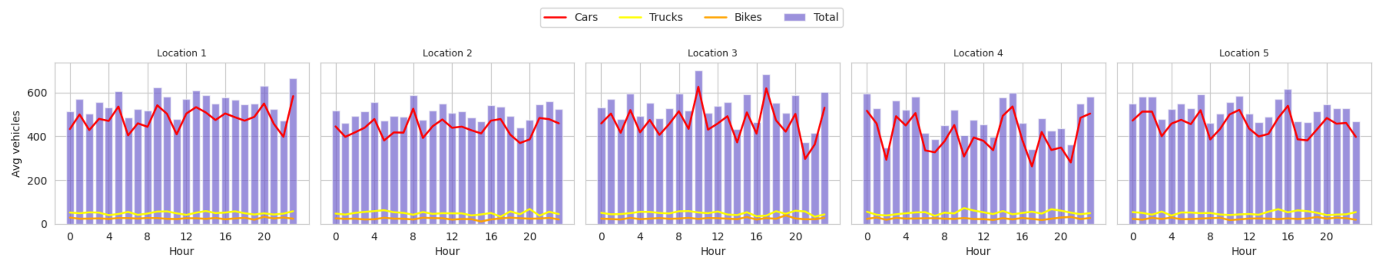
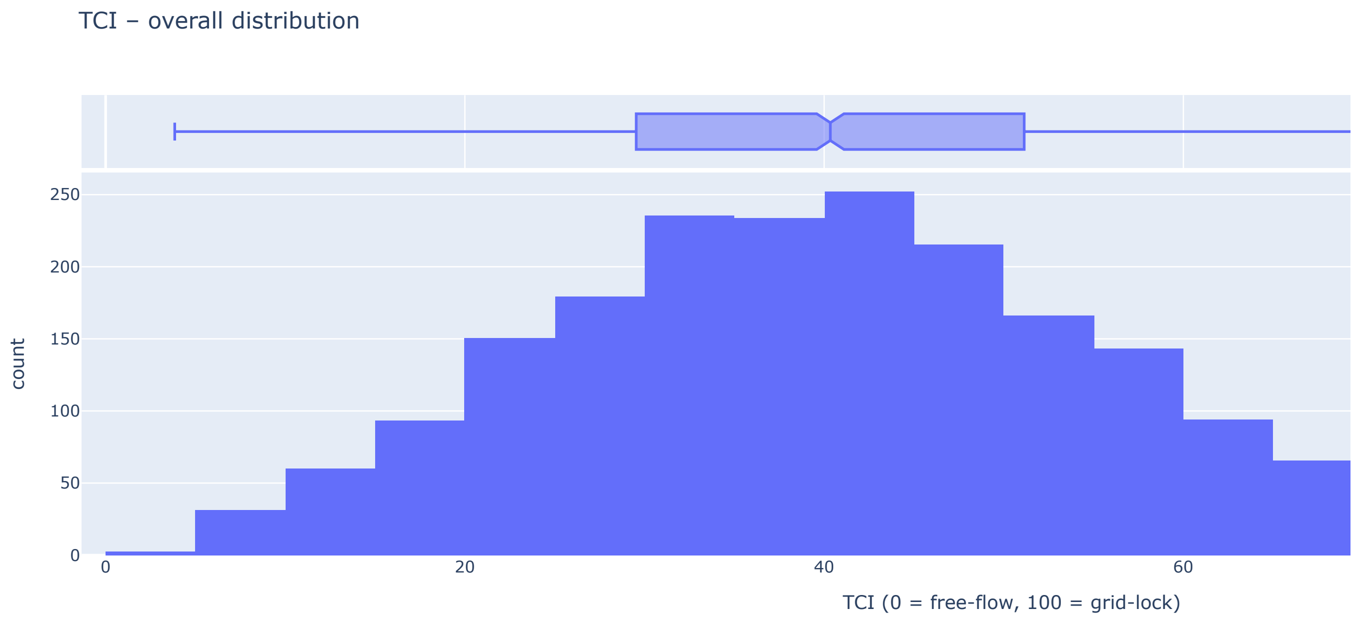
Exploratory Data Analysis - Smart Traffic Management Dataset

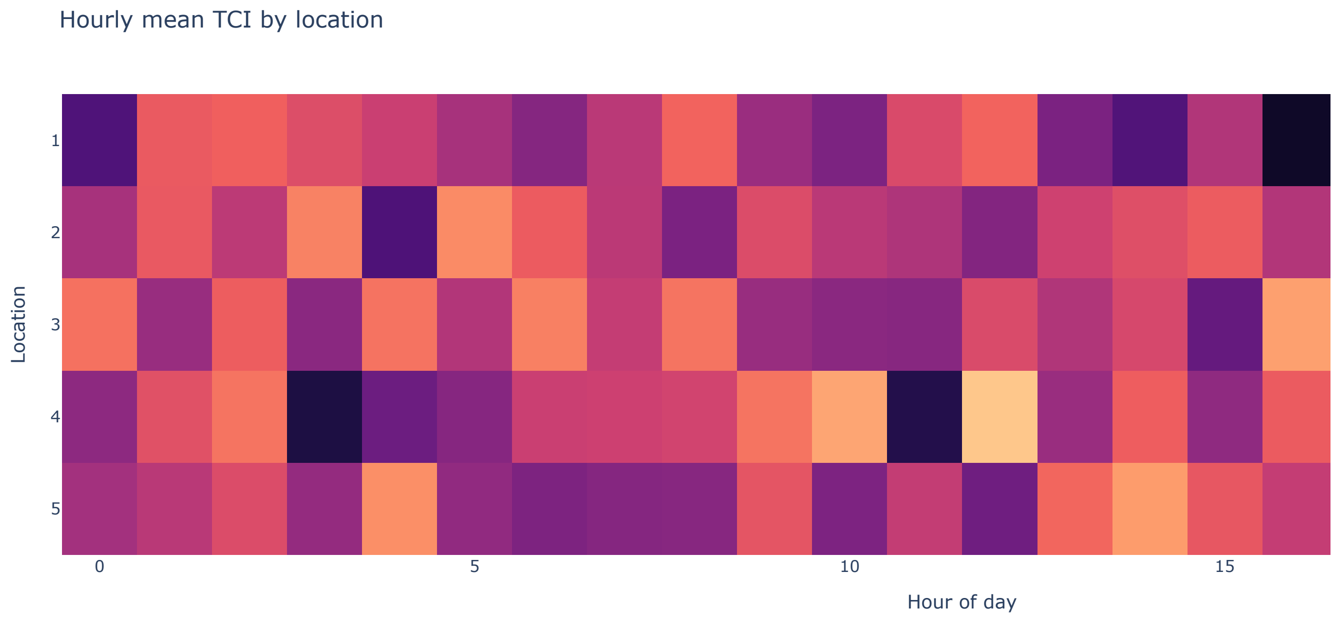
```
# -------------------------------------------------------------------
# Exploratory Data Analysis – (10 figures)
# -------------------------------------------------------------------
import pandas as pd, numpy as np, plotly.express as px, seaborn as sns, matplotlib.pyplot as plt
from pathlib import Path

# 0 ‣ load ..................................................................
df_raw = pd.read_csv(Path("smart_traffic_management_dataset.csv"))
df_raw["timestamp"] = pd.to_datetime(df_raw["timestamp"])
df_raw["hour"]      = df_raw["timestamp"].dt.hour
df_raw["dow"]       = df_raw["timestamp"].dt.day_name().str[:3]   # Mon, Tue …

# 1 ‣ compute Traffic-Congestion Index  ................
FFS, CAP = 65, 1800                          # free-flow speed, capacity
def tci(r):
    dens  = (r.vehicle_count_cars + 1.5*r.vehicle_count_trucks + 0.3*r.vehicle_count_bikes)/CAP
    speed = max(0, 1 - r.avg_vehicle_speed/FFS)
    inc   = 0.25 if r.accident_reported        else 0
    sig   = {"Green":0, "Yellow":.05, "Red":.20}.get(r.signal_status, 0)
    wet   = .15 if r.weather_condition in ("Rainy","Foggy","Snowy") else (.05 if r.weather_condition=="Windy" else 0)
    return 100*(.55*dens + .30*speed + inc + sig + wet)
df_raw["TCI"] = df_raw.apply(tci, axis=1)

veh_cols = ["vehicle_count_cars",
            "vehicle_count_trucks",
            "vehicle_count_bikes"]

df_raw["total_vehicles"] = df_raw[veh_cols].sum(axis=1)

# ─── Hourly average vehicle counts by location ─────────────────────────────
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("whitegrid")

veh_cols = ["vehicle_count_cars",
            "vehicle_count_trucks",
            "vehicle_count_bikes"]

# ensure total_vehicles exists
df_raw["total_vehicles"] = df_raw[veh_cols].sum(axis=1)

# choose the first 5 locations (adjust if you want specific IDs)
loc_ids = sorted(df_raw["location_id"].unique())[:5]

fig, axes = plt.subplots(
    nrows=1, ncols=5, figsize=(18, 3),      # ~3.5 in wide per panel
    sharey=True                             # common y-axis for easy comparison
)

for ax, loc in zip(axes, loc_ids):
    hourly = (df_raw[df_raw["location_id"] == loc]
              .groupby("hour")[veh_cols + ["total_vehicles"]]
              .mean()
              .reset_index())

    # bars – total vehicles
    ax.bar(hourly["hour"], hourly["total_vehicles"],
           color="slateblue", alpha=.65, label="Total")

    # overlay lines – per-class means
    ax.plot(hourly["hour"], hourly["vehicle_count_cars"],
            color="red",       lw=1.8, label="Cars")
    ax.plot(hourly["hour"], hourly["vehicle_count_trucks"],
            color="yellow", lw=1.8, label="Trucks")
    ax.plot(hourly["hour"], hourly["vehicle_count_bikes"],
            color="orange",     lw=1.8, label="Bikes")

    ax.set_title(f"Location {loc}", fontsize=9)
    ax.set_xlabel("Hour")              # keep x-axis
    ax.set_xticks(range(0, 24, 4))     # 0,4,8,…20
    if ax is axes[0]:
        ax.set_ylabel("Avg vehicles")
```

```
# one shared legend centred above all panels
handles, labels = axes[0].get_legend_handles_labels()
fig.legend(handles, labels, ncol=4, loc="upper center", bbox_to_anchor=(0.5, 1.15))
plt.tight_layout()
plt.show()


# ╔══════════════════  T C I  –  5 figures  ════════════════════════════════╗
# A ▸ distribution ----------------------------------------------------------------
px.histogram(df_raw, x="TCI", nbins=40, marginal="box",
             labels=dict(TCI="TCI (0 = free-flow, 100 = grid-lock)"),
             title="TCI – overall distribution").show()

# B ▸ heat-map: hour × location ----------------------------------------------------
hm = (df_raw.groupby(["location_id","hour"]).TCI.mean().reset_index()
                .pivot(index="location_id", columns="hour", values="TCI"))
px.imshow(hm, aspect="auto", color_continuous_scale="magma_r",
          labels=dict(x="Hour of day", y="Location", color="Mean TCI"),
          title="Hourly mean TCI by location").show()

# C ▸ violin: TCI by weekday -------------------------------------------------------
px.violin(df_raw, x="dow", y="TCI", box=True, points=False,
          category_orders={"dow":["Day 1","Day 2"]},
          title="TCI Daily Distribution",
          labels=dict(dow="Day", TCI="TCI")).show()

wx_colors = dict(Sunny="gold",
                 Cloudy="lightslategray",
                 Windy="mediumslateblue",
                 Rainy="steelblue",
                 Foggy="thistle")

# D ▸ box: TCI by weather
fig_tci_wx = px.box(
    df_raw, x="weather_condition", y="TCI",
    points="outliers",                          # keep the dots
    color="weather_condition",                  # colour *by* weather
    color_discrete_map=wx_colors,
    labels=dict(weather_condition="Weather", TCI="TCI"),
    title="TCI vs. weather condition"
)
fig_tci_wx.update_layout(showlegend=False)
fig_tci_wx.show()

# E ▸ scatter: TCI vs. average speed --------------------------------------------
scatter_df = df_raw.sample(
    n=min(4000, len(df_raw)),         # cap sample size
    random_state=1
)
figE = px.scatter(
    scatter_df,
    x="avg_vehicle_speed", y="TCI",
    color="signal_status",
    trendline="lowess", height=450,
    labels=dict(avg_vehicle_speed="Speed (km/h)", TCI="TCI", signal_status="Signal"),
    title="Inverse relationship between speed and congestion"
)
figE.show()


# ╔═════════════════  N O N - T C I  – 5 figures  ══════════════════════════╗
# F ▸ hourly speed profile ---------------------------------------------------------
speed_h = (df_raw.groupby("hour").avg_vehicle_speed.mean()
                 .reset_index())
px.line(speed_h, x="hour", y="avg_vehicle_speed", markers=True,
        labels=dict(avg_vehicle_speed="Speed (km/h)", hour="Hour"),
        title="Average vehicle speed – diurnal pattern").show()

# G ▸ bar: mean speed vs. traffic-light status  (colour-matched)
sig_speed = (
    df_raw.groupby("signal_status")["avg_vehicle_speed"]
          .mean()
          .reset_index()
          .sort_values("avg_vehicle_speed")
)

colour_map = {"Red":"red", "Yellow":"gold", "Green":"limegreen"}
```

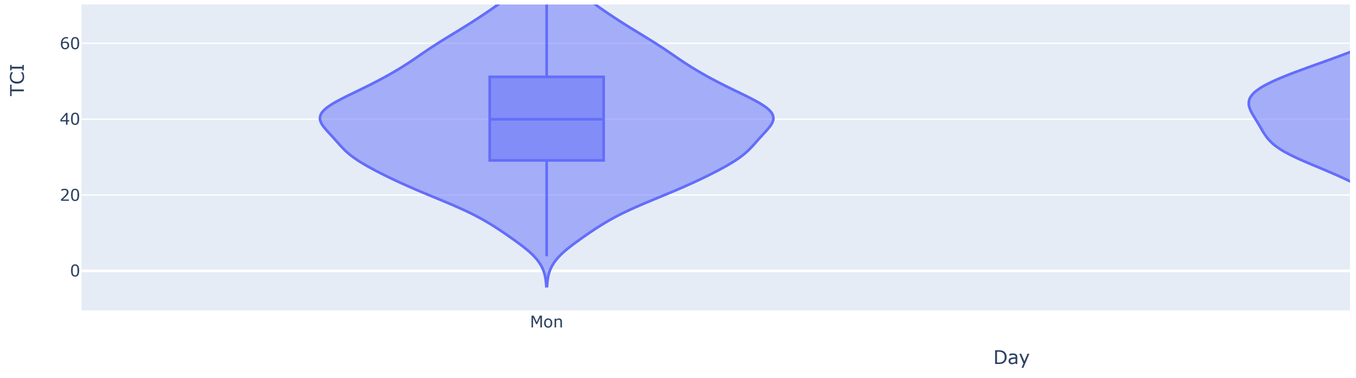
```
    colour_map = {"Red":"red", "Yellow":"gold", "Green":"limegreen"}

    fig_speed = px.bar(
        sig_speed, x="signal_status", y="avg_vehicle_speed",
        title="Mean vehicle speed by traffic-light status",
        labels=dict(signal_status="Signal colour",
                    avg_vehicle_speed="Mean speed (km/h)"),
        color="signal_status", color_discrete_map=colour_map
    )
    # keep bars easy to read
    fig_speed.update_yaxes(range=[0, sig_speed["avg_vehicle_speed"].max()*1.10])
    fig_speed.update_layout(showlegend=False)
    fig_speed.show()


    # H ▸ bar: weather frequency (colour-coded)
    weather_ct = (
        df_raw["weather_condition"].value_counts()
              .rename_axis("weather_condition")
              .reset_index(name="count")
    )

    wx_colors = dict(Sunny="gold",
                     Cloudy="lightslategray",
                     Windy="mediumslateblue",
                     Rainy="steelblue",
                     Foggy="thistle")

    fig_weather = px.bar(
        weather_ct, x="weather_condition", y="count",
        text="count", title="Weather condition frequency",
        labels=dict(weather_condition="Weather", count="Observations"),
        color="weather_condition", color_discrete_map=wx_colors
    )
    fig_weather.update_layout(showlegend=False, yaxis_title="Observations")
    fig_weather.show()


    # J ▸ correlation heat-map of numeric features ------------------------------------
    core = ["avg_vehicle_speed","vehicle_count_cars","vehicle_count_trucks",
            "vehicle_count_bikes","temperature","humidity"]
    corr = df_raw[core].corr(method="spearman").round(2)
    plt.figure(figsize=(7,6))
    sns.heatmap(corr, annot=True, cmap="coolwarm", vmin=-1, vmax=1, fmt=".2f")
    plt.title("Spearman correlations among core raw numerics")
    plt.tight_layout()
    plt.show()
```


Cars
Trucks
Bikes
Total
Location 1
Location 2
Location 3
Location 4
Location 5
Avg vehicles
Hour

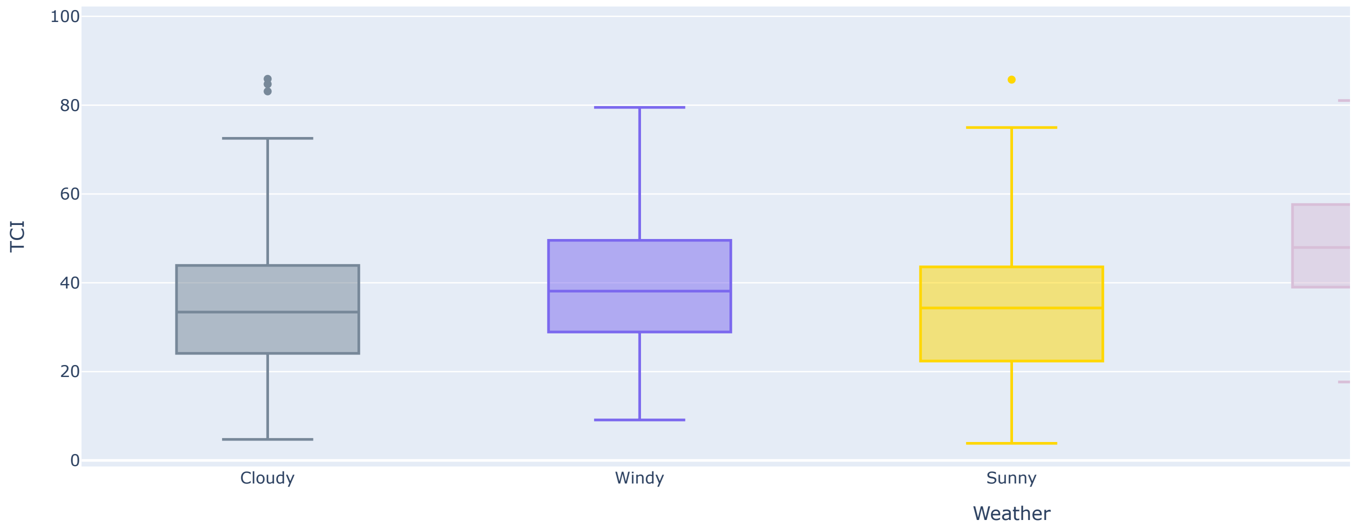

TCI – overall distribution
count
TCI (0 = free-flow, 100 = grid-lock)


Hourly mean TCI by location
Location
Hour of day

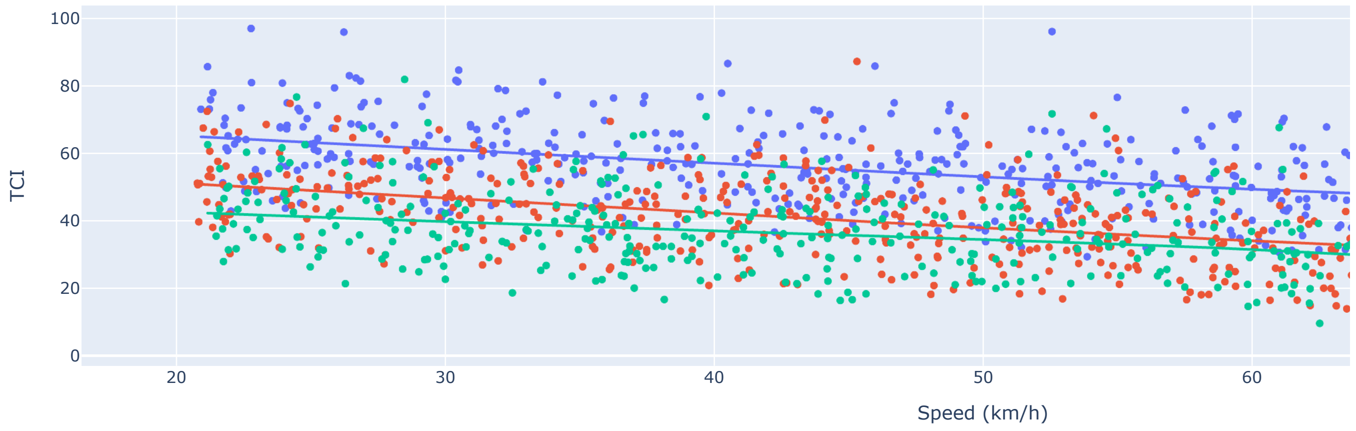
TCI Daily Distribution



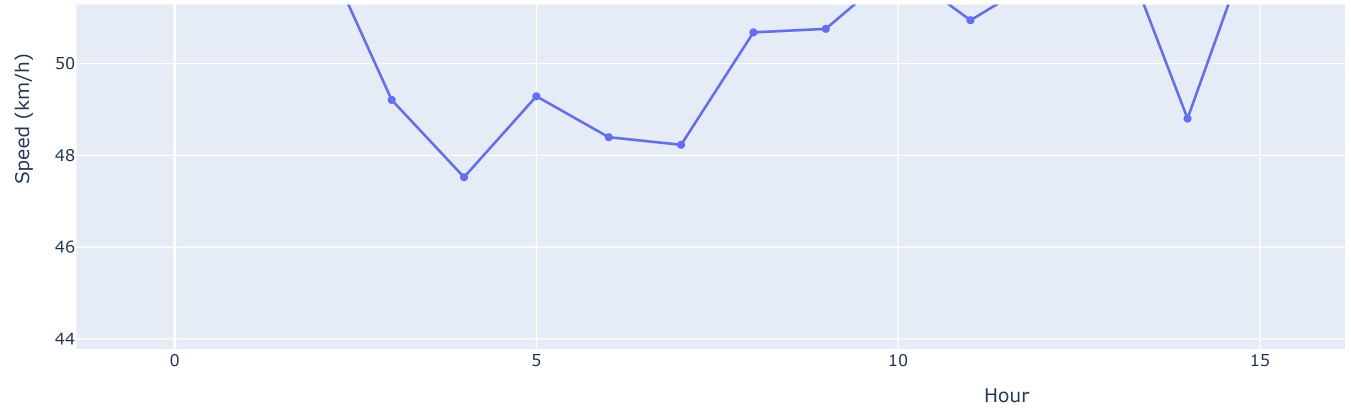
TCI vs. weather condition



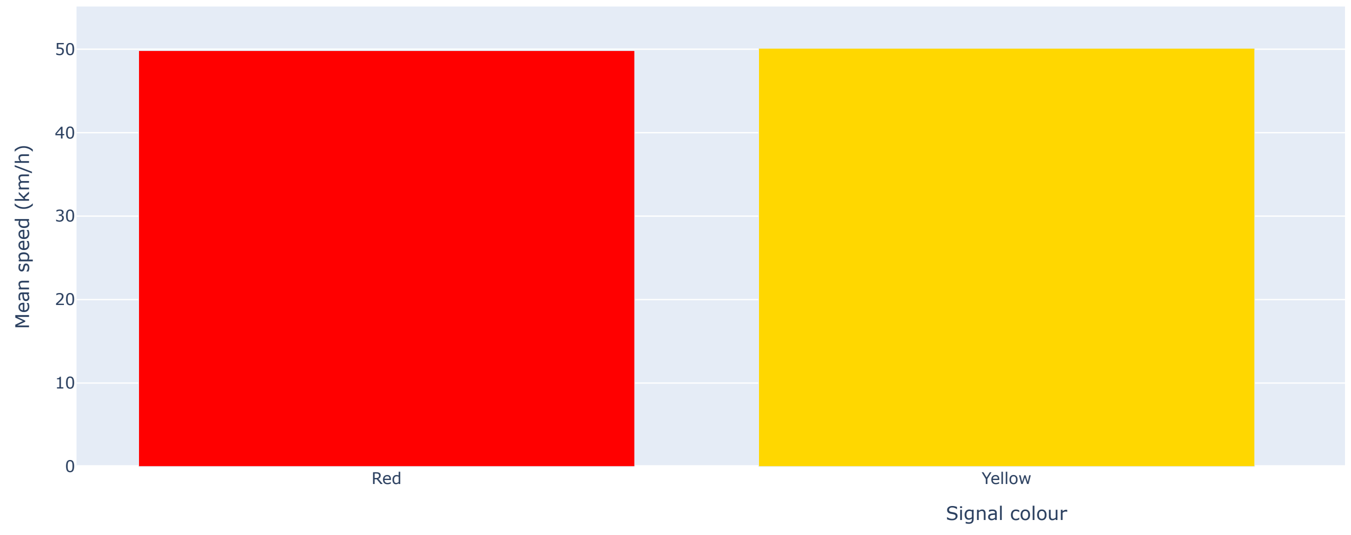
Inverse relationship between speed and congestion



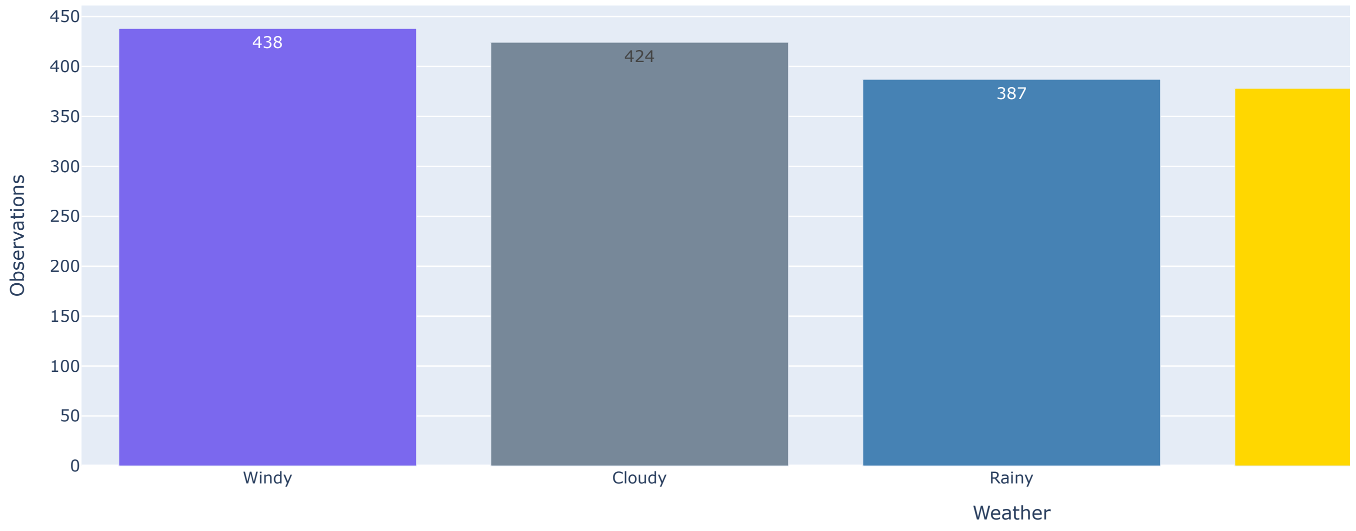
Average vehicle speed – diurnal pattern



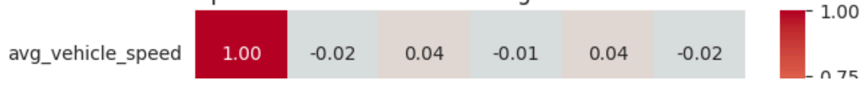
Mean vehicle speed by traffic-light status

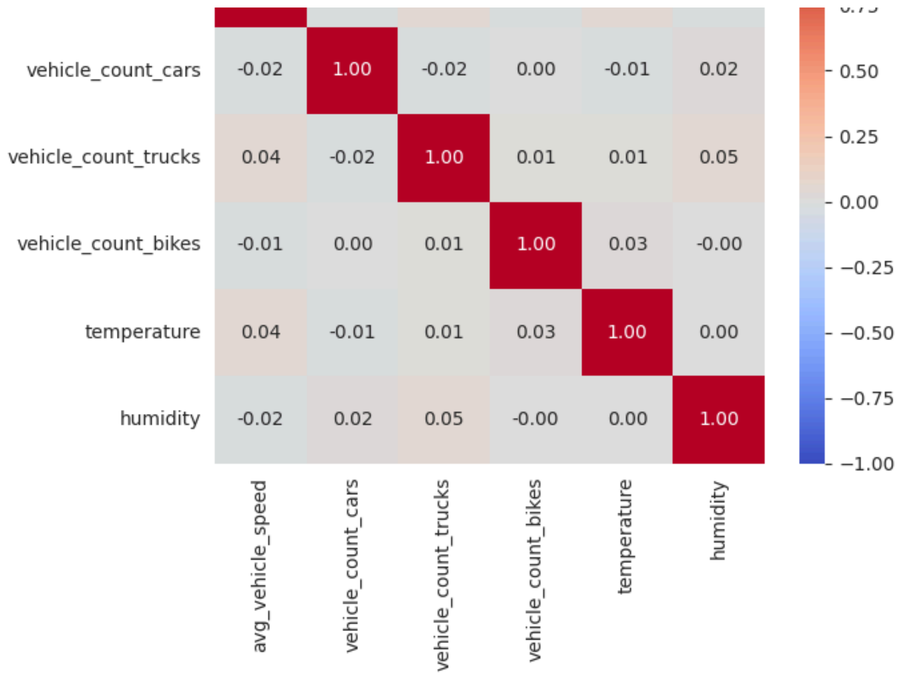


Weather condition frequency



Spearman correlations among core raw numerics




vehicle_count_cars
vehicle_count_trucks
vehicle_count_bikes
temperature
humidity
-0.02 1.00 -0.02 0.00 -0.01 0.02
0.04 -0.02 1.00 0.01 0.01 0.05
-0.01 0.00 0.01 1.00 0.03 -0.00
0.04 -0.01 0.01 0.03 1.00 0.00
-0.02 0.02 0.05 -0.00 0.00 1.00
avg_vehicle_speed
vehicle_count_cars
vehicle_count_trucks
vehicle_count_bikes
temperature
humidity
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

Baseline Model Comparison

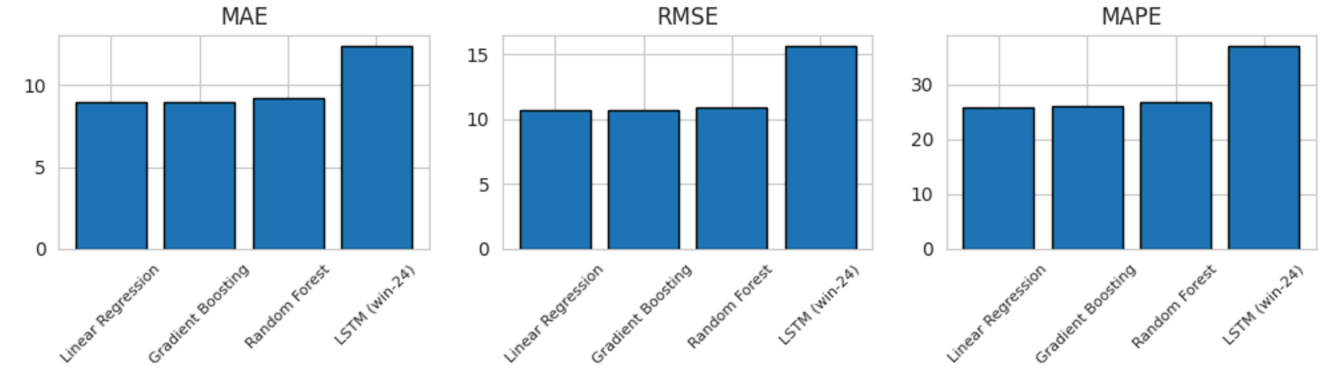
```
# -----------------------------------------------------------
# Benchmark Models for Comparison
# -----------------------------------------------------------

import os, warnings, pandas as pd, numpy as np, matplotlib.pyplot as plt
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model      import LinearRegression
from sklearn.ensemble          import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics           import (
    mean_absolute_error, mean_squared_error, mean_absolute_percentage_error
)
warnings.filterwarnings("ignore", category=FutureWarning)

# ──────────────────────────────────────────────────────────────
# 1. LOAD  +  FEATURE ENGINEERING (incl. lags, rollings, new TCI target)
# ──────────────────────────────────────────────────────────────
df = pd.read_csv("smart_traffic_management_dataset.csv")
df["timestamp"] = pd.to_datetime(df["timestamp"])

# ---------- basic numeric & categorical ----------
num_cols = [
    "avg_vehicle_speed", "temperature", "humidity",
    "vehicle_count_cars", "vehicle_count_trucks",
    "vehicle_count_bikes", "accident_reported",
]
cat_df = pd.get_dummies(df[["weather_condition", "signal_status"]],
                        drop_first=True)
# ---------- cyclical hour ----------
df["hour"]     = df["timestamp"].dt.hour
df["hour_sin"] = np.sin(2 * np.pi * df["hour"] / 24)
df["hour_cos"] = np.cos(2 * np.pi * df["hour"] / 24)


# ---------- Compute TCI target (Traffic-Congestion Index) ----------
FFS       = 65      # km/h free-flow speed (median of FWHA report)
capacity  = 1800    # veh/h/lane    (median of FWHA report)

def compute_tci(row):
    demand_eq = (row["vehicle_count_cars"]
                 + 1.5 * row["vehicle_count_trucks"]
                 + 0.3 * row["vehicle_count_bikes"])
    dens      = demand_eq / capacity                        # 0 … >1
    speed_pen = max(0, 1 - row["avg_vehicle_speed"] / FFS)  # 0 … 1

    inc_pen = 0.25 if row["accident_reported"] else 0
    sig_pen = 0.20 if row.get("signal_status_Red", 0)   else \
              0.05 if row.get("signal_status_Yellow", 0) else 0
    wet_pen = 0.15 if any(row.get(f"weather_condition_{w}", 0)
                          for w in ("Rainy", "Foggy", "Snowy")) else \
              0.05 if row.get("weather_condition_Windy", 0) else 0

    raw = 0.55 * dens + 0.30 * speed_pen + inc_pen + sig_pen + wet_pen
    return 100 * raw          # 0 (free) … 100 (grid-lock)

# add one-hot columns so compute_tci can “see” them
df = pd.concat([df, cat_df], axis=1)
df["traffic_congestion"] = df.apply(compute_tci, axis=1)

# ---------- lags & rolling stats **after** TCI exists ----------
lags      = [1, 3, 6, 12, 24]          # hours
roll_win  = [3, 6, 24]                 # hours

for lag in lags:
    df[f"cong_lag_{lag}"] = df["traffic_congestion"].shift(lag)

for w in roll_win:
    df[f"cong_roll_mean_{w}"] = (df["traffic_congestion"]
                                 .shift(1).rolling(w).mean())
    df[f"cong_roll_std_{w}"]  = (df["traffic_congestion"]
                                 .shift(1).rolling(w).std())
```

```
# drop rows with any NaNs produced by lags/rollings
df = df.dropna().reset_index(drop=True)

# ---------- build final predictor DataFrame ----------
lag_cols   = [f"cong_lag_{l}" for l in lags]
roll_cols  = [c for w in roll_win
                for c in (f"cong_roll_mean_{w}", f"cong_roll_std_{w}")]
time_cols  = ["hour_sin", "hour_cos"]

feat_df = pd.concat([
    df[["location_id"]],
    df[num_cols],
    df[lag_cols],
    df[roll_cols],
    cat_df.loc[df.index],      # same filtered indices
    df[time_cols]
], axis=1)

# ---------- scale predictors; leave raw target for baselines ----------
x_scaler   = StandardScaler()
feat_scaled = pd.DataFrame(
    x_scaler.fit_transform(feat_df),
    columns=feat_df.columns
)

# target scaler (for LSTM only) – fit on congestion values
y_scaler   = StandardScaler()
y_scaled   = y_scaler.fit_transform(
    df["traffic_congestion"].values.reshape(-1, 1)
).ravel()

# predictors for baselines; append raw congestion as target
scaled = feat_scaled.copy()
scaled["traffic_congestion"] = df["traffic_congestion"].values

# ──────────────────────────────────────────────────────────────
# 2. TRADITIONAL BASELINES
# ──────────────────────────────────────────────────────────────
X_tab = scaled.drop(columns="traffic_congestion")
y_tab = scaled["traffic_congestion"]

X_train, X_test, y_train, y_test = train_test_split(
    X_tab, y_tab, test_size=0.2, random_state=42, shuffle=False    # **time-aware split**
)

models  = {
    "Linear Regression": LinearRegression(),
    "Random Forest":     RandomForestRegressor(random_state=42),
    "Gradient Boosting": GradientBoostingRegressor(random_state=42),
}
results = []

for name, mdl in models.items():
    mdl.fit(X_train, y_train)
    pred = mdl.predict(X_test)
    results.append(dict(
        Model = name,
        MAE   = mean_absolute_error(y_test, pred),
        RMSE  = sqrt(mean_squared_error(y_test, pred)),
        MAPE  = mean_absolute_percentage_error(y_test, pred)*100
    ))

# ──────────────────────────────────────────────────────────────
# 3. SEQUENCE WINDOWS  (24-step window)
# ──────────────────────────────────────────────────────────────
def make_windows(df_seq: pd.DataFrame, target_col: str,
                 window:int = 24, horizon:int = 1):
    X, y = [], []
    for i in range(len(df_seq) - window - horizon + 1):
        X.append(df_seq.iloc[i:i+window].values)
        y.append(df_seq[target_col].iloc[i+window + horizon - 1])
    return np.stack(X), np.array(y)

# ── build a seq_df that uses the *scaled* features + *scaled* target ──
seq_df = (
    feat_scaled
```

```
        .assign(
            traffic_congestion=y_scaled,     # the scaled-target series
            timestamp      = df["timestamp"]   # for correct sorting
        )
        .sort_values("timestamp")
        .drop(columns="timestamp")
    )

    # sliding-window on the scaled target
    X_seq, y_seq_scaled = make_windows(
        seq_df, "traffic_congestion",
        window=24, horizon=1
    )

    split_idx         = int(0.8 * len(X_seq))
    X_seq_tr, X_seq_te = X_seq[:split_idx],   X_seq[split_idx:]
    y_seq_tr_scaled, y_seq_te_scaled = (
        y_seq_scaled[:split_idx],
        y_seq_scaled[split_idx:]
    )


    # ──────────────────────────────────────────────────────────────
    # 4. LSTM
    # ──────────────────────────────────────────────────────────────
    import tensorflow as tf
    tf.get_logger().setLevel("ERROR")

    from tensorflow.keras.layers import Input, LSTM, Dropout, Dense, Add, GlobalAveragePooling1D, MultiHeadAttention
    from tensorflow.keras.models import Model

    def build_lstm(num_feats: int) -> Model:
        inp = Input(shape=(None, num_feats))

        # 1) stacked LSTMs (keep return_sequences=True for attention)
        x = LSTM(128, return_sequences=True)(inp)
        x = Dropout(0.2)(x)
        x = LSTM(64, return_sequences=True)(x)

        # 2) self-attention over the time dimension
        #    query, key, value all = the LSTM outputs
        attn_out = MultiHeadAttention(num_heads=4, key_dim=64)(x, x)
        # residual connection
        x = Add()([x, attn_out])

        # 3) pool down to a vector
        x = GlobalAveragePooling1D()(x)

        # 4) final dense heads
        x = Dense(32, activation="relu")(x)
        out = Dense(1, name="congestion")(x)

        return Model(inputs=inp, outputs=out)

    # compile with a fixed LR (Option A)
    lstm = build_lstm(X_seq_tr.shape[-1])
    lstm.compile(
        optimizer = tf.keras.optimizers.Adam(learning_rate=3e-4),
        loss      = "mse"
    )

    early_stop = tf.keras.callbacks.EarlyStopping(
        patience=5, restore_best_weights=True, monitor="val_loss"
    )
    reduce_lr   = tf.keras.callbacks.ReduceLROnPlateau(
        patience=3, factor=0.5, verbose=0
    )

    history = lstm.fit(
        X_seq_tr, y_seq_tr_scaled,
        validation_split=0.1,
        epochs=50,
        batch_size=128,
        callbacks=[early_stop, reduce_lr],
        verbose=0
    )
```

```
# 1) get back the scaled-predictions
y_pred_scaled = lstm.predict(X_seq_te, verbose=0).squeeze()

# 2) invert the target-scaling so we're back in "cars/hour" units
y_pred = y_scaler.inverse_transform(
    y_pred_scaled.reshape(-1,1)
).ravel()
y_true = y_scaler.inverse_transform(
    y_seq_te_scaled.reshape(-1,1)
).ravel()

# 3) compute metrics on the real-unit values
results.append(dict(
    Model = "LSTM (win-24)",
    MAE   = mean_absolute_error(y_true, y_pred),
    RMSE  = sqrt(mean_squared_error(y_true, y_pred)),
    MAPE  = mean_absolute_percentage_error(y_true, y_pred)*100
))


# ────────────────────────────────────────────────────────────
# 5. RESULT TABLE  +  MINI-BARPLOTS
# ────────────────────────────────────────────────────────────
res_df = (
    pd.DataFrame(results)
    .round(3)
    .sort_values("MAPE")
    .reset_index(drop=True)
)
print("\nBaseline + improved LSTM comparison:\n")
print(res_df.to_markdown(index=False))

metrics = ["MAE", "RMSE", "MAPE"]
fig, axs = plt.subplots(1, 3, figsize=(10, 3), sharex=True)
for ax, m in zip(axs, metrics):
    ax.bar(res_df["Model"], res_df[m], edgecolor="black")
    ax.set_title(m); ax.tick_params(axis="x", rotation=45, labelsize=8)
plt.tight_layout(); plt.show()
```

```
Baseline + improved LSTM comparison:

| Model             |    MAE |   RMSE |   MAPE |
|:------------------|-------:|-------:|-------:|
| Linear Regression |  8.957 | 10.705 | 25.762 |
| Gradient Boosting |  8.962 | 10.746 | 25.986 |
| Random Forest     |  9.177 | 10.952 | 26.675 |
| LSTM (win-24)     | 12.38  | 15.65  | 37.049 |
```



Updated Feature Set

```
feat_df.to_csv('feat_df_with_lags.csv', index=False)
feat_df.head(10)
```

	location_id	avg_vehicle_speed	temperature	humidity	vehicle_count_cars	vehicle_count_trucks	vehicle_count_bikes	accident_report
0	1	52.628344	15.056930	71.753301	272	42	11	
1	3	25.992009	16.756222	45.155706	336	88	13	
2	3	61.282667	22.616026	87.462246	434	33	43	
3	2	34.513333	34.389237	31.809682	114	29	9	
4	4	74.664122	25.406762	87.485231	804	70	19	
5	4	70.295922	33.403703	43.005832	43	69	14	
6	3	62.562978	33.984496	32.929323	647	54	30	
7	4	76.930757	15.202750	47.548442	682	66	6	
8	4	21.885309	17.779840	77.176938	869	12	40	
9	1	64.172026	29.249807	75.146520	69	2	9	

10 rows × 27 columns

Setup environment

Compute TCI & Split Data by Site

```
import pandas as pd
import numpy as np
!pip install scikit-learn
import os, shutil

# 1) Read the full dataset & Recompute Target
df = pd.read_csv("feat_df_with_lags.csv")

FFS, CAP = 65, 1800                       # free-flow speed, lane capacity

def compute_tci(row):
    demand_eq = (
        row["vehicle_count_cars"]
        + 1.5 * row["vehicle_count_trucks"]
        + 0.3 * row["vehicle_count_bikes"]
    )
    dens      = demand_eq / CAP
    speed_pen = max(0, 1 - row["avg_vehicle_speed"] / FFS)

    inc_pen = 0.25 if row["accident_reported"] else 0
    sig_pen = 0.20 if row.get("signal_status_Red", 0) else \
              0.05 if row.get("signal_status_Yellow", 0) else 0
    wet_pen = 0.15 if any(
                 row.get(f"weather_condition_{w}", 0)
                 for w in ("Rainy", "Foggy", "Snowy")
             ) else 0.05 if row.get("weather_condition_Windy", 0) else 0

    return 100 * (0.55*dens + 0.30*speed_pen + inc_pen + sig_pen + wet_pen)

df["traffic_congestion"] = df.apply(compute_tci, axis=1)

# 2) Check for output folder
out_dir = "src/data_sites"
os.makedirs(out_dir, exist_ok=True)
#shutil.move("traffic_model.py", os.path.join("src", "traffic_model.py"))

# 3) Find all unique location IDs
location_ids = sorted(df["location_id"].unique())

# 4) For each location_id, filter and write a CSV
for loc in location_ids:
    df_loc = df[df["location_id"] == loc]
    filename = f"{out_dir}/site-{loc}.csv"
    df_loc.to_csv(filename, index=False)
    print(f"Wrote {len(df_loc)} rows to {filename}")

# 5) Quick sanity check
```

```
print("Files now in", out_dir, ":", os.listdir(out_dir))
```

```
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.12/dist-packages (1.6.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.16.1)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (1.5.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn) (3.6.0)
Wrote 413 rows to src/data_sites/site-1.csv
Wrote 397 rows to src/data_sites/site-2.csv
Wrote 374 rows to src/data_sites/site-3.csv
Wrote 390 rows to src/data_sites/site-4.csv
Wrote 402 rows to src/data_sites/site-5.csv
Files now in src/data_sites : ['site-5.csv', 'site-1.csv', 'site-4.csv', 'site-3.csv', 'site-2.csv']
```

Install nvflare and dependencies:

```
! pip install --ignore-installed blinker
#!pip install --upgrade --force-reinstall numpy
#!pip install --upgrade --force-reinstall pandas
#!pip install scikit-learn
! pip install nvflare~=2.5.0rc torch torchvision tensorboard
```

```
Collecting blinker
  Downloading blinker-1.9.0-py3-none-any.whl.metadata (1.6 kB)
Downloading blinker-1.9.0-py3-none-any.whl (8.5 kB)
Installing collected packages: blinker
Successfully installed blinker-1.9.0
Collecting nvflare~=2.5.0rc
  Downloading nvflare-2.5.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: torch in /usr/local/lib/python3.12/dist-packages (2.8.0+cu126)
Requirement already satisfied: torchvision in /usr/local/lib/python3.12/dist-packages (0.23.0+cu126)
Requirement already satisfied: tensorboard in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: cryptography>=36.0.0 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (43.0.3)
Collecting Flask==3.0.2 (from nvflare~=2.5.0rc)
  Downloading flask-3.0.2-py3-none-any.whl.metadata (3.6 kB)
Collecting Werkzeug==3.0.3 (from nvflare~=2.5.0rc)
  Downloading werkzeug-3.0.3-py3-none-any.whl.metadata (3.7 kB)
Collecting Flask-JWT-Extended==4.6.0 (from nvflare~=2.5.0rc)
  Downloading Flask_JWT_Extended-4.6.0-py2.py3-none-any.whl.metadata (3.9 kB)
Collecting Flask-SQLAlchemy==3.1.1 (from nvflare~=2.5.0rc)
  Downloading flask_sqlalchemy-3.1.1-py3-none-any.whl.metadata (3.4 kB)
Collecting SQLAlchemy==2.0.16 (from nvflare~=2.5.0rc)
  Downloading SQLAlchemy-2.0.16-py3-none-any.whl.metadata (9.4 kB)
Requirement already satisfied: grpcio>=1.62.1 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (1.74.0)
Collecting gunicorn>=22.0.0 (from nvflare~=2.5.0rc)
  Downloading gunicorn-23.0.0-py3-none-any.whl.metadata (4.4 kB)
Collecting numpy<2.0.0 (from nvflare~=2.5.0rc)
  Downloading numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 3.8 MB/s eta 0:00:00
Requirement already satisfied: protobuf>=4.24.4 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (5.29.5)
Requirement already satisfied: psutil>=5.9.1 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (5.9.5)
Requirement already satisfied: PyYAML>=6.0 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (6.0.2)
Requirement already satisfied: requests>=2.28.0 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (2.32.4)
Requirement already satisfied: six>=1.15.0 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (1.17.0)
Requirement already satisfied: msgpack>=1.0.3 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (1.1.1)
Collecting docker>=6.0 (from nvflare~=2.5.0rc)
  Downloading docker-7.1.0-py3-none-any.whl.metadata (3.8 kB)
Requirement already satisfied: websockets>=10.4 in /usr/local/lib/python3.12/dist-packages (from nvflare~=2.5.0rc) (15.0.1)
Collecting pyhocon (from nvflare~=2.5.0rc)
  Downloading pyhocon-0.3.61-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.12/dist-packages (from Flask==3.0.2->nvflare~=2.5.0rc) (3.1.6)
Requirement already satisfied: itsdangerous>=2.1.2 in /usr/local/lib/python3.12/dist-packages (from Flask==3.0.2->nvflare~=2.5.0rc) (2
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.12/dist-packages (from Flask==3.0.2->nvflare~=2.5.0rc) (8.2.1)
Requirement already satisfied: blinker>=1.6.2 in /usr/local/lib/python3.12/dist-packages (from Flask==3.0.2->nvflare~=2.5.0rc) (1.9.0)
Requirement already satisfied: PyJWT<3.0,>=2.0 in /usr/local/lib/python3.12/dist-packages (from Flask-JWT-Extended==4.6.0->nvflare~=2.
Requirement already satisfied: typing-extensions>=4.2.0 in /usr/local/lib/python3.12/dist-packages (from SQLAlchemy==2.0.16->nvflare~=
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.12/dist-packages (from SQLAlchemy==2.0.16->nvflare~=2.5.0rc)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.12/dist-packages (from Werkzeug==3.0.3->nvflare~=2.5.0rc) (
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.19.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from torchvision) (11.3.0)
Requirement already satisfied: absl-py>=0.4 in /usr/local/lib/python3.12/dist-packages (from tensorboard) (1.4.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard) (3.8.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorboard) (25.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorboard) (0.
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography>=36.0.0->nvflare~=2.5.0rc) (1.
Requirement already satisfied: urllib3>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from docker>=6.0->nvflare~=2.5.0rc) (2.5.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.0->nvflare~=2.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.0->nvflare~=2.5.0rc) (3.10
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.28.0->nvflare~=2.5.0rc)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: pyparsing<4,>=2 in /usr/local/lib/python3.12/dist-packages (from pyhocon->nvflare~=2.5.0rc) (3.2.3)
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->cryptography>=36.0.0->nvflare~=2
Downloading nvflare-2.5.2-py3-none-any.whl (3.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.7/3.7 MB 10.4 MB/s eta 0:00:00
Downloading flask-3.0.2-py3-none-any.whl (101 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 101.3/101.3 kB 10.2 MB/s eta 0:00:00
Downloading Flask_JWT_Extended-4.6.0-py2.py3-none-any.whl (22 kB)
Downloading flask_sqlalchemy-3.1.1-py3-none-any.whl (25 kB)
Downloading SQLAlchemy-2.0.16-py3-none-any.whl (1.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.8/1.8 MB 64.5 MB/s eta 0:00:00
Downloading werkzeug-3.0.3-py3-none-any.whl (227 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 227.3/227.3 kB 21.6 MB/s eta 0:00:00
Downloading docker-7.1.0-py3-none-any.whl (147 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 147.8/147.8 kB 14.5 MB/s eta 0:00:00
Downloading gunicorn-23.0.0-py3-none-any.whl (85 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 85.0/85.0 kB 9.0 MB/s eta 0:00:00
Downloading numpy-1.26.4-cp312-cp312-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.0/18.0 MB 71.4 MB/s eta 0:00:00
Downloading pyhocon-0.3.61-py3-none-any.whl (25 kB)
Installing collected packages: Werkzeug, SQLAlchemy, pyhocon, numpy, gunicorn, Flask, docker, Flask-SQLAlchemy, Flask-JWT-Extended, nv
  Attempting uninstall: Werkzeug
    Found existing installation: Werkzeug 3.1.3
    Uninstalling Werkzeug-3.1.3:
      Successfully uninstalled Werkzeug-3.1.3
  Attempting uninstall: SQLAlchemy
    Found existing installation: SQLAlchemy 2.0.43
    Uninstalling SQLAlchemy-2.0.43:
      Successfully uninstalled SQLAlchemy-2.0.43
  Attempting uninstall: numpy
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: Flask
    Found existing installation: Flask 3.1.1
    Uninstalling Flask-3.1.1:
      Successfully uninstalled Flask-3.1.1
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the sourc
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompatible.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you have numpy 1.26.4 which is incompatible.
thinc 8.3.6 requires numpy<3.0.0,>=2.0.0, but you have numpy 1.26.4 which is incompatible.
Successfully installed Flask-3.0.2 Flask-JWT-Extended-4.6.0 Flask-SQLAlchemy-3.1.1 SQLAlchemy-2.0.16 Werkzeug-3.0.3 docker-7.1.0 gunic
WARNING: The following packages were previously imported in this runtime:
  [numpy]
You must restart the runtime in order to use newly installed versions.
```

RESTART SESSION

RNN Model

```
%%writefile src/traffic_rnn_fl.py
# src/traffic_rnn_fl.py
# Federated BiLSTM + Self-Attention client with:
# - Per-site z-score scaling (fit on TRAIN only)
# - FedBN (keep input BN local to each site)
# - Optional site embedding
# - Huber loss (SmoothL1) and log1p target
# - TensorBoard logging + CSV metric dump per round

import os
import random
import urllib.request
from pathlib import Path

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from sklearn.metrics import mean_absolute_error, mean_squared_error

import nvflare.client as flare
from nvflare.client.tracking import SummaryWriter

# ------------------------- Runtime & data -------------------------
DEVICE   = "cuda" if torch.cuda.is_available() else "cpu"
DATA_DIR = "/tmp/nvflare/data"
REPO_RAW = "https://raw.githubusercontent.com/javier-rodrigues/task1/main/data"

# ------------------------- Day-1 switches ------------------------
USE_BATCHNORM = True           # FedBN keeps these BN params local
USE_SITE_EMB  = True           # tiny learned site embedding per client
EMB_DIM       = 8
SEQ_LEN       = 24
LOCAL_EPOCHS  = 5
BATCH_SIZE    = 256
```

```
LR           = 3e-4
WEIGHT_DECAY  = 1e-4

def set_seed(seed: int = 42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
set_seed(42)

# =========================== Data ===============================
def load_partition(site_id: str,
                   seq_len: int = SEQ_LEN,
                   t_val: float = 0.15,
                   t_test: float = 0.15):
    """
    Returns: (X_tr, y_tr_log), (X_val, y_val_log), (X_te, y_te_log)
    X_* has shape [N, seq_len, F]; y_* is log1p-scaled.
    Per-site z-score uses TRAIN rows only (no leakage).
    """
    os.makedirs(DATA_DIR, exist_ok=True)
    fname      = f"{site_id}.csv"
    url        = f"{REPO_RAW}/{fname}"
    local_path = os.path.join(DATA_DIR, fname)
    print(f"[DEBUG] Downloading {url}")
    urllib.request.urlretrieve(url, local_path)

    df = pd.read_csv(local_path)
    if "timestamp" in df.columns:
        df = df.sort_values("timestamp").reset_index(drop=True)
    else:
        df = df.reset_index(drop=True)

    # target & features
    y_full = df["traffic_congestion"].astype(np.float32).values
    feat_df = df.drop(columns=["traffic_congestion", "timestamp", "location_id"], errors="ignore")
    X_full  = feat_df.astype(np.float32).values                      # [T, F]
    T, Fdim = X_full.shape

    # ---------- chronological split on rows (before scaling) ----------
    n_test  = int(round(T * t_test))
    n_val   = int(round(T * t_val))
    n_train = T - n_val - n_test
    tr_rows = slice(0,        n_train)
    va_rows = slice(n_train,  n_train + n_val)
    te_rows = slice(n_train + n_val, T)

    # ---------- per-site z-score using TRAIN only ----------
    mu = X_full[tr_rows].mean(axis=0, keepdims=True)
    sd = X_full[tr_rows].std(axis=0, keepdims=True) + 1e-6
    Xz = (X_full - mu) / sd

    # ---------- build windows AFTER scaling ----------
    def make_windows(X, y, win: int):
        # sliding windows over both axes; align target at window end
        sw = np.lib.stride_tricks.sliding_window_view(X, window_shape=(win, X.shape[1]))
        X_seq = sw.squeeze(1)                  # (n_seq, win, F)
        y_seq = y[win - 1:]                    # (n_seq,)
        return X_seq, y_seq

    X_all, y_all = make_windows(Xz, y_full, seq_len)
    n_seq = X_all.shape[0]

    # split in SEQUENCE domain (preserves chronology)
    n_test  = int(round(n_seq * t_test))
    n_val   = int(round(n_seq * t_val))
    n_train = n_seq - n_val - n_test
    sl_tr   = slice(0, n_train)
    sl_va   = slice(n_train, n_train + n_val)
    sl_te   = slice(n_train + n_val, n_seq)

    def tt(a): return torch.as_tensor(a, dtype=torch.float32, device=DEVICE)
    return (tt(X_all[sl_tr]), torch.log1p(tt(y_all[sl_tr]))), \
           (tt(X_all[sl_va]), torch.log1p(tt(y_all[sl_va]))), \
           (tt(X_all[sl_te]), torch.log1p(tt(y_all[sl_te])))
```

```
# ============================ Model ===============================
class SelfAttn(nn.Module):
    def __init__(self, hidden_dim: int, heads: int = 4):
        super().__init__()
        self.attn = nn.MultiheadAttention(embed_dim=hidden_dim, num_heads=heads, batch_first=True)
    def forward(self, x):                 # x:[B,T,H]
        y, _ = self.attn(x, x, x)
        return x + y                      # residual

class TrafficRNN(nn.Module):
    def __init__(self,
                 input_dim: int =17,
                 site_index: int = 0,
                 num_sites: int = 5,
                 hidden_dim: int = 128,
                 num_layers: int = 2,
                 use_bn: bool = USE_BATCHNORM,
                 use_site_emb: bool = USE_SITE_EMB,
                 emb_dim: int = EMB_DIM):
        super().__init__()
        self.use_bn = use_bn
        self.use_site_emb = use_site_emb

        feat_dim = input_dim
        if self.use_bn:
            # apply BN over feature axis -> use (B*T, F)
            self.bn_in = nn.BatchNorm1d(feat_dim)

        if self.use_site_emb:
            self.site_emb = nn.Embedding(num_sites, emb_dim)
            self.register_buffer("site_idx_buf", torch.as_tensor(site_index, dtype=torch.long))
            feat_dim += emb_dim

        self.lstm = nn.LSTM(feat_dim, hidden_dim, num_layers=num_layers,
                            bidirectional=True, batch_first=True, dropout=0.2)
        self.attn = SelfAttn(hidden_dim * 2)
        self.head = nn.Sequential(
            nn.LayerNorm(hidden_dim * 2),
            nn.Linear(hidden_dim * 2, 64), nn.ReLU(),
            nn.Linear(64, 1)
        )

    def forward(self, x):                 # x:[B,T,F]
        B, T, F = x.shape

        if self.use_bn:
            x = x.reshape(B * T, F)
            x = self.bn_in(x)
            x = x.reshape(B, T, F)

        if self.use_site_emb:
            e = self.site_emb(self.site_idx_buf)             # [emb_dim]
            e = e.unsqueeze(0).unsqueeze(0).expand(B, T, -1) # broadcast across [B,T]
            x = torch.cat([x, e], dim=-1)                    # [B,T,F+emb]

        h, _ = self.lstm(x)                                  # [B,T,2H]
        h = self.attn(h)
        h = h.mean(dim=1)                                    # global average over time
        out = self.head(h).squeeze(-1)                       # [B]
        return out

# FedBN: keep ALL bn_in.* params local (weights, bias, running stats)
def is_bn_param(name: str) -> bool:
    return name.startswith("bn_in.")

# ============================ Utils ===============================
@torch.no_grad()
def evaluate(model: nn.Module, X: torch.Tensor, y_log: torch.Tensor, batch: int = 1024):
    model.eval()
    preds, truth = [], []
    for i in range(0, len(X), batch):
        out = model(X[i:i + batch])
        preds.append(torch.expm1(out).cpu())
        truth.append(torch.expm1(y_log[i:i + batch]).cpu())
    preds = torch.cat(preds).numpy()
    truth = torch.cat(truth).numpy()
```

```
        mae  = mean_absolute_error(truth, preds)
        mse  = mean_squared_error(truth, preds)          # sklearn<=1.1 compatible
        rmse = np.sqrt(mse)
        mape = np.mean(np.abs((truth - preds) / (truth + 1e-5))) * 100.0
        return mae, rmse, mape

    def append_metrics_csv(site_id: str, round_num: int,
                           tr: tuple, va: tuple, te: tuple,
                           model_tag: str = "RNN",
                           out_dir: str = "/tmp/nvflare/metrics"):
        os.makedirs(out_dir, exist_ok=True)
        path = os.path.join(out_dir, f"{site_id}_{model_tag}.csv")
        header = not os.path.exists(path)
        with open(path, "a") as f:
            if header:
                f.write("round,model,site,split,mae,rmse,mape\n")
            for split_name, (mae, rmse, mape) in {"train": tr, "val": va, "test": te}.items():
                f.write(f"{round_num},{model_tag},{site_id},{split_name},{mae:.6f},{rmse:.6f},{mape:.6f}\n")

    # ============================== Main ==============================
    def main():
        flare.init()
        writer = SummaryWriter()

        site_id  = flare.get_site_name()             # "site-1" .. "site-5"
        site_idx = max(0, int(site_id.split("-")[-1]) - 1)

        (X_tr, y_tr), (X_va, y_va), (X_te, y_te) = load_partition(site_id)

        model = TrafficRNN(
            input_dim=X_tr.shape[-1],
            site_index=site_idx,
            num_sites=5,
            hidden_dim=128,
            num_layers=2
        ).to(DEVICE)

        opt     = torch.optim.AdamW(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
        loss_fn = nn.SmoothL1Loss()                    # Huber

        # One-off baseline on Test (round = -1)
        te_mae, te_rmse, te_mape = evaluate(model, X_te, y_te)
        writer.add_scalar("test/mape", te_mape, -1)

        while flare.is_running():
            fl_model = flare.receive()
            r = fl_model.current_round

            # ---- smart load with FedBN (skip bn_in.*) ----
            own = model.state_dict()
            for k, v in fl_model.params.items():
                if k in own and v.shape == own[k].shape and not is_bn_param(k):
                    own[k].copy_(v.to(DEVICE))

            # ---- local training ----
            model.train()
            idx = torch.randperm(len(X_tr), device=DEVICE)
            for _ in range(LOCAL_EPOCHS):
                for b in idx.split(BATCH_SIZE):
                    opt.zero_grad(set_to_none=True)
                    pred = model(X_tr[b])
                    loss = loss_fn(pred, y_tr[b])
                    loss.backward()
                    opt.step()

            # ---- evaluate ----
            tr_mae, tr_rmse, tr_mape = evaluate(model, X_tr, y_tr)
            va_mae, va_rmse, va_mape = evaluate(model, X_va, y_va)
            te_mae, te_rmse, te_mape = evaluate(model, X_te, y_te)

            writer.add_scalar("train/mape", tr_mape, r)
            writer.add_scalar("val/mape",   va_mape, r)
            writer.add_scalar("test/mape",  te_mape, r)

            # CSV metric dump (for tables)
            append_metrics_csv(site_id, r,
                               (tr_mae, tr_rmse, tr_mape),
                               (va_mae, va_rmse, va_mape),
                               (te_mae, te_rmse, te_mape),
                               model_tag="RNN")

            # ---- send model back (filter FedBN params) ----
            to_send = model.cpu().state_dict()
            to_send = {k: v for k, v in to_send.items() if not is_bn_param(k)}
            flare.send(
                flare.FLModel(
                    params=to_send,
                    metrics={
                        "tr_mae": tr_mae, "tr_rmse": tr_rmse, "tr_mape": tr_mape,
                        "val_mae": va_mae, "val_rmse": va_rmse, "val_mape": va_mape,
                        "te_mae": te_mae, "te_rmse": te_rmse, "te_mape": te_mape,
                    },
                    meta={"current_round": r},
                )
            )
            model.to(DEVICE)

    if __name__ == "__main__":
        main()
```

```
Writing src/traffic_rnn_fl.py
```

GNN Model

```
%%writefile src/traffic_gnn_fl.py
# src/traffic_gnn_fl.py
# Federated "GCN-style" MLP+Self-Attention client with:
# - Per-site z-score scaling (fit on TRAIN only)
# - FedBN (keep input BN local to each site)
# - Huber loss (SmoothL1) on log1p(TCI) targets
# - TensorBoard logging + CSV metric dump per round
# - Same data pipeline as the RNN for a fair comparison

import os
import random
import urllib.request
from pathlib import Path

import numpy as np
import pandas as pd
import torch
import torch.nn as nn
from sklearn.metrics import mean_absolute_error, mean_squared_error

import nvflare.client as flare
from nvflare.client.tracking import SummaryWriter

# ------------------------- Runtime & data -------------------------
DEVICE   = "cuda" if torch.cuda.is_available() else "cpu"
DATA_DIR = "/tmp/nvflare/data"
REPO_RAW = "https://raw.githubusercontent.com/javier-rodrigues/task1/main/data"

# ------------------------- Loss Toggle & Model Property Initialization -------------------------
USE_FEDBN     = True
USE_HUBER     = True
LOCAL_EPOCHS  = 5
BATCH_SIZE    = 256
LR            = 1e-3
WEIGHT_DECAY  = 1e-4

def set_seed(seed: int = 42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed_all(seed)
set_seed(42)

# =========================== Data ===============================
def load_partition(site_id: str, t_val: float = 0.15, t_test: float = 0.15):
    """
```

```
        Chronological split; per-site z-score using TRAIN rows only.
        Returns tensors: X_tr, y_tr_log, X_val, y_val_log, X_te, y_te_log
        """
        os.makedirs(DATA_DIR, exist_ok=True)
        fname      = f"{site_id}.csv"
        url        = f"{REPO_RAW}/{fname}"
        local_path = os.path.join(DATA_DIR, fname)
        print(f"[DEBUG] Downloading partition for {site_id} → {url}")
        urllib.request.urlretrieve(url, local_path)

        df = pd.read_csv(local_path)
        if "timestamp" in df.columns:
            df = df.sort_values("timestamp").reset_index(drop=True)
        else:
            df = df.reset_index(drop=True)

        y_all = df["traffic_congestion"].astype(np.float32).values

        drop_cols = {"traffic_congestion", "timestamp", "location_id"}
        feat_df   = df.drop(columns=[c for c in drop_cols if c in df.columns])
        X_np      = feat_df.astype(np.float32).values  # [T,F]
        T         = len(df)

        # ---------- chrono split on rows (before scaling) ----------
        n_test  = int(round(T * t_test))
        n_val   = int(round(T * t_val))
        n_train = T - n_val - n_test
        sl_tr   = slice(0, n_train)
        sl_va   = slice(n_train, n_train + n_val)
        sl_te   = slice(n_train + n_val, T)

        # ---------- per-site z-score using TRAIN statistics ----------
        mu = X_np[sl_tr].mean(axis=0, keepdims=True)
        sd = X_np[sl_tr].std(axis=0, keepdims=True) + 1e-6
        Xz = (X_np - mu) / sd

        def tt(a): return torch.from_numpy(a.astype(np.float32)).to(DEVICE)
        # log1p-transform targets for training stability
        return (
            tt(Xz[sl_tr]), torch.log1p(tt(y_all[sl_tr])),
            tt(Xz[sl_va]), torch.log1p(tt(y_all[sl_va])),
            tt(Xz[sl_te]), torch.log1p(tt(y_all[sl_te])),
        )

    # ============================ Model ===============================
    class TrafficGCN(nn.Module):
        """
        Treat each sample as a single-node 'graph':
        - BatchNorm on inputs (BN kept local via FedBN)
        - Linear projection -> hidden
        - 2 × Multi-Head Self-Attention blocks with residual FF
        - Global average pool over the (trivial) node axis
        - Small MLP regressor
        """
        def __init__(self, input_dim: int = 17, hidden_dim: int = 64, heads: int = 4):
            super().__init__()
            assert hidden_dim % heads == 0, "hidden_dim must be divisible by heads"

            self.bn_in   = nn.BatchNorm1d(input_dim)   # <-- FedBN keeps this local
            self.proj_in = nn.Linear(input_dim, hidden_dim)

            self.attn1 = nn.MultiheadAttention(embed_dim=hidden_dim, num_heads=heads, batch_first=True)
            self.ff1   = nn.Sequential(nn.Linear(hidden_dim, hidden_dim), nn.ReLU())

            self.attn2 = nn.MultiheadAttention(embed_dim=hidden_dim, num_heads=heads, batch_first=True)
            self.ff2   = nn.Sequential(nn.Linear(hidden_dim, hidden_dim), nn.ReLU())

            self.readout   = nn.AdaptiveAvgPool1d(1)
            self.regressor = nn.Sequential(
                nn.LayerNorm(hidden_dim),
                nn.Linear(hidden_dim, 32), nn.ReLU(),
                nn.Linear(32, 1)
            )

        def forward(self, x):            # x : [N, F]
            x = self.bn_in(x)
            h = self.proj_in(x)          # [N, H]
            h = h.unsqueeze(1)           # [N, 1, H]

            a1, _ = self.attn1(h, h, h); h = self.ff1(h + a1)
            a2, _ = self.attn2(h, h, h); h = self.ff2(h + a2)

            g = self.readout(h.transpose(1, 2)).squeeze(-1)  # [N, H]
            out = self.regressor(g).squeeze(-1)              # [N]
            return out

    # FedBN: keep ALL bn_in.* params local (weights, bias, running stats)
    def is_bn_param(name: str) -> bool:
        return name.startswith("bn_in.")

    # =========================== Utils ==============================
    @torch.no_grad()
    def evaluate(model: nn.Module, X: torch.Tensor, y_log: torch.Tensor, batch: int = 4096):
        model.eval()
        preds, truth = [], []
        for i in range(0, len(X), batch):
            out = model(X[i:i + batch])
            preds.append(torch.expm1(out).cpu())
            truth.append(torch.expm1(y_log[i:i + batch]).cpu())
        preds = torch.cat(preds).numpy()
        truth = torch.cat(truth).numpy()

        mae  = mean_absolute_error(truth, preds)
        mse  = mean_squared_error(truth, preds)     # sklearn<=1.1 compat
        rmse = np.sqrt(mse)
        mape = np.mean(np.abs((truth - preds) / (truth + 1e-5))) * 100.0
        return mae, rmse, mape

    def append_metrics_csv(site_id: str, round_num: int,
                           tr: tuple, va: tuple, te: tuple,
                           model_tag: str = "GNN",
                           out_dir: str = "/tmp/nvflare/metrics"):
        os.makedirs(out_dir, exist_ok=True)
        path = os.path.join(out_dir, f"{site_id}_{model_tag}.csv")
        header = not os.path.exists(path)
        with open(path, "a") as f:
            if header:
                f.write("round,model,site,split,mae,rmse,mape\n")
            for split_name, (mae, rmse, mape) in {"train": tr, "val": va, "test": te}.items():
                f.write(f"{round_num},{model_tag},{site_id},{split_name},{mae:.6f},{rmse:.6f},{mape:.6f}\n")

    # ============================= Main =============================
    def main():
        flare.init()
        writer = SummaryWriter()

        site_id = flare.get_site_name()  # "site-1" .. "site-5"
        X_tr, y_tr, X_va, y_va, X_te, y_te = load_partition(site_id)

        model = TrafficGCN(input_dim=X_tr.shape[1]).to(DEVICE)

        opt     = torch.optim.AdamW(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
        loss_fn = nn.SmoothL1Loss() if USE_HUBER else nn.MSELoss()

        # One-off baseline on Test (round = -1)
        te_mae, te_rmse, te_mape = evaluate(model, X_te, y_te)
        writer.add_scalar("test/mape", te_mape, -1)

        while flare.is_running():
            fl_model = flare.receive()
            r = fl_model.current_round

            # ---- smart load with FedBN (skip bn_in.*) ----
            own = model.state_dict()
            for k, v in fl_model.params.items():
                if k in own and v.shape == own[k].shape and not (USE_FEDBN and is_bn_param(k)):
                    own[k].copy_(v.to(DEVICE))

            # ---- local training ----
            model.train()
            idx = torch.randperm(len(X_tr), device=DEVICE)
            for _ in range(LOCAL_EPOCHS):
                for b in idx.split(BATCH_SIZE):
                    opt.zero_grad(set_to_none=True)
                    pred = model(X_tr[b])
                    loss = loss_fn(pred, y_tr[b])
                    loss.backward()
                    opt.step()

            # ---- evaluate ----
            tr_mae, tr_rmse, tr_mape = evaluate(model, X_tr, y_tr)
            va_mae, va_rmse, va_mape = evaluate(model, X_va, y_va)
            te_mae, te_rmse, te_mape = evaluate(model, X_te, y_te)

            writer.add_scalar("train/mape", tr_mape, r)
            writer.add_scalar("val/mape",   va_mape, r)
            writer.add_scalar("test/mape",  te_mape, r)

            # CSV metric dump (for tables)
            append_metrics_csv(site_id, r,
                               (tr_mae, tr_rmse, tr_mape),
                               (va_mae, va_rmse, va_mape),
                               (te_mae, te_rmse, te_mape),
                               model_tag="GNN")

            # ---- send model back (filter FedBN params if enabled) ----
            to_send = model.cpu().state_dict()
            if USE_FEDBN:
                to_send = {k: v for k, v in to_send.items() if not is_bn_param(k)}

            flare.send(
                flare.FLModel(
                    params=to_send,
                    metrics={
                        "tr_mae": tr_mae, "tr_rmse": tr_rmse, "tr_mape": tr_mape,
                        "val_mae": va_mae, "val_rmse": va_rmse, "val_mape": va_mape,
                        "te_mae": te_mae, "te_rmse": te_rmse, "te_mape": te_mape,
                    },
                    meta={"current_round": r},
                )
            )
            model.to(DEVICE)

    if __name__ == "__main__":
        main()
```

```
Writing src/traffic_gnn_fl.py
```

Federated Averaging with NVFlare

Now, we need to adapt this centralized training code to something that can run in a federated setting.

On the client side, the training workflow is as follows:

1. Receive the model from the FL server.
2. Perform local training on the received global model and/or evaluate the received global model for model selection.
3. Send the new model back to the FL server.

Using NVFlare's client API, we can easily adapt machine learning code that was written for centralized training and apply it in a federated scenario. For a general use case, there are three essential methods to achieve this using the Client API :

- `init()` : Initializes NVFlare Client API environment.
- `receive()` : Receives model from the FL server.
- `send()` : Sends the model to the FL server.

Run an NVFlare Job

Now that we have defined the FedAvg controller to run our federated compute workflow on the FL server, and our client training script to receive the global models, run local training, and send the results back to the FL server, we can put everything together using NVFlare's Job API.

2. Define a FedJob

The `FedJob` is used to define how controllers and executors are placed within a federated job using the `to(object, target)` routine.

Here we use a PyTorch `BaseFedJob`, where we can define the job name and the initial global model. The `BaseFedJob` automatically configures components for model persistence, model selection, and TensorBoard streaming for convenience.

3. Define the Controller Workflow + Run & Export Jobs

Define the controller workflow and send to server.

```
# --- Common imports for jobs ---
from nvflare.app_common.workflows.fedavg import FedAvg
from nvflare.app_opt.pt.job_config.base_fed_job import BaseFedJob
from nvflare.job_config.script_runner import ScriptRunner

# ---- Detect feature dimension from site-1 to avoid shape mismatches ----
from src.traffic_rnn_fl import load_partition as rnn_load, TrafficRNN
from src.traffic_gnn_fl import load_partition as gnn_load, TrafficGCN

(_, _), (_, _), (X_te_probe, _) = rnn_load("site-1")   # downloads site-1.csv
INPUT_DIM = X_te_probe.shape[-1]
print("Detected input_dim =", INPUT_DIM)

# ---------------- RNN job ----------------
rnn_job = BaseFedJob(
    name="traffic_congestion_rnn_fedavg",
    initial_model=TrafficRNN(input_dim=INPUT_DIM),
    min_clients=5
)
rnn_job.to(FedAvg(num_clients=5, num_rounds=100), "server")
for i in range(5):
    rnn_job.to(ScriptRunner(script="src/traffic_rnn_fl.py", script_args=""), f"site-{i+1}")
rnn_export = "/tmp/nvflare/jobs/job_config_rnn"
import shutil, pathlib; shutil.rmtree(rnn_export, ignore_errors=True); pathlib.Path(rnn_export).mkdir(parents=True, exist_ok=True)
rnn_job.export_job(rnn_export)

!nvflare simulator -w /tmp/nvflare/workdir_rnn -n 5 -t 5 -gpu 0 /tmp/nvflare/jobs/job_config_rnn/traffic_congestion_rnn_fedavg

# ---------------- GNN job ----------------
gnn_job = BaseFedJob(
    name="traffic_congestion_gnn_fedavg",
    initial_model=TrafficGCN(input_dim=INPUT_DIM),
    min_clients=5
)
gnn_job.to(FedAvg(num_clients=5, num_rounds=100), "server")
for i in range(5):
    gnn_job.to(ScriptRunner(script="src/traffic_gnn_fl.py", script_args=""), f"site-{i+1}")
gnn_export = "/tmp/nvflare/jobs/job_config_gnn"
shutil.rmtree(gnn_export, ignore_errors=True); pathlib.Path(gnn_export).mkdir(parents=True, exist_ok=True)
gnn_job.export_job(gnn_export)

!nvflare simulator -w /tmp/nvflare/workdir_gnn -n 5 -t 5 -gpu 0 /tmp/nvflare/jobs/job_config_gnn/traffic_congestion_gnn_fedavg
```

```
    2025-08-26 10:17:00,968 - ClientRunner - INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
```

```
2025-08-26 19:17:00,968 - ClientRunner - INFO - [identity=site-1, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:00,968 - ClientTaskWorker - INFO - Finished one task run for client: site-1 interval: 2 task_processed: True
2025-08-26 19:17:00,973 - ClientRunner - INFO - [identity=site-5, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:00,973 - ClientRunner - INFO - [identity=site-5, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:00,973 - ClientRunner - INFO - [identity=site-5, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:00,974 - Cell - INFO - broadcast: channel='aux_communication', topic='__task_check__', targets=['server.simulate_job'
2025-08-26 19:17:00,977 - ClientRunner - INFO - [identity=site-5, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:00,977 - FederatedClient - INFO - Starting to push execute result.
2025-08-26 19:17:00,980 - ServerRunner - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=sim
2025-08-26 19:17:00,981 - IntimeModelSelector - WARNING - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, pe
2025-08-26 19:17:01,203 - ServerRunner - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=sim
2025-08-26 19:17:01,203 - WFCommServer - INFO - [identity=simulator_server, run=simulate_job, wf=controller]: task train exit with sta
2025-08-26 19:17:01,204 - SubmitUpdateCommand - INFO - submit_update process. client_name:site-5   task_id:85cd1ee0-012f-4d74-abec-879
2025-08-26 19:17:01,205 - Communicator - INFO -  SubmitUpdate size: 189.4KB (189418 Bytes). time: 0.228385 seconds
2025-08-26 19:17:01,205 - ClientRunner - INFO - [identity=site-5, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:01,206 - ClientTaskWorker - INFO - Finished one task run for client: site-5 interval: 2 task_processed: True
2025-08-26 19:17:01,403 - FedAvg - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=simulate_
2025-08-26 19:17:01,404 - FedAvg - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=simulate_
2025-08-26 19:17:01,406 - FedAvg - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=simulate_
2025-08-26 19:17:01,407 - FedAvg - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=simulate_
2025-08-26 19:17:01,407 - FedAvg - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=simulate_
2025-08-26 19:17:01,407 - FedAvg - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=simulate_
2025-08-26 19:17:01,407 - WFCommServer - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-5, peer_run=sim
2025-08-26 19:17:01,743 - ServerRunner - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-3, peer_run=sim
2025-08-26 19:17:01,744 - ServerRunner - INFO - [identity=simulator_server, run=simulate_job, wf=controller, peer=site-3, peer_run=sim
2025-08-26 19:17:01,744 - GetTaskCommand - INFO - return task to client.  client_name: site-3  task_name: train   task_id: d1d84d41-91
2025-08-26 19:17:01,748 - Communicator - INFO - Received from simulator_server server. getTask: train size: 189.5KB (189509 Bytes) tim
2025-08-26 19:17:01,748 - FederatedClient - INFO - pull_task completed. Task name:train Status:True
2025-08-26 19:17:01,748 - ClientRunner - INFO - [identity=site-3, run=simulate_job, peer=simulator_server, peer_run=simulate_job]: got
2025-08-26 19:17:01,748 - ClientRunner - INFO - [identity=site-3, run=simulate_job, peer=simulator_server, peer_run=simulate_job, task
2025-08-26 19:17:01,748 - PTInProcessClientAPIExecutor - INFO - [identity=site-3, run=simulate_job, peer=simulator_server, peer_run=si
2025-08-26 19:17:01,749 - PTInProcessClientAPIExecutor - INFO - [identity=site-3, run=simulate_job, peer=simulator_server, peer_run=si
2025-08-26 19:17:01,749 - PTInProcessClientAPIExecutor - INFO - [identity=site-3, run=simulate_job, peer=simulator_server, peer_run=si
2025-08-26 19:17:01,749 - PTInProcessClientAPIExecutor - INFO - [identity=site-3, run=simulate_job, peer=simulator_server, peer_run=si
```

Aggregated Per Site Metrics for Train/Val/Test Splits

```python
# Show aggregated per-site final MAPE table
import glob, pandas as pd, os
paths = glob.glob("/tmp/nvflare/metrics/*.csv")
df = pd.concat([pd.read_csv(p) for p in paths], ignore_index=True)

last_round = df.groupby(["model","site"])["round"].max().reset_index()
final = df.merge(last_round, on=["model","site","round"], how="inner")
final_mape = final.pivot_table(index=["model","site"], columns="split", values="mape")
display(final_mape.sort_index())

os.makedirs("/tmp/nvflare/summary", exist_ok=True)
df.to_csv("/tmp/nvflare/summary/metrics_all_rounds.csv", index=False)
final_mape.to_csv("/tmp/nvflare/summary/final_mape_by_model_site.csv", index=True)
print("Saved: /tmp/nvflare/summary/metrics_all_rounds.csv and final_mape_by_model_site.csv")
```

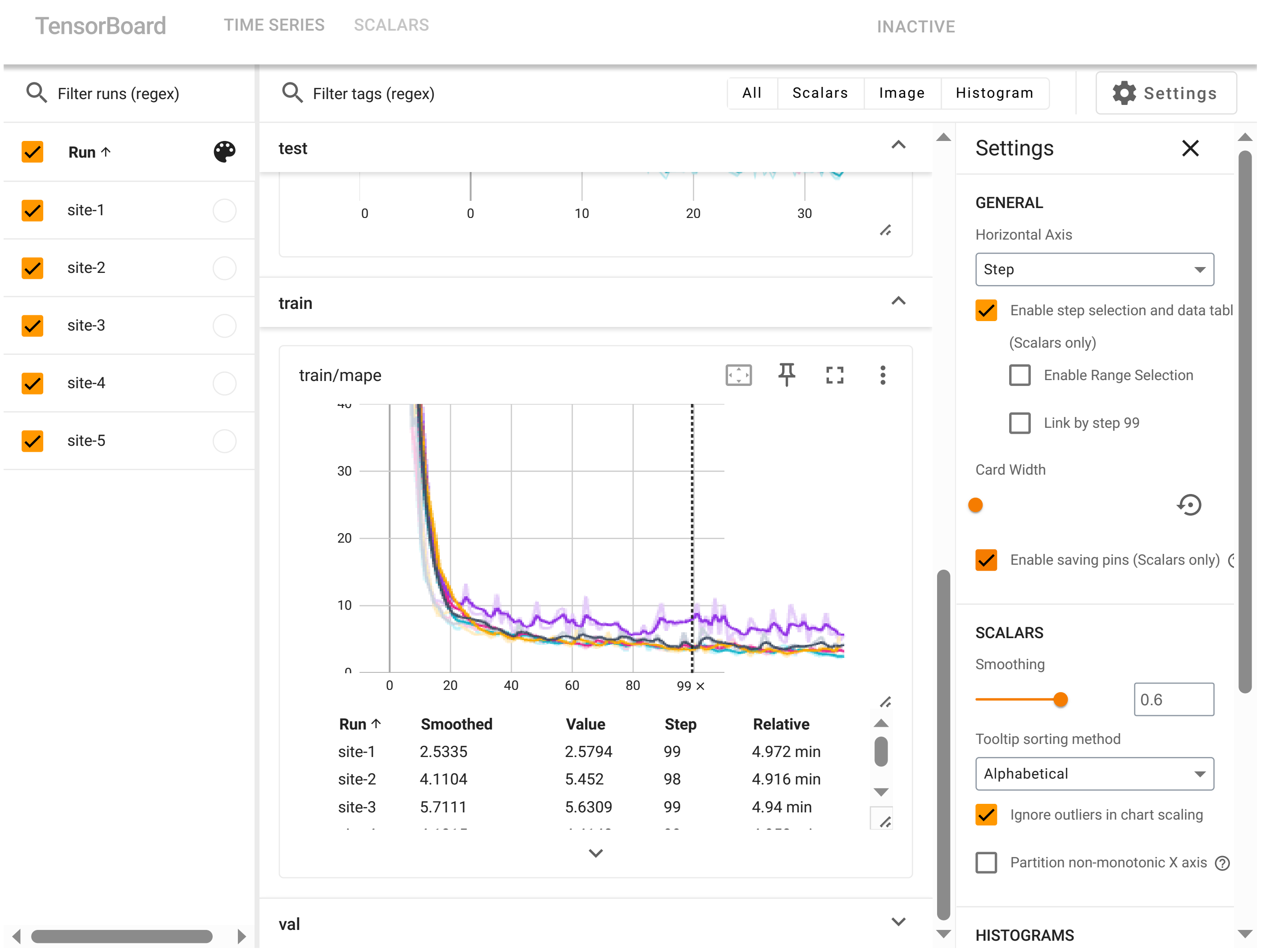
	split	test	train	val
model	site			
GNN	site-1	4.465880	2.579365	3.375899
	site-2	6.213045	3.706142	6.430436
	site-3	6.343228	5.630856	5.706332
	site-4	6.208234	4.414772	5.168714
	site-5	4.185732	3.017922	4.526762
RNN	site-1	4.582676	3.324891	4.470011
	site-2	5.522890	3.971411	5.598060
	site-3	4.055430	3.168976	3.059402
	site-4	8.332928	8.807396	9.013575
	site-5	6.217823	4.592663	5.487363

```
Saved: /tmp/nvflare/summary/metrics_all_rounds.csv and final_mape_by_model_site.csv
```

Tensorboard Per Round Metrics

```
import glob, os, pathlib
cands = glob.glob("/tmp/nvflare/**/tb_events", recursive=True)
event_dir = max(cands, key=lambda p: os.path.getmtime(pathlib.Path(p).parent))
print("TensorBoard logdir →", event_dir)

%reload_ext tensorboard
%tensorboard --logdir {event_dir}
```

```
TensorBoard logdir → /tmp/nvflare/workdir_gnn/server/simulate_job/tb_events
```



Run ↑	Smoothed	Value	Step	Relative
site-1	2.5335	2.5794	99	4.972 min
site-2	4.1104	5.452	98	4.916 min
site-3	5.7111	5.6309	99	4.94 min

Settings

GENERAL

Horizontal Axis: Step

✓ Enable step selection and data tabl (Scalars only)

☐ Enable Range Selection

☐ Link by step 99

Card Width

✓ Enable saving pins (Scalars only)

SCALARS

Smoothing: 0.6

Tooltip sorting method: Alphabetical

✓ Ignore outliers in chart scaling

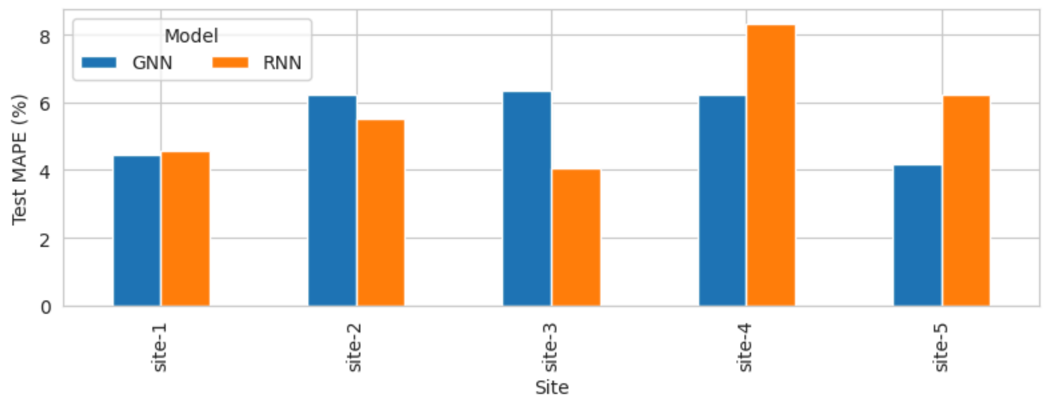
☐ Partition non-monotonic X axis

HISTOGRAMS

Final Results Assets

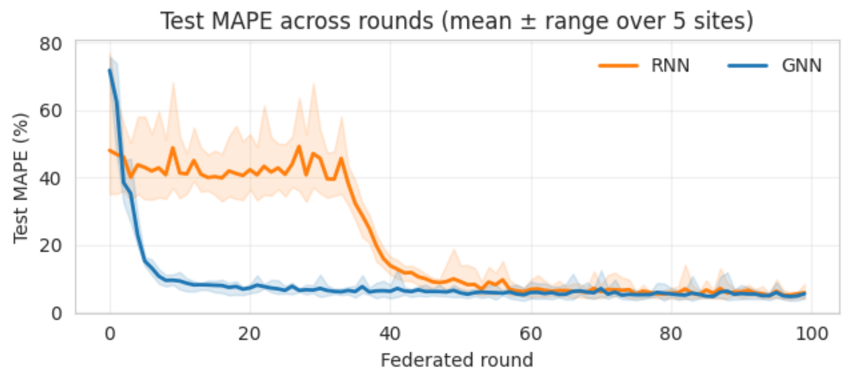
Per Site Model Comparison

```
# Bar chart
import pandas as pd, matplotlib.pyplot as plt
pv = pd.read_csv("/tmp/nvflare/summary/final_mape_by_model_site.csv", index_col=[0,1])  # MultiIndex
test = pv["test"].unstack(0)  # columns=model, rows=site
ax = test.plot(kind="bar", figsize=(8,3.2))
ax.set_ylabel("Test MAPE (%)"); ax.set_xlabel("Site")
ax.legend(title="Model", ncol=2); plt.tight_layout()
plt.savefig("/tmp/nvflare/summary/fig_test_mape_bar.png", dpi=300)
```



MAPE Evaluation Model Comparison

```
# Learning curves: mean ± band across sites for each model
import pandas as pd, matplotlib.pyplot as plt, numpy as np
from pathlib import Path

Path("/tmp/nvflare/summary").mkdir(parents=True, exist_ok=True)
curves = pd.read_csv("/tmp/nvflare/summary/metrics_all_rounds.csv")
# expected columns: round, model, site, split, mae, rmse, mape

# use TEST split only
te = curves[curves["split"]=="test"].copy()

def plot_model(ax, df_model, label, color):
    g = (df_model.groupby("round")["mape"]
                 .agg(["mean","min","max"]).reset_index())
    ax.plot(g["round"], g["mean"], label=label, color=color, lw=2)
    ax.fill_between(g["round"], g["min"], g["max"],
                    alpha=0.15, color=color)

fig, ax = plt.subplots(figsize=(6.5,3))
plot_model(ax, te[te["model"]=="RNN"], "RNN", "tab:orange")
plot_model(ax, te[te["model"]=="GNN"], "GNN", "tab:blue")
ax.set_xlabel("Federated round"); ax.set_ylabel("Test MAPE (%)")
ax.set_title("Test MAPE across rounds (mean ± range over 5 sites)")
ax.grid(True, alpha=.3); ax.legend(frameon=False, ncol=2)
plt.tight_layout(); plt.savefig("/tmp/nvflare/summary/fig_learning_curves.png", dpi=300)
plt.show()
```



Model Evaluation Metrics

```
# === Paper table: mean ± std across sites (TEST split, final round) ===
import pandas as pd
from pathlib import Path

Path("/tmp/nvflare/summary").mkdir(parents=True, exist_ok=True)

df = pd.read_csv("/tmp/nvflare/summary/metrics_all_rounds.csv")
# keep TEST only
```

```
te = df[df["split"] == "test"].copy()

# final round per (model, site)
final = (te.sort_values("round")
           .groupby(["model", "site"], as_index=False)
           .tail(1))

# aggregate across sites for each model
agg = final.groupby("model").agg(
    mae_mean=("mae", "mean"),  mae_std=("mae", "std"),
    rmse_mean=("rmse", "mean"), rmse_std=("rmse", "std"),
    mape_mean=("mape", "mean"), mape_std=("mape", "std"),
).round(3)

summary = pd.DataFrame({
    "Model": agg.index,
    "MAE (↓)":  [f"{m:.3f} ± {s:.3f}" for m, s in zip(agg.mae_mean,  agg.mae_std)],
    "RMSE (↓)": [f"{m:.3f} ± {s:.3f}" for m, s in zip(agg.rmse_mean, agg.rmse_std)],
    "MAPE (↓)": [f"{m:.3f} ± {s:.3f}" for m, s in zip(agg.mape_mean, agg.mape_std)],
}).reset_index(drop=True)

# Save CSV (and optional LaTeX for direct paste in the paper)
summary.to_csv("/tmp/nvflare/summary/table_summary_by_model.csv", index=False)
try:
    with open("/tmp/nvflare/summary/table_summary_by_model.tex", "w") as f:
        f.write(summary.to_latex(index=False, escape=False))
except Exception:
    pass

print(summary.to_markdown(index=False))
```

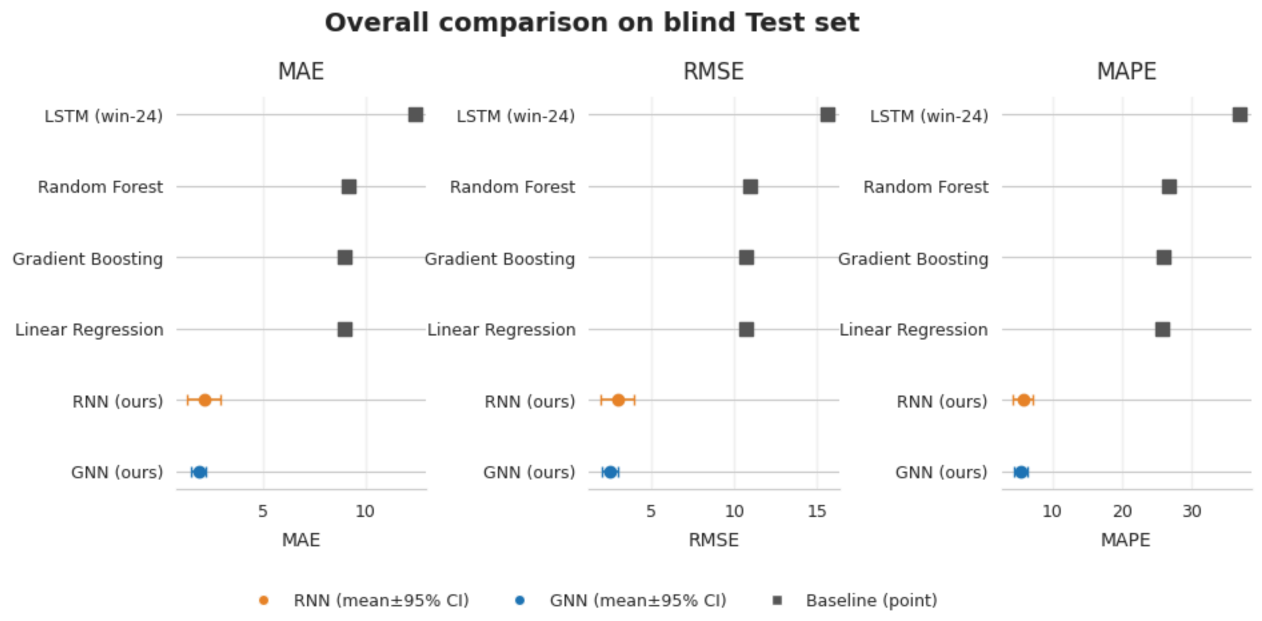
```
| Model   | MAE (↓)       | RMSE (↓)      | MAPE (↓)      |
|:--------|:--------------|:--------------|:--------------|
| GNN     | 1.900 ± 0.420 | 2.479 ± 0.564 | 5.483 ± 1.063 |
| RNN     | 2.170 ± 0.918 | 2.908 ± 1.174 | 5.742 ± 1.672 |
```

Baseline Models vs Deep Learning Models Evaluation Comparison

```
# === Save baselines + build final MAE/RMSE/MAPE comparison figure ===
import pandas as pd, numpy as np, matplotlib.pyplot as plt
from pathlib import Path

# 1) Save baselines from res_df
summary_dir = Path("/tmp/nvflare/summary"); summary_dir.mkdir(parents=True, exist_ok=True)
baseline_out = summary_dir / "baseline_results.csv"
res_df[["Model","MAE","RMSE","MAPE"]].to_csv(baseline_out, index=False)
print(f"Saved baseline results to: {baseline_out}")

# 2) Load our FL results (RNN/GNN) and baselines
metrics_path  = summary_dir / "metrics_all_rounds.csv"
assert metrics_path.exists(), f"Missing {metrics_path}"
curves = pd.read_csv(metrics_path)

te = curves[curves["split"]=="test"].sort_values("round")
final = te.groupby(["model","site"], as_index=False).tail(1)

def agg_ci(d, col):
    g = (d.groupby("model")[col]
           .agg(["mean","std","count"])
           .assign(ci95=lambda x: 1.96*x["std"]/np.sqrt(x["count"])))
    g.index = g.index.map(lambda s: f"{s} (ours)")
    return g[["mean","ci95"]]

ours = {
    "MAE":  agg_ci(final, "mae"),
    "RMSE": agg_ci(final, "rmse"),
    "MAPE": agg_ci(final, "mape"),
}

base = pd.read_csv(baseline_out).set_index("Model")
base_long = {
    "MAE":  base[["MAE"]].rename(columns={"MAE":"mean"}).assign(ci95=np.nan),
    "RMSE": base[["RMSE"]].rename(columns={"RMSE":"mean"}).assign(ci95=np.nan),
    "MAPE": base[["MAPE"]].rename(columns={"MAPE":"mean"}).assign(ci95=np.nan),
}
```

```
# --- helper (safe to redeclare) ---
def combine(metric):
    a = ours[metric].copy();  a["label"] = a.index; a["kind"]="ours"
    b = base_long[metric].copy(); b["label"] = b.index; b["kind"]="baseline"
    out = pd.concat([a.reset_index(drop=True), b.reset_index(drop=True)], axis=0)
    return out.sort_values("mean", ascending=False).reset_index(drop=True)

# build a GridSpec with explicit margins & spacing
fig = plt.figure(figsize=(13.0, 4.6))
gs  = fig.add_gridspec(
    nrows=1, ncols=3,
    left=0.36,    # extra room for long y tick labels
    right=0.98,
    top=0.86,     # reserved area for main title
    bottom=0.22,  # reserved area for legend
    wspace=0.65   # more horizontal space between panels
)

axes = [fig.add_subplot(gs[0, i]) for i in range(3)]

colors = {"RNN (ours)":"#E6862A", "GNN (ours)":"#1F77B4"}
baseline_color = "0.35"

for ax, metric in zip(axes, ["MAE","RMSE","MAPE"]):
    d = combine(metric); y = np.arange(len(d))
    ax.grid(axis="x", alpha=0.25); ax.set_axisbelow(True)

    # draw points (error bars for our models)
    for i, row in d.iterrows():
        lbl, m, c = row["label"], row["mean"], row["ci95"]
        if row["kind"] == "ours":
            ax.errorbar(m, y[i], xerr=c, fmt='o',
                        color=colors.get(lbl, "C0"),
                        ecolor=colors.get(lbl, "C0"),
                        capsize=3, lw=1.2)
        else:
            ax.scatter(m, y[i], s=45, color=baseline_color, marker='s')

    # styling
    ax.set_yticks(y); ax.set_yticklabels(d["label"], fontsize=9)
    ax.invert_yaxis()
    ax.set_xlabel(metric, fontsize=10, labelpad=6)
    ax.set_title(metric, fontsize=12, pad=10, loc="center", x=0.5)  # **centered title**
    for spine in ["top", "right", "left"]:
        ax.spines[spine].set_visible(False)
    ax.tick_params(axis="x", labelsize=9)

# centered main title (within reserved top margin)
fig.suptitle(
    "Overall comparison on blind Test set",
    fontsize=14, fontweight="bold", x=0.6, y=1, ha="center"
)

# legend centered below panels (within reserved bottom margin)
handles = [
    plt.Line2D([0],[0], marker='o', color='w',
               markerfacecolor=colors["RNN (ours)"], label="RNN (mean±95% CI)"),
    plt.Line2D([0],[0], marker='o', color='w',
               markerfacecolor=colors["GNN (ours)"], label="GNN (mean±95% CI)"),
    plt.Line2D([0],[0], marker='s', color='w',
               markerfacecolor=baseline_color, label="Baseline (point)"),
]
fig.legend(handles=handles, loc="lower center", ncol=3, frameon=False,
           bbox_to_anchor=(0.6, 0.0), fontsize=9)

out_path = summary_dir / "fig_all_models_mae_rmse_mape.png"
plt.savefig(out_path, dpi=300, bbox_inches="tight", pad_inches=0.1)
plt.show()
print(f"Saved figure: {out_path}")
```

```
Saved baseline results to: /tmp/nvflare/summary/baseline_results.csv
```



```
Saved figure: /tmp/nvflare/summary/fig_all_models_mae_rmse_mape.png
```

Asset Freeze & Collection

```
!pip install kaleido
```

```
Requirement already satisfied: kaleido in /usr/local/lib/python3.12/dist-packages (1.0.0)
Requirement already satisfied: choreographer>=1.0.5 in /usr/local/lib/python3.12/dist-packages (from kaleido) (1.0.10)
Requirement already satisfied: logistro>=1.0.8 in /usr/local/lib/python3.12/dist-packages (from kaleido) (1.1.0)
Requirement already satisfied: orjson>=3.10.15 in /usr/local/lib/python3.12/dist-packages (from kaleido) (3.11.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from kaleido) (25.0)
Requirement already satisfied: simplejson>=3.19.3 in /usr/local/lib/python3.12/dist-packages (from choreographer>=1.0.5->kaleido) (3.20.
```

Compatability check for asset freeze

```
# --- Plotly/Kaleido compatibility pin & reload ---
import sys, subprocess, importlib
from packaging.version import Version as V

import plotly
print("Plotly version:", plotly.__version__)

# Decide the Kaleido version that matches your Plotly
target_kaleido = "0.2.1" if V(plotly.__version__) < V("6.1.1") else "1.0.0"

# (Re)install the right Kaleido version
subprocess.check_call([sys.executable, "-m", "pip", "install", "-qU", f"kaleido=={target_kaleido}"])

# Purge any previously imported kaleido modules so the new version is used
for m in list(sys.modules):
    if m.startswith("kaleido"):
        sys.modules.pop(m)

# Import and report versions
import kaleido
print("Kaleido version:", getattr(kaleido, "__version__", "unknown"))
```

```
Plotly version: 5.24.1
Kaleido version: 0.2.1
```

Collect Assets for Research Paper

```
# === Paper asset freeze: collect ALL figures into one folder ===
# - Rebuilds EDA PNGs
# - Copies previously saved NVFlare results PNGs/tables
# - Writes a MANIFEST and a ZIP for Overleaf/GitHub

# 0) deps for static Plotly export

import os, io, shutil, glob, json, textwrap, math
from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# ------------------ paths ------------------
DATA_CSV     = Path("smart_traffic_management_dataset.csv")
SUMMARY_DIR  = Path("/tmp/nvflare/summary")                   # where FL results were saved
ASSETS_DIR   = Path("/tmp/nvflare/paper_assets")              # final folder
ASSETS_DIR.mkdir(parents=True, exist_ok=True)

assert DATA_CSV.exists(), f"Couldn't find {DATA_CSV}. Put the dataset next to the notebook or update DATA_CSV."

# ------------------ helpers ------------------
def save_plotly(fig, out_path, width=1000, height=500, scale=2):
    """
    Save a Plotly figure to PNG using Kaleido.
    Works with both Kaleido 0.2.x (Plotly 5.x) and Kaleido 1.x (Plotly 6.x).
    """
    from pathlib import Path
    out_path = Path(out_path)

    # Try the standard Plotly writer first
    try:
        fig.write_image(
            str(out_path),
            engine="kaleido",
            width=width,
            height=height,
            scale=scale,
        )
        return out_path
    except Exception as e1:
        # Fallback: call Kaleido's scope directly (handle both import paths)
        try:
            try:
                # Kaleido 0.2.x
                from kaleido.scopes.plotly import PlotlyScope
            except Exception:
                # Kaleido 1.x
                from kaleido import PlotlyScope

            scope = PlotlyScope()
            image_bytes = scope.transform(
                fig.to_plotly_json(),
                format="png",
                width=width,
                height=height,
                scale=scale,
            )
            with open(out_path, "wb") as f:
                f.write(image_bytes)
            return out_path
        except Exception as e2:
            raise RuntimeError(
                "Kaleido export failed.\n"
                f"write_image error: {e1}\n"
                f"fallback error:  {e2}"
            )


def save_matplotlib(fig, out_path, dpi=300, bbox="tight"):
    out_path = Path(out_path)
    fig.savefig(out_path, dpi=dpi, bbox_inches=bbox)
    plt.close(fig)
    return out_path

def cp_if_exists(path, dest_dir):
    path = Path(path)
    if path.exists():
        return shutil.copy(path, dest_dir / path.name)
    return None

manifest = []

def add_manifest(path, note):
    if path and Path(path).exists():
        sz_kb = Path(path).stat().st_size / 1024
        manifest.append({"file": str(Path(path).name), "kb": round(sz_kb,1), "note": note})

# ------------------ recompute EDA frame (incl. TCI) ------------------
df = pd.read_csv(DATA_CSV)
df["timestamp"] = pd.to_datetime(df["timestamp"])
df["hour"]      = df["timestamp"].dt.hour
df["dow"]       = df["timestamp"].dt.day_name().str[:3]  # Mon, Tue …

FFS, CAP = 65, 1800  # free-flow speed (km/h), capacity (veh/h/lane)

def tci(r):
    dens  = (r.vehicle_count_cars + 1.5*r.vehicle_count_trucks + 0.3*r.vehicle_count_bikes)/CAP
    speed = max(0, 1 - r.avg_vehicle_speed/FFS)
    inc   = 0.25 if r.accident_reported else 0
    sig   = {"Green":0, "Yellow":.05, "Red":.20}.get(r.signal_status, 0)
    wet   = .15 if r.weather_condition in ("Rainy","Foggy","Snowy") else (.05 if r.weather_condition=="Windy" else 0)
    return 100*(.55*dens + .30*speed + inc + sig + wet)

df["TCI"] = df.apply(tci, axis=1)

# ------------------ 1) EDA FIGURES (10 PNGs) ------------------
sns.set_style("whitegrid")

# A. TCI histogram
figA = px.histogram(df, x="TCI", nbins=40, marginal="box",
                    labels=dict(TCI="TCI (0=free-flow, 100=grid-lock)"),
                    title="TCI – overall distribution")
pA = save_plotly(figA, ASSETS_DIR/"eda_tci_hist.png")
add_manifest(pA, "EDA: TCI histogram")

# B. TCI heatmap (hour × location)
hm = (df.groupby(["location_id","hour"]).TCI.mean()
        .reset_index().pivot(index="location_id", columns="hour", values="TCI"))
figB = px.imshow(hm, aspect="auto", color_continuous_scale="magma_r",
                 labels=dict(x="Hour of day", y="Location", color="Mean TCI"),
                 title="Hourly mean TCI by location")
pB = save_plotly(figB, ASSETS_DIR/"eda_tci_heatmap.png", width=1000, height=450)
add_manifest(pB, "EDA: TCI heatmap by hour×location")

# C. Violin by weekday
figC = px.violin(df, x="dow", y="TCI", box=True, points=False,
                 category_orders={"dow":["Mon","Tue","Wed","Thu","Fri","Sat","Sun"]},
                 title="TCI distribution by weekday", labels=dict(dow="Weekday", TCI="TCI"))
pC = save_plotly(figC, ASSETS_DIR/"eda_tci_violin_weekday.png")
add_manifest(pC, "EDA: TCI by weekday (violin)")

# D. Box by weather
wx_colors = dict(Sunny="gold", Cloudy="lightslategray", Windy="mediumslateblue", Rainy="steelblue", Foggy="thistle")
figD = px.box(df, x="weather_condition", y="TCI", points="outliers",
              color="weather_condition", color_discrete_map=wx_colors,
              title="TCI vs. weather condition", labels=dict(weather_condition="Weather", TCI="TCI"))
figD.update_layout(showlegend=False)
```