LLM-Driven Agents for Traffic Signal Optimization

GitHub repo: https://github.com/NiemaAM/LLM-Driven-Agents-for-Traffic-Signal-Optimization/
Deployment: https://trafficllm.streamlit.app/

2. Milestone 2: Development of Proof-of-Concept

Milestone 2 establishes a functional baseline Proof-of-Concept (PoC) for the LLM-driven traffic signal optimization system. The objective is to translate the rule-based conflict detection engine introduced by Masri et al. (2025) into an end-to-end interactive application that demonstrates the core pipeline: vehicle scenario ingestion, conflict detection, priority assignment, wait-time computation, and animated visualization. The PoC serves as the empirical foundation from which subsequent milestones will introduce fine-tuned LLM classification and real-time monitoring.

2.1. Model Integration

The baseline model integrated in this milestone is the rule-based conflict detection system published by Masri et al. (2025) in their paper Large Language Models (LLMs) as Traffic Control Systems at Urban Intersections: A New Paradigm (arXiv:2411.10869). The original repository was adopted as the detection engine and embedded directly into the application's source module at src/conflict_detection.py.

Intersection Layout

The engine operates on a fixed four-arm intersection defined in data/intersection_layout.json. Each arm carries two lanes, yielding eight lanes in total numbered 1 through 8. Odd-numbered lanes (1, 3, 5, 7) are entry lanes carrying vehicles toward the intersection; even-numbered lanes (2, 4, 6, 8) include both entry and dedicated exit lanes. Each lane maps to a subset of eight exit destinations labelled A through H.

Direction	Lane	Type	Valid Destinations
North	1	Entry (right / straight)	F, H
North	2	Exit ←	E, D, C
East	3	Exit ←	H, B
East	4	Entry (left)	G, E, F
South	5	Entry (right / straight)	B, D
South	6	Exit ←	A, G, H
West	7	Exit ←	D, F
West	8	Entry (left)	B, C, A

Detection Pipeline

The conflict detection function detect_conflicts() iterates over every pair of vehicles in a scenario and applies the following four-stage logic:

- **Path intersection check:** Movement types (straight, left turn, right turn) are derived from each vehicle's lane and destination. Perpendicular straights conflict; opposite straights do

not. All left-turn combinations are treated as conflicting unless both vehicles approach from opposite directions.

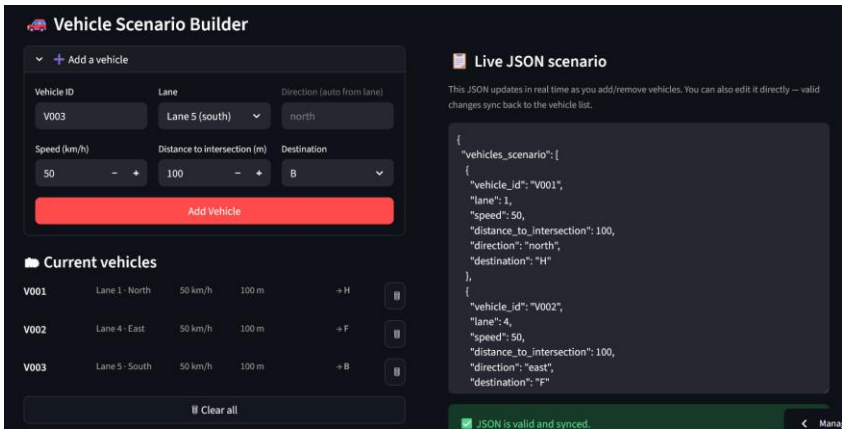
- **Arrival-time proximity:** A conflict is only raised if both vehicles reach the stop-line within a configurable threshold (default 4 seconds). Pairs separated by more than this window are assigned no conflict.
- **Priority assignment:** Three nested rules determine which vehicle must yield: (1) straight movement takes priority over any turn; (2) right turn takes priority over left turn; (3) when movement types are equal, the right-hand rule applies — the vehicle approaching from the right of the other has priority.
- **Wait-time computation:** The lower-priority vehicle is assigned a wait time equal to max(0, TTA_priority + 2s − TTA_yielding), where 2 seconds represents the estimated intersection traversal time.

The function returns a list of conflict dictionaries, each containing: vehicle1_id, vehicle2_id, decision (human-readable string), place, priority_order (mapping vehicle IDs to rank 1 or 2), and waiting_times (mapping vehicle IDs to seconds).

2.2. App Development

The application is built with Streamlit and is deployed publicly at trafficllm.streamlit.app. The codebase follows the repository's existing structure, placing the main entrypoint at Streamlit/app.py and the visualization module at src/visualization.py.

Vehicle Scenario Builder



The original raw-JSON text area was replaced with a structured form that guides users through scenario construction without requiring knowledge of the JSON schema. The form exposes six fields per vehicle:

- **vehicle_id:** Free text with auto-suggestion (V001, V002, …). Duplicate IDs are rejected on submission.
- **lane:** Dropdown listing lanes 1–8, each labelled with its associated direction (e.g. "Lane 1 (north)").
- **direction:** Read-only field, automatically resolved from the selected lane — no manual entry required.

- **destination:** Dropdown dynamically populated from intersection_layout.json for the chosen lane.
- **speed:** Numeric input in km/h (range 1–200).
- **distance_to_intersection:** Numeric input in metres (range 1–5000).

The vehicle list and a live JSON panel are displayed side by side. Every addition or deletion immediately rewrites the JSON panel via session state. Conversely, any direct edit to the JSON panel is parsed in real time via an on_change callback: if valid, the vehicle list updates accordingly; if invalid, a red warning is shown inline while the previous state is preserved. The Detect Conflicts button always reads from the live JSON text, ensuring that direct JSON edits are never silently discarded.

Visualization Module

The module src/visualization.py implements two animated Plotly figures rendered inline via st.plotly_chart. Both figures share a common dark-themed top-down intersection layout that draws the four road arms with lane dividers, stop-line markers, lane number badges (1–8), directional arrows, destination labels (A–H), and compass indicators.

- **Problem view (visualize_intersection):** Runs detect_conflicts() internally to identify conflicting pairs, then renders the scenario with no wait times applied. Conflicting vehicle pairs are connected by a red dashed line through the intersection centre, making the collision risk immediately visible.
- **Solution view (visualize_solution):** Replays the scenario with the waiting_times values from detect_conflicts() applied. Lower-priority vehicles pause at the stop-line for their assigned duration before crossing, visually demonstrating the resolution. A summary table below the chart shows each conflict pair, their priority (first / yield), and wait time in seconds.

Speed-Aware Animation

Vehicle motion is driven by real physics rather than fixed timeline fractions. The total simulated duration is computed as:

total_sim_time = max over all vehicles of (TTA + wait + 4 s crossing)

Each vehicle's normalised timeline is then: approach_end = TTA / total_sim_time and wait_end = (TTA + wait) / total_sim_time. A vehicle travelling 100 m at 100 km/h (TTA = 3.6 s) therefore reaches its stop-line much earlier in the animation than one covering the same distance at 30 km/h (TTA = 12 s). This ensures that the moment of simultaneous stop-line arrival — the precise instant of conflict — is rendered faithfully.

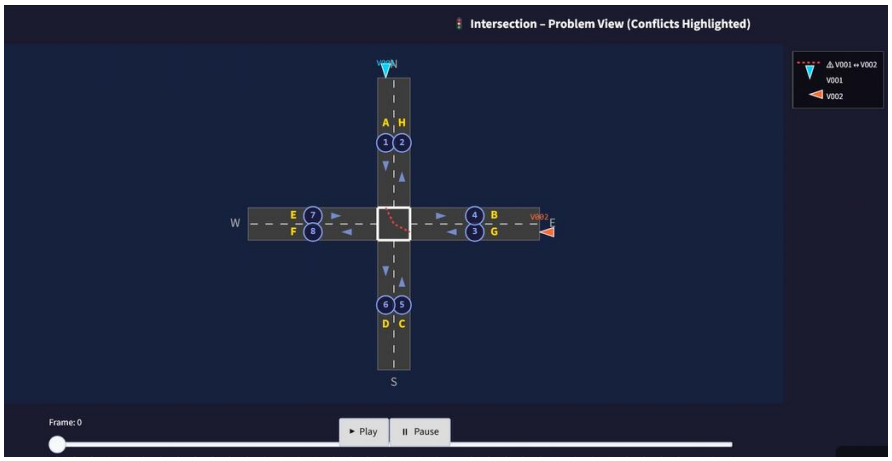
2.3. End-to-End Scenario Testing

Three representative scenarios were executed through the complete pipeline to validate correctness of the detection engine, the wait-time computation, and the visual output.

Scenario 1: Classic Perpendicular Conflict (Default)

This is the default scenario loaded by the application. Two vehicles approach from perpendicular directions at identical speeds and distances.

Vehicle	Lane	Direction	Destination	Speed (km/h)	Distance (m)
V001	1	North	F (straight)	50	100
V002	3	East	B (straight)	50	100

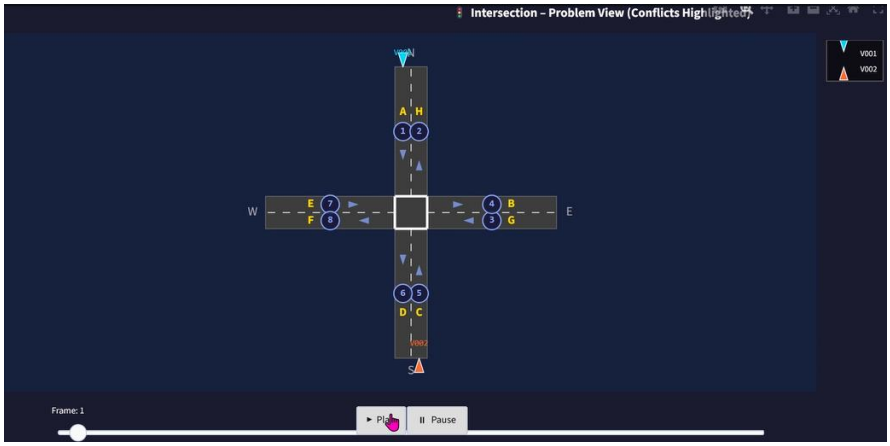


Both vehicles have TTA = 7.2 s and movement type straight. Their paths cross (perpendicular straights). The right-hand rule assigns priority to V002 (east is to the right of north). V001 is assigned wait = max(0, 7.2 + 2 − 7.2) = 2 s. In the solution view, V001 pauses at the stop-line for 2 s while V002 crosses, then proceeds.

Scenario 2: Speed Differential – No Conflict

This scenario validates the arrival-time threshold: a fast vehicle and a slow vehicle on crossing paths do not generate a conflict if their arrival times differ by more than 4 seconds.

Vehicle	Lane	Direction	Destination	Speed (km/h)	Distance (m)
V001	1	North	F	120	100
V002	5	South	D	20	100

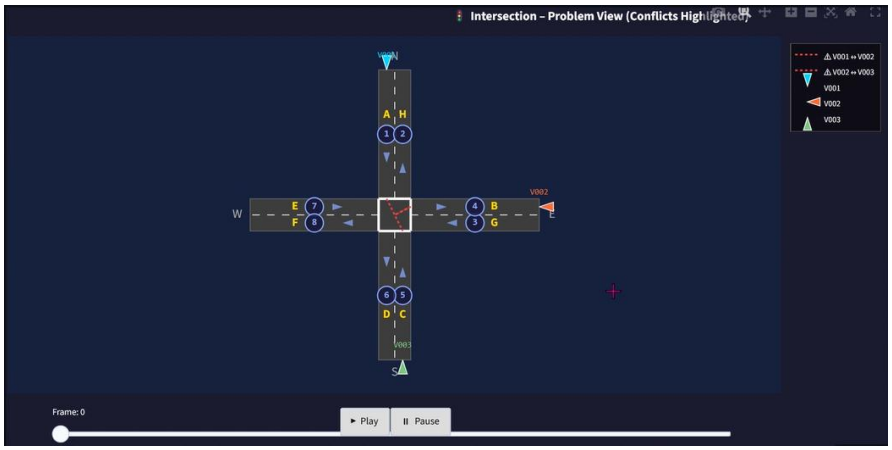


V001 TTA = 3.0 s; V002 TTA = 18.0 s. Time difference = 15 s, which exceeds the 4 s threshold. Result: no conflict detected. The problem view displays both vehicles with no red connector line, and the solution view shows both crossing without waiting. The animation faithfully shows V001 clearing the box long before V002 arrives.

Scenario 3: Multi-Vehicle Priority Chain

Three vehicles approach simultaneously, creating two independent conflict pairs. This tests the engine's ability to handle multiple overlapping conflicts and correctly aggregate wait times per vehicle.

Vehicle	Lane	Direction	Destination	Speed (km/h)	Distance (m)
V001	1	North	H (right)	50	100
V002	4	East	F (left)	50	100
V003	5	South	B (right)	50	100



Two conflicts are detected: V001 vs V002 (right turn vs left turn → V002 yields, wait = 2 s) and V002 vs V003 (left turn vs right turn from adjacent direction → V002 yields again, wait = 2 s). V002's aggregated wait time across both conflicts is 2 s (maximum). In the solution view, two red dashed connectors appear in the problem animation; in the solution view V002 waits while V001 and V003 cross first. The resolution table correctly shows V001 and V003 as first-priority and V002 as the yielding vehicle in both pairs.

Summary

All three scenarios produced outputs consistent with the traffic priority rules defined in the baseline model. The live JSON panel correctly reflected each scenario in real time, the animated visualizations rendered vehicle speeds proportionally, and the resolution logic applied wait times precisely as computed by detect_conflicts(). The PoC is deployed and accessible at trafficllm.streamlit.app, confirming successful end-to-end integration of the baseline model into a user-facing application.