1. Spatial Modeling – Diffusion Convolution

- Road network = **Directed Graph**. Nodes = sensors; edges = traffic connection strength.
- Instead of using traditional CNN (which works on images), they use **Diffusion Convolution**.
- **Diffusion = Random walk on graph**:
  - Forward walk: downstream traffic
  - Backward walk: upstream traffic
- **Equation:**
  They apply filters over the graph using this diffusion process. It's like applying a convolution kernel over a graph, not a grid.

➡️ Think of this as "information diffusing" across nearby roads, weighted by their connectivity.

2. 🕒 Temporal Modeling – DCGRU

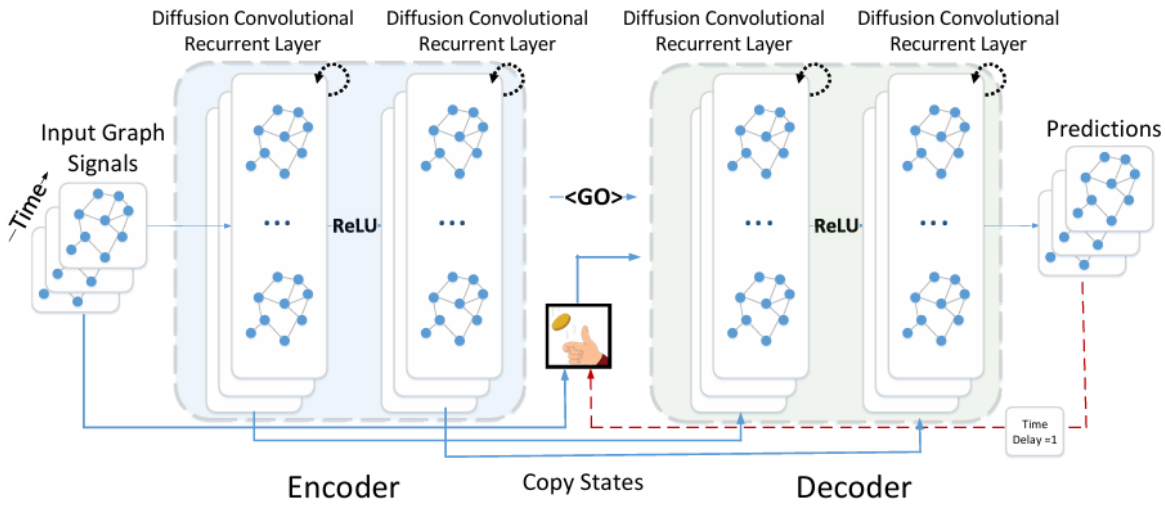
- Use **GRU** (Gated Recurrent Unit), but instead of matrix multiplication, use **Diffusion Convolution**.
- Called it: **DCGRU** (Diffusion Convolutional GRU)

Formally:

- `r(t), u(t)` are reset and update gates like in GRU
- But each multiplication is replaced with diffusion convolution:
  `0r ?G [X(t), H(t−1)]` etc.

3. 🔄 Encoder–Decoder Architecture

- **Encoder**: Reads past traffic data → compresses it to hidden states
- **Decoder**: Uses encoder's last state to predict future steps
- **<GO> Token**: Start of decoder input



4. 🎯 Scheduled Sampling

- During training, sometimes use real previous value (ground truth), sometimes use model's own prediction.
- This gradually shifts the model to depend on its own predictions → useful for **long-term forecasting**.

📊 Summary of DCRNN Pipeline:

```
Past Traffic Data (Graph Signals)
        ↓
Diffusion Convolution + GRU (DCGRU)
        ↓
Encoder compresses info
        ↓
Decoder expands to future predictions
        ↓
Future Traffic Forecast
```

☑ Why It Works Better:

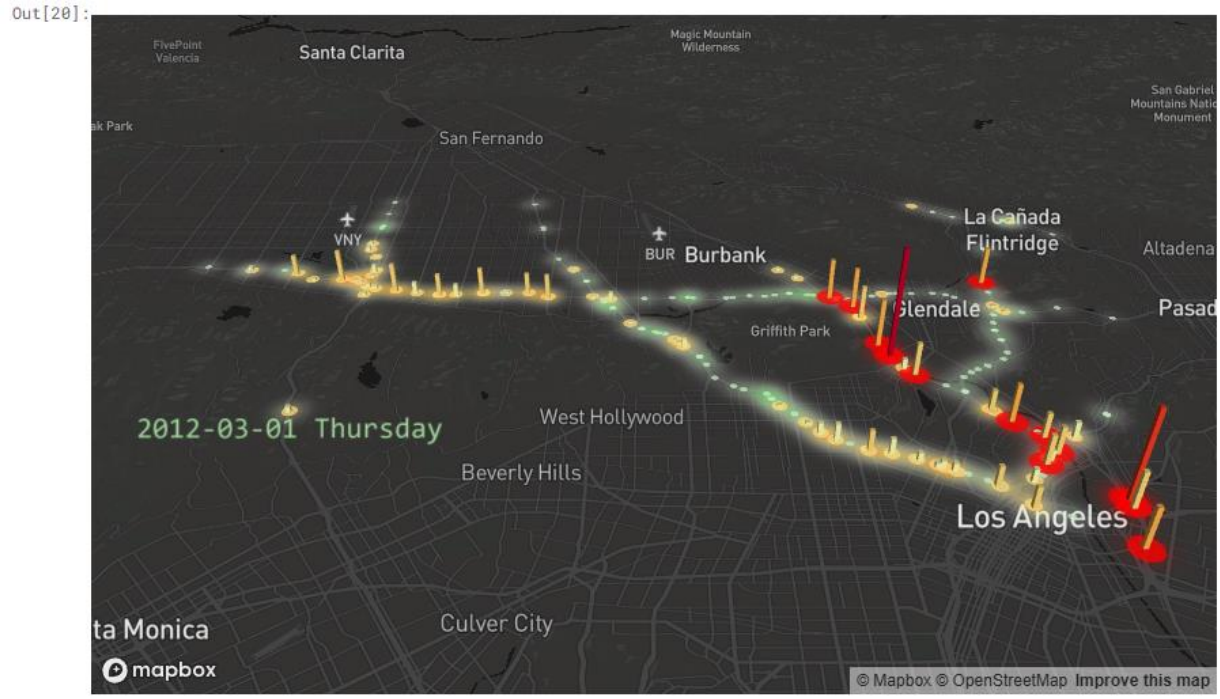
- Models **directional spatial dependencies** (not just closeness, but where traffic flows).
- Handles **nonlinear time patterns** via GRU.
- Learns **both spatial and temporal** patterns together.
- Performs better than CNNs or LSTMs alone on traffic prediction.

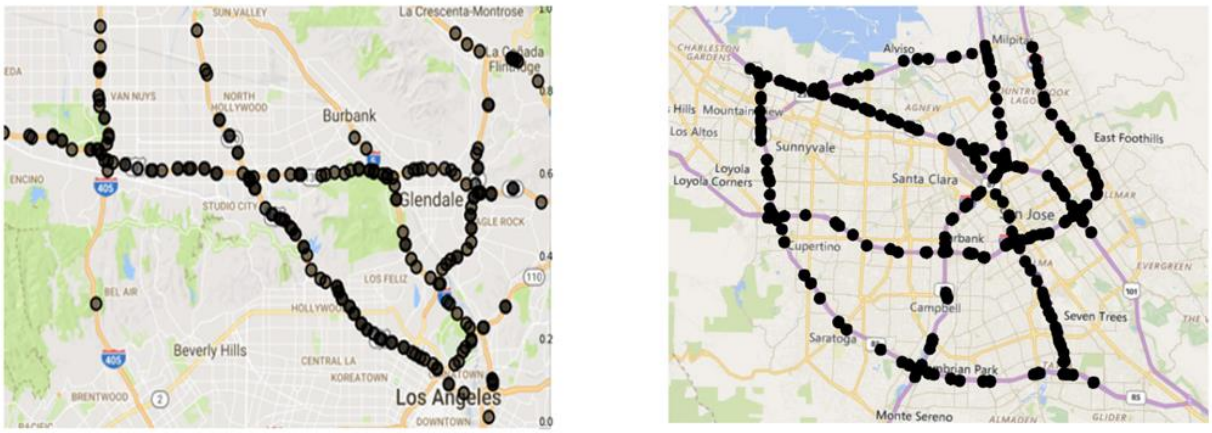
📦 Datasets Used in DCRNN Paper

1. METR-LA (Los Angeles)

- 📍 **Location**: Highways in **Los Angeles County**
- 🧪 **Sensors**: 207 loop detectors
- 🕓 **Time Period**: March 1, 2012 – June 30, 2012 (4 months)
- ⏱️ **Interval**: 5-minute intervals
- 📊 **Data Points**: 6,519,002 readings
- 🧮 **Features**: Traffic speed (velocity)
- 🔁 **Split**:
    - Train: 70%
    - Validation: 10%
    - Test: 20%

https://www.kaggle.com/code/xiaohualu/mapvisualization-metrla-pydeck





(a) METR-LA (b) PEMS-BAY

Figure 8: Sensor distribution of the METR-LA and PEMS-BAY dataset.

2. PEMS-BAY (Bay Area, California)

- 📍 **Location**: Bay Area highways (California)
- 🧪 **Sensors**: 325 sensors (from CalTrans PeMS)

- 🕓 **Time Period**: Jan 1, 2017 – May 31, 2017 (5 months)
- ⏱️ **Interval**: 5-minute intervals
- 📊 **Data Points**: 16,937,179 readings

https://www.kaggle.com/code/yizhongchao/dcrnn-on-pems-bay

🧠 Preprocessing & Graph Construction

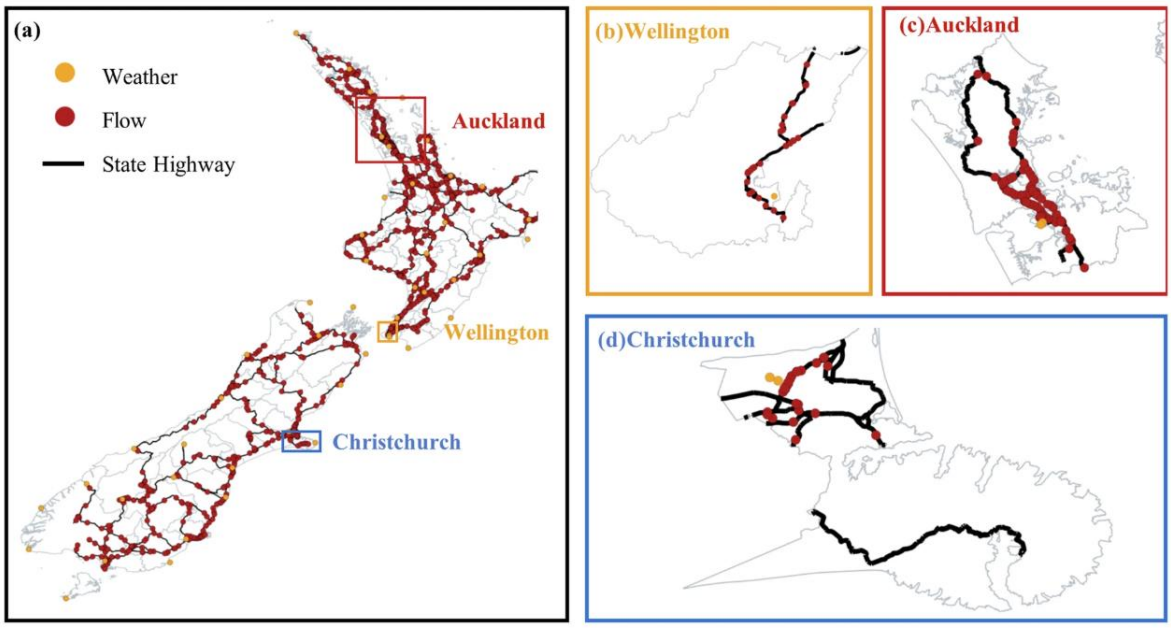
- **Z-Score Normalization**: Applied to all sensor readings.
- **Graph Construction**:
  - Nodes = Sensors
  - Edges = Proximity based on **road network distance**

New Zealand Traffic Dataset Overview

- **Coverage**: Jan 2013–Dec 2021 (9 years), sampled at **15-minute intervals** nature.com.
- **Scale**: **2042 sensors** across highways figshare.com+4springernature.figshare.com+4paperity.org+4.
- **Traffic Types**:
  - Light-duty vehicles (< 5.5 m)
  - Heavy-duty (> 11 m)
  - Medium vehicles split evenly figshare.com+13nature.com+13nature.com+13.
- **Traffic Metrics**: Speed, headway, flow direction inferred via dual-sensor activation x-mol.com+3nature.com+3figshare.com+3.
- **Meta Data**: Comes with geospatial coords, highway IDs, and direction.

  https://github.com/yuruotao/NZ_dataset

From: High-resolution multi-source traffic data in New Zealand



Geospatial distribution of traffic flow sensors and weather sensors. The red and yellow points represent the traffic flow sensors and weather sensors, respectively. The black lines represent highway roads. Three representative cities, including Christchurch, Auckland, and Wellington, are selected for further investigation.

Auxiliary Data Provided

- **Weather**: NOAA-sourced climate metrics (temperature, precipitation, dew point, humidity) from 55 stations nature.com.
- **Extreme Events**: Catalogued floods, fog, hail, tornadoes from NIWA (2013–2021) nature.com.
- **Holidays**: National & regional holiday calendars (e.g. Waitangi Day, Good Friday) .

Format & Structure

- **Storage**: Delivered via **Figshare** and as an **SQLite3 database** with six tables:
  - Sensor locations
  - Traffic records
  - Weather station metadata
  - Weather time-series
  - Extreme weather events
  - Holidays

  nature.com+4nature.com+4springernature.figshare.com+4nature.com+12s

pringernature.figshare.com+12x-mol.com+12nature.com+2nature.com+2nature.com+2

- **Imputation**:
  - Rows with ≥ 30% missing were removed.
  - Remaining missing values filled by linear interpolation (mean of neighbor days) nature.com+4nature.com+4nature.com+4nature.com+2springernature.figshare.com+2nature.com+2.
- **Normalization**: All numeric variables standardized (Z-score).

☑ Is This Rail-Ready for DCRNN?

✅ Requirement	Dataset Ready?
Fixed time interval	15-min, adjustable to 5-min
Multi-sensor network	2,042 nodes – ideal
Road graph available	Sensor coords + highways
Time series aligned	Yes, cleaned & normalized
Metadata for edges	Highway distances + direction info included

🧠 Next Steps for DCRNN

1. **Resample** to 5-minute intervals (if desired).
2. Extract node features: speed, flow, vehicle type counts.
3. Build **Graph Adjacency Matrix**:
   a. Use highway topology + sensor coords.
   b. Apply Gaussian kernel with threshold (σ, κ).
4. Load data into **npz** or Tensor format matching METR-LA structure.
5. Set sensors = 2042 (nodes), horizon = your forecast window (e.g. 12 steps).
6. Train DCRNN via official code using your datasets.
7. Use weather/holiday/extreme-event metadata as **external features** for improved forecasting.

8. Perform baseline tests; refine by feature engineering and hyperparameter tuning.

🛠 Can Assist With:

- Python scripts to read the SQLite DB.
- Constructing adjacency matrix.
- Converting time series to required model input.
- Integrating external events/weather into model training.

- **Adjacency Matrix**:

$$W_{ij} = \exp\left(-\frac{\text{dist}(v_i, v_j)^2}{\sigma^2}\right) \text{ if } \text{dist}(v_i, v_j) \leq \kappa, \text{ else } 0$$

Where:

- `dist(v_i, v_j)` = road distance between sensors
- `σ` = standard deviation of distances
- `κ` = threshold