

# DashPilot AI Agent: Dynamic Urban Mobility Dashboarding

Fahad Shaikh<sup>1</sup> Gemini 3.0 Flash (Preview)<sup>2</sup>

<sup>1</sup>Pace Univserity <sup>2</sup>fs12516n@pace.edu

## Motivation & Problem

Urban mobility systems generate rich trip data, but extracting insights often requires SQL, notebooks, or BI tooling. Stakeholders want **fast exploration** (dashboard) and **flexible investigation** (ad-hoc queries) without writing code.

**Goal:** Provide a single application where users can:

- Explore a local dataset instantly (no backend dependency).
- Query a Supabase database for operational/analyst views.
- Add new charts and tables to the dashboard through a guided chatbot.

## Core Idea: Two Data Planes + One UX

Dashboard = Local CSV for speed and portability. AI Analyst = Supabase for searchable tables and filtered retrieval.

**Why this works:**

- Local CSV enables sub-second rendering of aggregates (hours, day-of-week, month).
- Supabase supports read-only queries for "latest trips", "station contains X", etc.
- A **Widget Catalog** constrains what the chatbot can add (reliable UI; fewer hallucinations).

## Data Handling & Cleaning (Local CSV Engine)

**Local ingestion:** PapaParse loads `trips_rows.csv` in the browser. Typical cleaning:

- Validate timestamps (`started_at`, `ended_at`)
- Filter invalid durations (e.g., negative; cap extreme trips such as  $\leq 240$  minutes)
- Compute aggregates: hourly counts, day-of-week counts, month counts, top stations/routes

**Outputs:** Aggregates (for charts) + sampled rows (for local tables).

## Implementation Stack

React 18 + Vite  
Recharts (interactive viz)  
PapaParse (CSV)

Supabase (Postgres)  
Gemini 3.0 Flash (Preview)

## Product UX: Dashboard + AI Analyst

**Dashboard (Local):** Baseline overview + pinned insights grid.

**AI Analyst (Supabase):** Chat-driven analysis that can:

- Return database tables (latest trips, filtered trips)
- Offer a **menu of addable widgets** (charts/tables) powered by local aggregates
- Preview results in chat, then **Add/Pin** to the dashboard

## Widget Catalog: New Addable Visuals & Tables

Instead of reusing the same default dashboard charts, the chatbot exposes newer add-ons such as:

**Charts:** Trips by Month (Area), Trips by Day-of-Week (Bar), Duration Histogram, Bike Type Split (Donut), Top Routes (Bar), Member vs Casual by Day (Stacked)

**Tables:** Latest trips (Local CSV), Top stations (Local), Supabase query results (DB)

**Flow:** Show widget menu → Preview → Add/Pin → Dashboard grid updates.

## AI Guardrails: Structured Actions (Tool JSON)

The model is prompted to emit **structured actions**, not free-form code:

```
show_menu
preview_widget(widgetId)
add_widget(widgetId)
supabase select with limited fields: columns, filters, order, limit
```

This reduces schema hallucination and ensures predictable rendering.

## Suggested Figure Inserts (from your UI)

Replace placeholders with screenshots exported from your app:

`figures/dashboard-screenshot.png`  
`figures/widget-menu-screenshot.png`

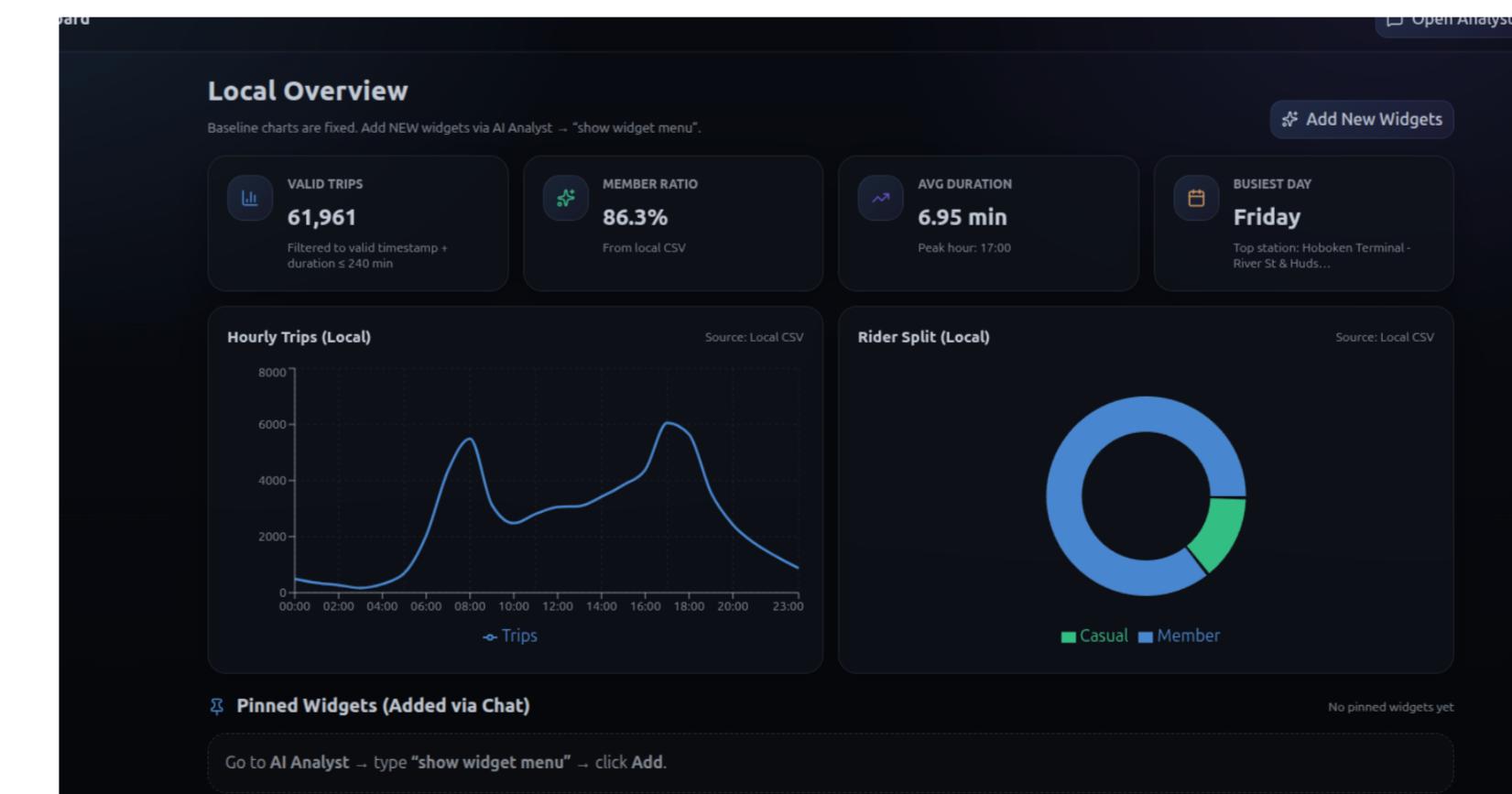


Figure 1. Dashboard view with baseline charts and pinned widgets (example screenshot).

## Architecture Workflow (Hybrid Local + Supabase + Widget Catalog)

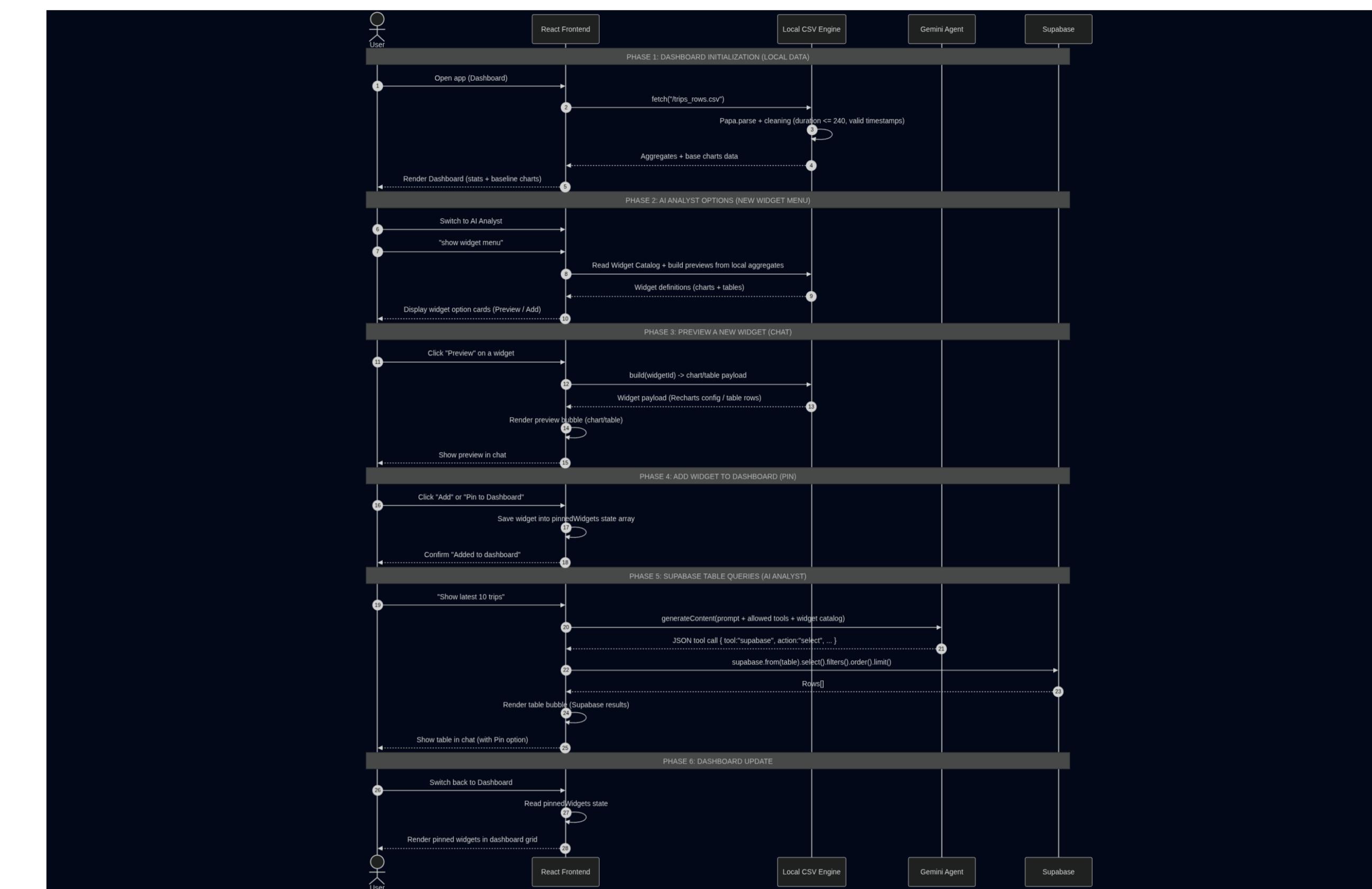


Figure 2. End-to-end workflow: local CSV initialization, widget menu (preview/add), Supabase table queries, and pin-to-dashboard rendering.

## System Benefits

Dimension	Typical Dashboard	CitlInsight
Setup	Backend required	Local CSV runs instantly
Queries	Manual filters/SQL	NL chat + constrained actions
Add Insights	Developer-only	User adds widgets via chatbot
Reliability	Free-form outputs	Catalog + read-only DB selects

## Roadmap

- Persist Layout:** Save pinned widgets + grid layout to Supabase per user/session.
- Richer Widget Builder:** Parameterized widgets (date range, station filters) while remaining constrained.
- Streaming Updates:** Supabase real-time subscriptions for near real-time feeds.
- Enrichment:** Weather/context signals (precipitation/temperature) to explain ridership changes.