow begins by generating a execution plan based on user input. The `generate_date_list` task parses the `start_date` and `end_date` parameters to produce a list of strings (e.g., `['20230101', '20230102', ...]`).

Simultaneously, the pipeline executes idempotent initialization tasks (`ensure_br_mobility...` and `ensure_sl_mobility...`). These tasks utilize `CREATE TABLE IF NOT EXISTS` logic to prepare the partitioning structures in the Bronze and Silver layers, ensuring that parallel workers do not encounter race conditions when attempting to write to non-existent tables.

The `br_process_single_day` task is mapped over the generated date list. Each instance handles a specific day, implementing robust checks to handle the known inconsistencies in the MITMA source server:

- **Pre-flight Validation:** Before attempting extraction, the task performs an HTTP `HEAD` request to the calculated URL. If the source file is missing (a common occurrence for specific dates in late 2023), the task logs a warning and skips execution gracefully rather than failing the pipeline.
- **Throttling and Retries:** To prevent overwhelming the source server or saturating local memory, concurrency is strictly limited via `max_active_tis_per_dag=2`. Network resilience is enforced with a retry policy of 5 attempts with a 30-second delay.
- **Streaming Ingestion:** Valid files are streamed directly from the remote server into the `mobility_data` table using DuckDB's `read_csv_auto`, bypassing local disk storage.

The transformation phase (`sl_process_single_day`) mirrors the mapping of the ingestion phase but introduces a strict cross-layer dependency: Silver processing for the batch only begins once Bronze ingestion is confirmed.

Inside this task, raw text data is transformed into the analytical schema. This involves joining the raw mobility records with the `dim_zones` table (created in DAG 1) to resolve MITMA/INE codes into internal surrogate keys. This effectively acts as a foreign key validation constraint; records with invalid zone codes are implicitly filtered during the inner join, ensuring referential integrity in the Silver layer.

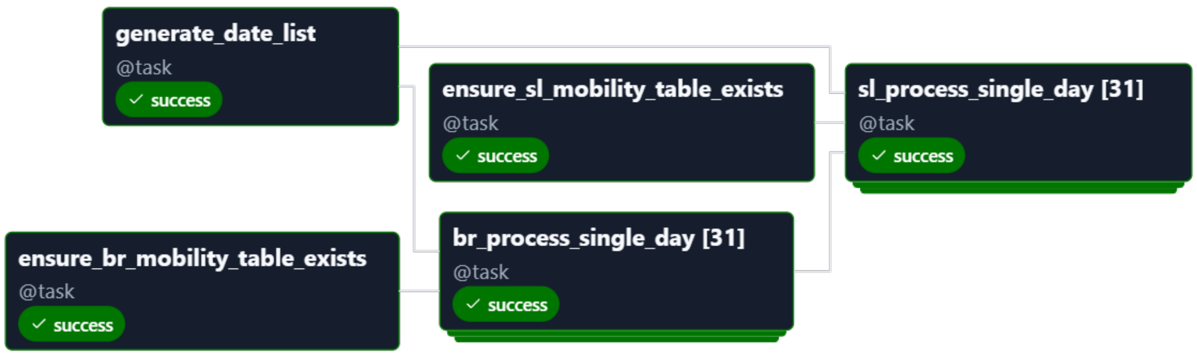

generate_date_list
@task
success
ensure_sl_mobility_table_exists
@task
success
sl_process_single_day [31]
@task
success
br_process_single_day [31]
@task
success
ensure_br_mobility_table_exists
@task
success

Figure 5: The Mobility Ingestion DAG utilizing Dynamic Task Mapping.

4.4 Gold Generations and consultings (DAGs 3)

The analytical layer is materialized through three dedicated Airflow pipelines. Each DAG is designed to answer a specific business question by transforming Silver data into aggregated Gold tables and subsequently generating visual reports stored in S3.

- **DAG 31: Typical Day Clustering** (`31_bq1_clustering`)
  This pipeline addresses the characterization of mobility patterns via unsupervised learning.
  - **Transformation:** It aggregates mobility data to extract hourly profiles and applies K-Means clustering ($k = 3$) using Scikit-Learn to categorize days. The pipeline uses two tables: `gold.typical_day_patterns` and `gold.typical_od_matrices` for spatial distribution.
  - **Output:** The DAG produces a comprehensive reporting suite in the results directory, including a static Matplotlib chart for temporal profiles, an interactive Plotly HTML heatmap for detailed OD analysis, and a summary Markdown report.
- **DAG 32: Infrastructure Gap Analysis** (`32_bq2_gaps`)
  This pipeline implements a Gravity Model to quantify the disparity between theoretical potential and actual mobility.
  - **Transformation:** It executes a SQL batch job to create `gold.infrastructure_gaps` table. By integrating population, income, and distance metrics, it calculates a theoretical "Gravity Score" ($P \cdot E/d^2$) and compares it with actual flows to derive a `mismatch_ratio` for every connection.
  - **Output:** The DAG utilizes Kepler.gl to generate two interactive geospatial reports: a *Service Level Ranking* (bubble map) identifying underserved zones and a *Mobility Gaps* visualization (arc map) showing specific connections where demand exceeds supply, alongside a summary Markdown report.
- **DAG 33: Functional Classification** (`33_bq3_functional_classification`)
  This pipeline categorizes municipalities based on their role in the metropolitan network (e.g., residential vs. commercial hubs).

– **Transformation:** It calculates critical flow metrics (inflow/outflow ratios and retention rates) via a SQL batch job. A logic-based decision tree assigns a functional label (e.g., "Bedroom Community", "Activity Hub") to each zone, persisting the results in `gold.zone_functional_classification`.

– **Output:** The DAG generates a Kepler.gl Choropleth map that colors zone geometries by their functional role and overlays an activity intensity layer (bubbles). This interactive dashboard, accompanied by a Markdown summary, allows urban planners to visualize the structural hierarchy of the region.

5 Results and Use Cases

5.1 Use Case 1: Typical Mobility Patterns and Profiling

The first business objective focuses on identifying and characterizing standard mobility behaviors within the reference year (2023). Instead of relying on static calendar rules (e.g., assuming all Mondays are identical), we implemented an unsupervised machine learning approach in the Gold layer to discover intrinsic daily patterns—distinguishing between standard workdays, holidays, and transition days based on actual hourly flow shapes.

Implementation Approach

The solution is orchestrated via an Airflow DAG (`31_bq1_clustering`). The pipeline employs a hybrid processing strategy: using DuckDB for heavy aggregations and Python (Scikit-Learn) for in-memory clustering. The logic follows a four-step process:

1. **Feature Extraction:** We aggregate the granular records from `silver.fact_mobility` into hourly profiles. To ensure the clustering algorithm focuses on the *shape* of the day (e.g., morning/evening peaks) rather than absolute volume, we normalize the data vector for every date d:
$$V_{d,h} = \frac{T_{d,h}}{\sum_{h=0}^{23} T_{d,h}} \tag{1}$$
where $T_{d,h}$ is the total trips on day d at hour h.

2. **Unsupervised Classification (K-Means):** These normalized vectors are fed into a K-Means algorithm (configured with $k = 3$ clusters). This automatically classifies every date in the analyzed period into distinct profiles (e.g., "Standard Workday," "Weekend/Leisure," or "Holiday").

3. **Pattern Projection:** Once dates are labeled, the system projects these labels back onto the massive dataset. Using DuckDB's SQL engine to handle the scale, we compute the average Origin-Destination (OD) matrix for each identified cluster, materializing the results in `gold.typical_od_matrices`.

Visualization and Reporting

To allow transport experts to validate these patterns, the pipeline automatically generates two types of analytical outputs stored in the results bucket:

- **Temporal Profiles (Static):** A line chart visualization (via Matplotlib) plotting the centroids of the identified clusters. This clearly highlights the temporal differences between profiles, such as the steep morning commute peak in workdays versus the smoother midday curve typical of weekends.

- **Interactive OD Heatmaps:** An HTML report generated using Plotly. This tool allows analysts to interactively select a specific cluster and hour (e.g., "Cluster 0 at 08:00 AM") to visualize the spatial distribution of trips. The heatmap uses a logarithmic color scale to handle the high variance in trip counts between major urban centers and rural peripheries.

5.2 Use Case 2: Infrastructure Gap Analysis

The second business objective addresses the identification of zones where transport infrastructure fails to meet potential demand. To solve this, we implemented a **Gravity Model** in the Gold layer, comparing the theoretical interaction potential between municipalities against the actual mobility flows recorded by MITMA.

Implementation Approach

The solution is orchestrated via an Airflow DAG (`32_bq2_gaps`). The logic follows a three-step process:

1. **Theoretical Modeling:** We calculate a "Gravity Score" for every origin-destination pair using the classic impedance formula:

$$T_{ij} = k \cdot \frac{P_i \cdot E_j}{d_{ij}^2} \tag{2}$$

   where P_i is the population at the origin (from `metric_population`), E_j is the economic attractiveness (using `income_per_capita` from `metric_ine_rent` as a proxy), and d_{ij} is the distance from our pre-computed `dim_zone_distances`.

2. **Dynamic Normalization (k-factor):** To make the theoretical score comparable to physical trips, the code dynamically calculates a normalization factor (k) by computing the ratio between the total sum of actual MITMA trips and the total sum of gravity scores for the period.

3. **Gap Detection:** The final metric, `mismatch_ratio`, is derived by dividing actual trips by the normalized potential. A ratio significantly below 1.0 indicates a connection where mobility is lower than demographic and economic factors suggest, highlighting a potential infrastructure deficit.

Spatial Analytics and Visualization

The pipeline includes a spatial filtering mechanism allowing users to run this analysis on specific regions (e.g., passing a WKT Polygon for the "Comunidad de Madrid"). Using DuckDB's spatial extension (`ST_Intersects`), the system aggregates the results to generate two key visualizations using Kepler.gl:

- **Service Level Ranking:** A bubble map where zones are sized by importance ($P \times E$) and colored by their average service level, instantly highlighting underserved economic hubs.
- **Mobility Arcs:** A flow map visualizing specific origin-destination connections that show the highest disparity between potential and actual traffic.

5.3 Use Case 3: Functional Zoning Classification

The third analytical product moves beyond simple traffic counting to characterize the *functional role* of each municipality within the metropolitan network. By analyzing the directionality and retention of mobility flows, this use case automatically classifies zones into semantic categories (e.g., "Bedroom Communities" vs. "Activity Hubs"), aiding urban planners in understanding regional dependencies.

Methodology and Classification Logic

The implementation is encapsulated in the DAG `33_bq3_functional_classification`. The logic aggregates the granular records from `fact_mobility` into three distinct flow types per zone: **Inflow** (trips entering), **Outflow** (trips leaving), and **Internal** (trips starting and ending in the same zone).

Using these aggregates, two key indices are calculated:

1. **Net Flow Ratio:** Measures the balance of attraction.

$$R_{\text{net}} = \frac{\text{Inflow} - \text{Outflow}}{\text{Inflow} + \text{Outflow}}$$

2. **Retention Rate:** Measures the zone's self-sufficiency.

$$R_{\text{retention}} = \frac{\text{Internal}}{\text{Outflow} + \text{Internal}}$$

Based on these metrics, the SQL pipeline applies a hierarchical decision tree to assign a `functional_label` to each municipality:

- **Self-Sustaining Cell:** Zones with high retention ($R_{\text{retention}} > 0.20$), indicating a municipality that satisfies most of its residents' needs locally.
- **Activity Hub (Importer):** Zones with a positive net flow ($R_{\text{net}} > 0$), typically business districts or commercial centers attracting commuters.
- **Bedroom Community (Exporter):** Zones with a negative net flow ($R_{\text{net}} < 0$), characterizing residential areas where the population commutes out for work.
- **Balanced / Transit Zone:** Areas with neutral flow ratios acting as connectors.

Visualization Strategy

The results are materialized in a Kepler.gl dashboard combining two visual layers to provide context:

- **Polygon Layer (Choropleth):** Displays the administrative boundaries colored by their `functional_label`. This reveals regional clusters (e.g., a ring of "Bedroom Communities" surrounding a central "Activity Hub").
- **Centroid Layer:** Displays bubbles sized by `total_activity` (Sum of In + Out + Internal), allowing users to distinguish between major metropolitan centers and smaller rural nodes regardless of their functional classification.

6 Discussion

The implementation of this 3-tier Data Lakehouse represents a significant advancement over traditional file-based workflows for mobility analysis. By transitioning from ad-hoc processing of CSV files to a structured Lakehouse architecture, the system achieves three critical operational goals:

- **Democratization of Big Data:** The architecture successfully decouples the complex data engineering lifecycle (ELT) from the analytical consumption. Transport experts, who previously required specialized programming skills to parse terabytes of daily CSVs, can now access the Gold layer using standard SQL queries or BI tools. The pre-computation of metrics like the *gravity score* or *functional labels* reduces the time-to-insight from days to seconds.

- **Cost-Effective Scalability:** The separation of storage (S3) and compute (DuckDB) allows the infrastructure to handle years of historical data with minimal overhead. Unlike coupled Data Warehouses, where costs accrue 24/7, this architecture only incurs compute costs during active transformation or querying. The use of ephemeral AWS Batch instances demonstrates that high-performance analytics can be achieved on a limited budget.
- **Contextual Enrichment:** The true value of the system lies in the integration achieved in the Silver layer. By enriching raw mobility flows with socio-economic context (INE data) and spatial distances, the system transforms raw trip counts into actionable accessibility metrics. This was demonstrated in Use Case 2, where the combination of population, rent, and distance revealed infrastructure gaps that raw mobility matrices alone could not identify.

7 Conclusions and Limitations

7.1 Conclusions

This project successfully designed and deployed a robust Data Lakehouse capable of ingesting, cleaning, and analyzing Spanish mobility data at scale. The 3-tier architecture ensures data governance, with a raw Bronze layer for auditability, a refined Silver layer for integration, and a business-ready Gold layer.

The implementation of the three use cases demonstrates the system's versatility: from identifying temporal patterns via Unsupervised Learning (Clustering) to solving domain-specific physics problems (Gravity Models) and geospatial classification. By leveraging DuckDB and DuckLake, the solution proves that ACID-compliant, high-performance analytics are achievable using open-source tools without the need for expensive enterprise licenses.

7.2 Limitations

Despite the success of the architecture, several limitations were identified during implementation, concerning both data quality and infrastructure constraints:

- **Source Data Anomalies:** As noted in the MITMA technical documentation, the source datasets contain significant gaps during October and November 2023 due to mobile network incidents. While our pipeline's robust ingestion logic prevents these missing files from crashing the system, the resulting analytics for Q4 2023 require statistical imputation to be fully representative.
- **Infrastructure Constraints (Neon Catalog):** A critical bottleneck was observed regarding the metadata catalog management. We utilized the Neon Serverless Postgres Free Tier as the central catalog for DuckLake. It was observed that this tier enforces strict connection lifecycle limits. During the processing of large historical backfills, the catalog would frequently terminate the connection due to timeout limits on idle transactions, even while the compute engine (DuckDB) was actively writing data to S3. This necessitated the fragmentation of batch loads into smaller temporal windows to ensure transaction commit stability.
- **Vertical Scaling vs. Horizontal Distribution:** A structural limitation of using DuckDB is its single-node architecture. Unlike distributed engines like Apache Spark which scale horizontally (adding more machines to a cluster), DuckDB scales vertically (requiring a larger single machine). While AWS Batch allows us to request high-memory instances (e.g., 64GB or 128GB RAM), there is a theoretical ceiling. If a specific join or aggregation

requires more memory than the largest available single EC2 instance, the query would fail or suffer severe performance degradation due to disk spilling.

- **Latency and Real-Time Analysis:** The current architecture is designed for $T + 1$ (Daily) batch processing. Due to the provisioning time of AWS Batch instances and the file-based ingestion nature of the source, this system is not suitable for real-time traffic monitoring. Use cases requiring sub-minute latency (e.g., immediate accident detection or live congestion management) would require a Streaming architecture (e.g., Kafka + Flink) rather than the current Lakehouse approach.

References

[1] Spanish Ministry of Transport (MITMA). *Open Data Mobility.* Available at: `https://www.transportes.gob.es/ministerio/proyectos-singulares/estudios-de-movilidad-con-big-data/opendata-movilidad` (Accessed on 4 January 2026).

[2] Spanish National Statistics Institute (INE). Available at: `https://www.ine.es/` (Accessed on 4 January 2026).

[3] Centro Nacional de Información Geográfica (CNIG). Available at: `https://www.ign.es/` (Accessed on 4 January 2026).

[4] Ayuntamiento de Madrid. *Portal de Datos Abiertos.* Available at: `https://datos.madrid.es/` (Accessed on 4 January 2026).

[5] DuckDB Documentation. Available at: `https://duckdb.org/` (Accessed on 4 January 2026).

[6] Amazon Web Service Batch. *AWS Batch Documentation.* Available at: `https://aws.amazon.com/es/batch/` (Accessed on 4 January 2026).

[7] DuckLake. *DuckLake Documentation.* Available at: `https://ducklake.select/docs/stable/` (Accessed on 4 January 2026).

[8] Apache Software Foundation. *Apache Airflow Documentation.* Available at: `https://airflow.apache.org/` (Accessed on 4 January 2026).