EVOLVELIGHT: A SELF-EVOLVING AGENTIC FRAMEWORK FOR ADAPTIVE TRAFFIC SIGNAL CONTROL

Anonymous authors
Paper under double-blind review

ABSTRACT

Traffic signal control (TSC) is fundamental to urban mobility, yet traffic demand is spatially heterogeneous and continuously evolving, making it difficult for a single control algorithm to sustain stable performance across regions and control periods. Existing approaches primarily improve adaptability through predefined control principles, specialized policy architectures, or task-specific training, limiting their adaptability in evolving traffic dynamics. We present *EvolveLight*, an LLM-based agentic framework that enables continual adaptive TSC through a closed loop of traffic diagnosis, controller construction, feedback, and skill development. EvolveLight coordinates three specialized agents: a planning agent that analyzes traffic dynamics, partitions the road network into spatiotemporally coupled regions, and assigns suitable control strategies; a modeling agent that constructs region-specific control policies; and a skill agent that develops reusable traffic analysis and control policy development skills. These skills are maintained in a persistent versioned skill library. Their utility is evaluated through downstream traffic improvements and fed back into future retrieval and refinement. To mutually improve the reasoning capabilities of the three agents, we introduce a decoupled cooperative multi-agent RL (DCMA-RL) mechanism. It starts from supervised cold-start training with rejection-sampled high-quality demonstrations. Afterwards, DCMA-RL alternates role-wise RL optimization with independently normalized cooperative and role-specific rewards, enabling stable specialization while incorporating delayed cooperative downstream TSC performance feedback into agent learning. Extensive experiments across five real-world traffic datasets demonstrate better or competitive improvements over existing baselines, with superior performance during 24-hour adaptation as experience and reusable skills accumulate. Our project is available at `https://anonymous.4open.science/r/EvolveLight-8341/`.

1 INTRODUCTION

Traffic signal control (TSC) is a fundamental urban-computing problem, as ineffective signal decisions increase delay, trigger queue spillback, and induce greenhouse gas emissions. Although modern sensing infrastructure provides detailed and fine-grained traffic observations, reliably translating them into network-level control remains challenging. Traffic demand is spatially heterogeneous across intersections and regions and continuously evolves over time. Consequently, a control algorithm that performs well under one traffic regime may degrade as demand shifts, often requiring manual redesign, architectural modification, or retraining.

Existing TSC methods improve adaptability through several paradigms. Signal-timing optimization derives schedules from historical demand (Koonce et al., 2008), while MaxPressure reacts to online queues (Varaiya, 2013). However, both of these methods rely on predefined rules, which are restrictive under evolving conditions. RL methods learn policies through environment interaction (Wei et al., 2021; 2019a;b), but require retraining or architectural redesign for new networks or scenarios. LLM-based LLMLight and CoLLMLight make interpretable online decisions (Lai et al., 2025; Yuan et al., 2026) but incur substantial roadside inference overhead. SignalClaw and EvolveSignal propose to generate executable phase-scoring programs and timing plans (Lei et al., 2026; Wang et al., 2025a) but they focus on improving individual types of algorithms rather than adaptation

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

across regions and periods. Overall, existing methods rely on predefined rules, model architectures, or retraining, providing limited support for continual online adaptability to evolving traffic.

Continual online adaptation introduces two fundamental challenges. First, adaptation under spatiotemporally heterogeneous traffic conditions requires the system to determine when, where, and how its control strategy should evolve. Traffic variations may be transient or persistent, localized or network-wide. The system must therefore diagnose evolving traffic dynamics and assign appropriate controllers rather than relying on a homogeneous network-wide strategy. Second, effective self-evolution requires the system to learn and retain traffic-analysis and algorithm-development capabilities that consistently improve downstream traffic efficiency. However, the utility of such procedures is often indirect and delayed, becoming apparent only through subsequent reasoning and TSC performance evaluation. Therefore, a key challenge is how to incorporate delayed performance feedback into agent learning, enabling specialized agents to progressively improve their development capabilities based on long-horizon environmental interactions.

To address these challenges, we present *EvolveLight*, an LLM-based agentic framework for continual traffic signal adaptive control under dynamic traffic conditions. EvolveLight coordinates three specialized agents: a planning agent that analyzes evolving traffic dynamics, partitions the network into spatiotemporally coupled regions, and assigns suitable control strategies; a modeling agent that constructs region-specific algorithms; and a skill agent that develops reusable traffic-analysis and algorithm-modeling skills. These skills reside in a persistent, versioned skill library. Their utility is evaluated through downstream traffic improvements and fed back into future retrieval and refinement. By transforming performance feedback into reusable development knowledge, EvolveLight continually improves its adaptive control capability over long-horizon operation.

To optimize this workflow while preserving agent role specialization, we introduce decoupled cooperative multi-agent RL (DCMA-RL). Supervised cold-start training first establishes reliable role-specific behaviors from demonstrations filtered by execution validity and downstream performance. DCMA-RL then alternates RL optimization across three agents. At each step, one target agent is optimized while the others remain frozen but still participate in end-to-end rollouts, preserving cooperative context. Each agent receives two complementary rewards: a shared cooperative reward reflecting downstream traffic improvement and a role-specific reward evaluating its intermediate output artifacts (*e.g.*control strategy, algorithm, or skill). By independently normalizing two rewards before advantage aggregation, DCMA-RL provides stable role-level supervision while separating delayed feedback into agent learning, enabling specialization and coordinated self-evolution.

Our contributions are threefold. (1) We introduce *EvolveLight*, to our knowledge the first LLM-based agentic framework for online adaptive traffic signal control. (2) We propose a multi-agent architecture jointly supporting network-level strategy planning, region-specific algorithm development, and reusable skill accumulation. We further introduce DCMA-RL, which alternates role-wise optimization and decouples cooperative from role-specific rewards for stable agent specialization and coordination training. (3) Extensive experiments on real-world networks demonstrate improved or competitive performance over existing methods and superior adaptability during 24-hour self-evolving adaptation as experience and reusable skills accumulate.

2 PRELIMINARIES

Definition 1 (Road Network)**.** A road network is a directed graph $\mathcal{G} = (\mathcal{I}, \mathcal{L})$, where $\mathcal{I}$ is the set of intersections and $\mathcal{L}$ is the set of directed lanes. Each lane $l \in \mathcal{L}$ connects an upstream intersection to a downstream intersection with either go-through, left-turn, or right-turn movement.

Definition 2 (Traffic Signal)**.** At each signal-switching step t, the intersection i selects a phase $a_{i,t}$ from a predefined set $\mathcal{A}_i = \{a_i^1, \ldots, a_i^n\}$. A phase is represented by the set of lanes granted right of way, *i.e.*, $a_i^j = \mathrm{set}(\mathcal{L}_{\mathrm{allow}})$, where the lanes in $\mathcal{L}_{\mathrm{allow}}$ are pairwise non-conflicting. The vehicles on $\mathcal{L}_{\mathrm{allow}}$ receive a green signal, while other incoming lanes receive a red signal.

Definition 3 (Traffic Signal Control)**.** The traffic signal control is a partially observable Markov decision process (POMDP) defined by the tuple $\langle \mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{R} \rangle$, where:

- **State**: $\mathcal{S}$ is the state space, the local state $s_{i,t}$ of intersection i at time t, including queue length, approaching-vehicle count, waiting time, average speed, density, the current signal phase, *etc.*

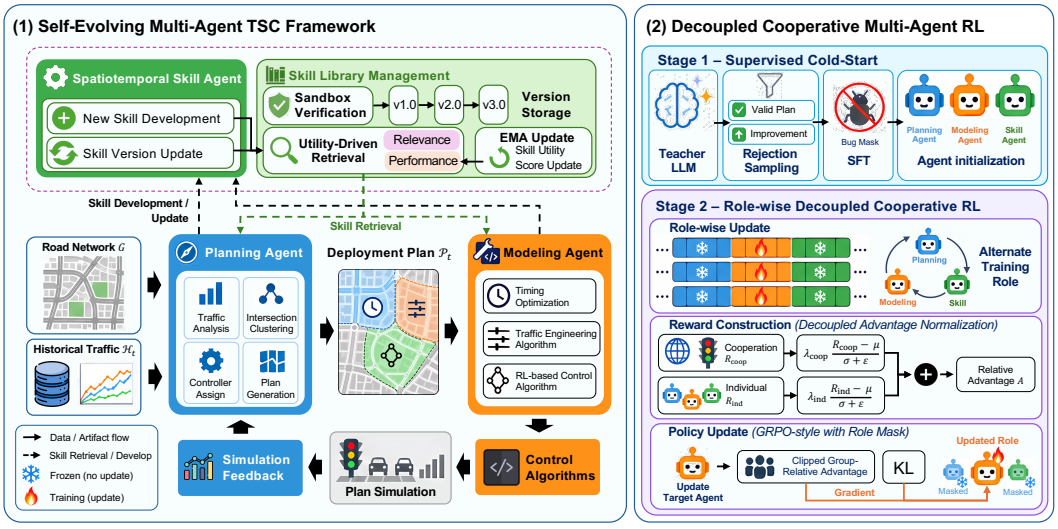

Figure 1: The overview of EvolveLight framework.

- **Observation**: $\mathcal{O}$ states the observation space of the intersection controller. $o_{i,t} \subseteq s_{i,t}$ is the observation of intersection i at time t.
- **Action**: $\mathcal{A}$ is the action space. $a_{i,t} \in \mathcal{A}$ is the signal phase executed by intersection i at time t.
- **Reward Function**: $R(\cdot)$ is the reward function. $r_{i,t}$ is the reward after executing action $a_{i,t}$.

Definition 4 (Online Adaptive Traffic Signal Control)**.** We divide the operational horizon into control time windows indexed by t. At t, the agent observes historical traffic dynamics $\mathcal{H}_t = \{s_{i,1}, \ldots, s_{i,t}\}_{i \in \mathcal{I}}$ and the road network $\mathcal{G}$. It updates a control strategy:

$$\mathcal{C}_t = \{(c_i, \pi_i)\}_{i \in \mathcal{I}}, \tag{1}$$

where c_i specifies the control method assigned to intersection i (*e.g.*, timing-based, traffic-engineering, or RL-based), and π_i is the corresponding control policy. The agent aims to maximize the traffic efficiency $\mathbb{E}\left[\sum_{t \in T} R_{\text{eff}}^{\text{net}}(s_t)\right]$, where T is the full operational horizon, and $R_{\text{eff}}^{\text{net}}(s_t)$ measures network-wide traffic efficiency at time t (*e.g.*, travel time or queue length).

3 Method

As illustrated in Figure 1, EvolveLight forms a closed-loop framework for online adaptive TSC: (1) EvolveLight coordinates three agents collaboratively perform traffic diagnosis, control policy development, and reusable skill evolution through a versioned skill library; (2) DCMA-RL optimizes this workflow via role-wise RL and coordinated downstream traffic feedback for stable self-evolution.

3.1 EvolveLight Framework

Planning Agent. The planning agent maps network-level traffic conditions to control strategy assignments. Its inputs include the road network $\mathcal{G}$, historical traffic observations $\mathcal{H}_t$, and the incumbent control strategy from the previous period. Its reasoning comprises four types of processes:

- **Spatiotemporal Traffic Analysis**: identifies persistent temporal changes in demand and spatial dependencies in congestion propagation across intersections;
- **Skill Retrieval**: retrieves relevant skills from the skill library to support subsequent reasoning;
- **Skill Development**: invokes the skill agent to construct skills required by subsequent reasoning;
- **Control Strategy Generation**: partitions intersections in the road network into regions $\mathcal{Z}_t = \{z_1, \ldots, z_n\}$ and assigns an appropriate control strategy to each region.

The resulting control strategy for the next period is defined as:

$$\mathcal{C}_t = (z_j, c_j, q_j)_{j=1}^{n} = f_{\text{plan}}(\mathcal{G}, \mathcal{H}_t, U), \tag{2}$$

where c_j specifies the control method assigned to intersections in region z_j, including timing-based, traffic-engineering, or RL-based methods, q_j describes the development specification (*e.g.*the proposed control strategy, modeling requirements, and revisions to issues from the previous control

strategy), f_{plan} is the planning process of the agent, and U_t is a set of retrieved skills. The agent iteratively selects and executes these reasoning processes until a valid control strategy is produced. This generated strategy is then passed to the modeling agent for specific control policy development.

Modeling Agent. Given the control strategy $\mathcal{C}_t$, the modeling agent realizes the control policy for each region. In addition to spatiotemporal data analysis, skill retrieval, and skill development, its reasoning includes three modeling processes:

- **Timing Policy Generation**: constructs timing-based control policies by determining phase sequences, green durations, and cycle lengths according to the historical traffic pattern;
- **Transportation-Based Policy Generation**: develops executable transportation-based control policies by configuring demand-based control rules and their associated parameters;
- **RL-Based Policy Generation**: constructs deep RL models by designing the observation representation, spatiotemporal neural network modules, and RL paradigms.

For each region, the selected modeling process produces control policies:

$$\pi_i = f_{\text{model}}^{c_i}\left(z_i, q_i, \mathcal{H}_t, U\right), \tag{3}$$

where $f_{\text{model}}^{c_i}$ denotes the modeling process of control method c_i, U denotes the retrieved skills used during policy construction, and π_i is the modeling agent's constructed control policy at intersection i for the next period. The constructed policies are executed through a simulator using recent historical traffic flows. The execution results are returned to the planning agent for diagnosis.

Spatiotemporal Skill Agent. The spatiotemporal skill agent is a reactive capability constructor invoked by the planning or modeling agent. Caller agents can issue a skill development or update request q_{skill} that specifies the required capability, interface requirements, task context, and the execution feedback of a previous skill. The skill agent supports two reasoning processes:

- **Skill Development**: creates a new spatiotemporal skill when the retrieved skills cannot meet the current planning or modeling requirement;
- **Skill Update**: revises a retrieved skill from its current version when the caller requires new functionality for self-evolving or identifies incorrect behavior and missing capability.

The skill agent produces or updates the skill artifact according to the selected reasoning process:

$$\tilde{u} = \begin{cases} f_{\text{dev}}(q_{\text{skill}}, \mathcal{H}_t), & a_{\text{skill}} = \texttt{skill_development}, \\ f_{\text{upd}}(q_{\text{skill}}, u, \mathcal{H}_t), & a_{\text{skill}} = \texttt{skill_update}, \end{cases} \tag{4}$$

where u denotes the retrieved parent skill, and f_{dev} and f_{upd} are the spatiotemporal skill development and update processes, respectively. Each generated or updated skill is stored in the spatiotemporal skill library with explicit version lineage. This versioned management enables skills to gradually evolve with emerging planning and modeling requirements and traffic dynamics, allowing reusable capability accumulation in long-term traffic signal control.

3.2 Spatiotemporal Skill Library Management

Skill Representation and Versioned Storage. The spatiotemporal skill library $\mathcal{U}$ serves as EvolveLight's persistent non-parametric memory, accumulating and evolving reusable capabilities across consecutive periods. Each skill is stored as an executable artifact:

$$u = \left[\texttt{Desc}, \texttt{Doc}, \texttt{IO}, \texttt{Ver}, \texttt{Code}, \texttt{Score}\right], \tag{5}$$

where `Desc` provides a functional description, `Doc` specifies usage instructions and invocation examples, `IO` defines typed input–output interfaces, `Ver` records the version lineage, `Code` contains the executable implementation, and $\texttt{Score} \in [-1, 1]$ is a performance-grounded utility score.

Upon a skill request q_{skill}, the caller first retrieves relevant skills from $\mathcal{U}$. If no candidate satisfies the requirement, the skill agent is invoked with `skill_development`. For a retrieved skill that requires additional functionality or has failed during execution, the skill agent is invoked with `skill_update`. The new skill artifact is checked via sandbox validation. Development creates a new entry with an initial version, whereas update appends an immutable child version to its parent:

$$\tilde{\mathcal{U}} = \begin{cases} \mathcal{U} \cup \{\tilde{u}^{(1)}\}, & a_{\text{skill}} = \texttt{skill_development}, \\ \text{Append}(\mathcal{U}, \tilde{u}^{(v+1)}), & a_{\text{skill}} = \texttt{skill_update}. \end{cases} \tag{6}$$

where $\tilde{u}^{(v+1)}$ denotes the child of the parent version $u^{(v)}$.

Self-Evolving Utility Scoring. Rather than treating all retrieved skills equally, EvolveLight tracks the demonstrated downstream utility of each skill. After each training-and-simulation round, we compare the resulting scalar downstream traffic optimization performance R_{eff} (*e.g.*network-wide travel time) with a reference performance $\tilde{R}_{\text{eff}}$ obtained with the previous skill version:

$$r = \text{clip}\left(\frac{\tilde{R}_{\text{eff}} - R_{\text{eff}}}{\gamma \tilde{R}_{\text{eff}}}, -1, 1\right), \tag{7}$$

where γ controls the reward scale. This reward is credited to every skill invoked in the round, and the utility score is updated through an exponential moving average (EMA) with learning rate α:

$$\texttt{Score}(u) \leftarrow (1-\alpha)\texttt{Score}(u) + \alpha r, \tag{8}$$

The EMA is initialized with the first observed reward. It smooths stochastic traffic fluctuations while allowing utility estimates to co-evolve with changing traffic conditions and the improving agents.

Utility-Aware Retrieval. For retrieval, semantic relevance remains the primary criterion. Let $\rho(q_{\text{skill}}, u)$ denote the relevance between the request and a skill description, and $\rho_{\max}$ the maximum relevance among candidates. Skills are retrieved and ranked by:

$$\begin{aligned} U &= \text{TopK}_{u\in\mathcal{U}}\left[\rho(q_{\text{skill}}, u) + \mathbb{1}[\rho(q_{\text{skill}}, u) \geq \rho_{\max} - \delta]\,\widehat{\texttt{Score}}(u)\right], \\ \text{where} \quad \widehat{\texttt{Score}}(u) &= \texttt{Score}(u) + b_{\text{fresh}}\mathbb{1}[n(u)=0] + \frac{c}{\sqrt{n(u)+1}}, \end{aligned} \tag{9}$$

where $\widehat{\texttt{Score}}(u)$ is the updated utility score of skill u. The adjusted utility score augments the accumulated utility $\texttt{Score}(u)$ with a freshness bonus b_{fresh} for newly developed skills and an exploration bonus controlled by c. $n(u)$ denotes the number of previous executions of skill u. The relevance margin δ ensures that utility only promotes semantically relevant candidates, while the exploration terms allow newly developed and under-evaluated skills to accumulate sufficient evidence before selection. Together, self-evolving utility scoring and utility-aware retrieval close the library evolution loop. The skills that repeatedly improve downstream traffic performance will be promoted with stronger retrieval preference and richer feedback.

3.3 Decoupled Cooperative Multi-Agent RL

Supervised cold-start. Direct RL from generalist LLMs is unstable as their experience in traffic signal control is limited, leading to insufficient reasoning and modeling capabilities and often resulting in invalid artifacts (*i.e.*control strategies, control policies, or skills). We therefore initialize each agent with supervised cold-start. We collect demonstrations from a frontier LLM (*e.g.*DeepSeek-V4) and retain high-quality trajectories through rejection sampling:

$$\mathcal{D}^+ = \{\tau \in \mathcal{D} \mid \text{Valid}(\tau) = 1,\ R_{\text{eff}}(\tau) > R_{\text{prv}}\}, \quad \mathcal{D} = \{\tau_1, \ldots, \tau_N\}, \tag{10}$$

where τ is a collected trajectory, $\text{Valid}(\tau)$ requires schema and execution validation, and R_{prv} denotes the performance of the previously deployed control strategies. During SFT, we apply a bug mask $m_t \in \{0, 1\}$ to exclude interaction turns with intermediate artifacts or execution failures:

$$\mathcal{L}_{\text{SFT}} = -\sum_{\tau\in\mathcal{D}^+} m_t \sum_{i=1}^{|y_t|} \log \pi_\theta(y_{t,i} \mid y_{t,<i}, s_t), \tag{11}$$

This initialization provides reliable role-specific capabilities before cooperative optimization.

Role-wise decoupled cooperative RL. The traffic efficiency improvement is jointly influenced by the cooperative reasoning and decision-making of the three agents. Directly optimizing all agents simultaneously introduces non-stationarity, as updates of one agent alter the interaction contexts and entangle role-specific contributions. Therefore, we optimize agents in a role-wise manner. We sample a joint group of trajectories from different roles of agents within the same context and evaluate complete interaction trajectories. For each trajectory τ, it receives a shared cooperation reward:

$$R^{\tau}_{\text{coop}} = \frac{R_{\text{eff}}(\mathcal{D}_{\text{prv}}) - R_{\text{eff}}(\widehat{\mathcal{D}}_i)}{R_{\text{eff}}(\mathcal{D}_{\text{prv}}) + \epsilon}, \tag{12}$$

where $R_{\text{eff}}(\cdot)$ indicates traffic efficiency (*e.g.*network-wide travel time), and ϵ is a small constant to avoid zero division. This reward provides global cooperative signals for all agents. Meanwhile, a role-specific reward $R^{\tau}_{\text{ind},k}$ evaluates the role-specific capabilities, including validity of control strategy for planning, control policy for modeling, and skill utility for skill development.

Inspired by GDPO (Liu et al., 2026b), to decouple the cooperative and role-specific optimization signals, we independently normalize the two rewards and aggregate them into a single advantage:

$$A^{\tau}_{k} = \lambda_{\text{coop}} \frac{R^{\tau}_{\text{coop}} - \mu_{\text{coop},k}}{\sigma_{\text{coop},k} + \epsilon} + \lambda_{\text{ind}} \frac{R^{\tau}_{\text{ind},k} - \mu_{\text{ind},k}}{\sigma_{\text{ind},k} + \epsilon}, \tag{13}$$

where k indexes the target role, λ_{coop} and λ_{ind} are the coefficients for the cooperative and role-specific rewards, respectively, and $\mu_{\cdot,k}$ and $\sigma_{\cdot,k}$ are the group mean and standard deviation of the corresponding reward. The resulting advantage provides localized role-specific supervision while simulating cooperative reasoning and decision-making for downstream task on traffic optimization.

To update only the target role while preserving inter-agent dependencies, we freeze the remaining agents during rollout. Let $\mathcal{T}^{i}_{k}$ denote the token positions generated by the target role in trajectory i, and define a token mask $m_{k,i,t} = \mathbb{I}[t \in \mathcal{T}^{i}_{k}]$. Only target-role tokens contribute to RL optimization:

$$\mathcal{L}_k = -\mathbb{E}_{i,t}\big[\, m_{k,i,t} \min\left(\rho_{i,t} A_{k,i}, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon) A_{k,i}\right) \big] + \beta \text{KL}\big(\pi_{\theta_k} \| \pi_{\text{ref},k}\big), \tag{14}$$

where $\rho_{i,t}(\theta_k)$ denotes the importance ratio of target-role tokens. This alternating role-wise optimization freezes non-target agents while retaining their contexts in rollout, preserving cooperation and stabilizing specialist learning. Cycling optimization across the three roles creates a stable learning process for each specialist while retaining cooperative dependencies.

4 Experiments

4.1 Experimental Setup

Datasets. We evaluate EvolveLight on five real-world traffic flow datasets: two from Jinan, China, one from Hangzhou, China, one from New York, USA, and one from Guangzhou, China. The New York dataset covers a different region and traffic patterns. The Guangzhou dataset spans 24 hours and is used to evaluate long-term optimization and continual adaptation under evolving traffic conditions. Detailed statistics are summarized in Table 1.

Table 1: Statistics of datasets.

Dataset	No. Inter.	No. Veh.	Arrival rate (vehicles/5 min)			
			Mean	Std	Max	Min
Jinan 1	12	4,365	363.8	74.7	493	237
Jinan 2	12	5,494	457.8	46.2	544	363
Hangzhou	16	6,984	582.0	318.5	1,146	203
New York	48	2,824	235.3	5.6	244	224
Guangzhou	44	17,326	60.2	29.2	166	15

Environment Settings. We use SUMO (Alvarez Lopez et al., 2026), a widely adopted traffic simulator in TSC research, as the simulation environment. SUMO provides comprehensive traffic observation interfaces and signal control APIs for agent-environment interaction. Right-turn movements are always permitted. Each signal phase consists of a 15-second green interval, followed by a 3-second yellow interval and a 2-second all-red interval. We simulate traffic scenarios for one hour on Jinan, Hangzhou, and New York networks. Guangzhou dataset spans 24 hours for simulation, where each control time window spans six hours.

Baselines. We compare EvolveLight with transportation-based methods, RL-based controllers, and LLM-based agents. Transportation-based methods include FixedTime and MaxPressure (Varaiya, 2013). RL-based methods include MPLight (Chen et al., 2020), CoLight (Wei et al., 2019b), Efficient-MPLight and Efficient-CoLight (Wu et al., 2021), and CityLight (Zeng et al., 2025). LLM-based agents include LLMLight (Lai et al., 2025) and CoLLMLight (Yuan et al., 2026). To provide a comparison with frontier LLMs in TSC, we evaluate DeepSeek-V4-Flash, GLM-5.3, and Qwen3.6-27B with our EvolveLight agent framework.

Skill Library Initialization. The skill library is initialized using the post-trained EvolveLight agents through simulated development interactions. Specifically, we use the Jinan, Hangzhou, and New York datasets as initialization environments. For each dataset, the EvolveLight agents interact with the traffic environment for 20 development rounds, during which newly developed and updated skills are accumulated in the library. After initialization, the skill library contains 43 distinct skills with 76 total versions, which are used as the starting capability pool for subsequent evaluations.

Table 2: Traffic control adaptability comparison. Lower ATT, AQL, and AWT are better, while higher TP is better. The best and second-best results are highlighted through boldface and underline.

Method	Jinan 1				Jinan 2				Hangzhou				New York			
	ATT	TP	AQL	AWT	ATT	TP	AQL	AWT	ATT	TP	AQL	AWT	ATT	TP	AQL	AWT
Transportation Methods																
FixedTime	516.23	4688	382.74	236.17	433.13	3935	183.88	151.77	518.85	2918	269.81	228.34	1143.63	1031	653.44	1008.86
MaxPressure	327.75	5697	112.04	64.23	310.13	4117	53.36	44.14	352.57	4058	76.97	54.02	1026.97	1110	602.77	887.92
RL Methods																
MPLight	449.53	4977	350.15	204.45	323.13	4109	71.84	59.34	371.64	3966	114.03	79.90	252.93	2629	52.01	66.47
CoLight	324.39	5704	112.50	64.46	316.07	4108	63.86	52.77	366.18	3997	99.76	70.00	233.79	2643	38.50	49.23
Efficient-MPLight	582.39	4766	540.94	319.52	552.64	3554	358.77	295.90	412.74	3842	176.92	123.74	1422.73	771	858.15	1303.34
Efficient-CoLight	327.11	5680	113.43	64.99	311.53	4110	55.45	45.84	355.86	4059	83.15	58.34	269.20	2628	59.99	76.68
CityLight	340.77	5656	135.34	77.51	331.33	4078	78.77	65.06	380.65	3818	126.46	90.42	997.21	1143	568.32	859.40
LLM Agents																
LLMLight	318.60	5705	99.19	56.88	306.54	4127	49.34	40.86	345.97	4068	68.64	48.25	298.34	2604	75.49	96.53
CoLLMLight	323.54	5680	104.84	60.10	**305.09**	4121	48.64	40.24	**342.87**	4081	**64.24**	**45.24**	235.97	2647	37.33	47.82
EvolveLight (Qwen3.6-27B)	321.68	5690	102.92	59.00	309.30	4127	52.38	43.32	352.32	4048	76.78	53.58	241.93	**2649**	42.98	55.02
EvolveLight (DS-V4-Flash)	318.47	5711	99.64	57.12	308.42	4126	51.25	42.40	350.04	4044	73.87	51.86	238.30	2643	44.88	57.40
EvolveLight (GLM-5.3)	322.96	5693	105.63	60.58	308.45	4125	51.50	42.62	349.24	4063	71.27	50.09	268.48	2593	68.32	87.35
EvolveLight	**315.59**	**5728**	**97.73**	**56.04**	305.83	**4132**	**48.44**	**40.09**	344.38	**4101**	65.94	46.37	**232.05**	2643	**36.94**	**47.32**

Agentic RL Settings. To jointly improve modeling and skill development capabilities, we adopt a mixed training strategy with two stages. In the first stage, EvolveLight agents interact with traffic environments with full access to the initialized skill library. In the second stage, 75% of the available skills are randomly masked, encouraging agents to identify capability gaps and develop new skills through interaction. The weights of the cooperative downstream traffic reward and role-specific reward are set to 1.0 and 0.5, respectively. We use Qwen3.6-27B as the backbone LLM and perform post-training for all agents with LoRA-based (Hu et al., 2022) fine-tuning. The learning rate is set to 1×10^{-5} for 50 training steps, with LoRA rank and scaling factor configured as 64 and 128, respectively. The Jinan, Hangzhou, and New York datasets are used for training, while the Guangzhou dataset is left for transferability and self-evolving adaptation evaluation.

Evaluation Metrics. We adopt average travel time (ATT), throughput (TP), average waiting time (AWT), and average queue length (AQL) as evaluation metrics (Wei et al., 2021; Lai et al., 2025). ATT measures the average travel time for vehicles from origins to destinations. TP measures the number of vehicles arrived their destinations. AWT captures the average waiting time at intersections. AQL is the average queue length at intersections.

4.2 Overall Performance Comparison

We first evaluate the overall performance of EvolveLight against previous SOTA baselines across heterogeneous traffic regimes in Table 2. Jinan 1, Jinan 2, and Hangzhou represent networks with moderate-to-high demand, while New York is a large-scale network with lower and more stable demand. Previous methods exhibit considerable performance variations across scenarios. MaxPressure performs competitively on networks in China but degrades notably on New York, while spatially coordinated RL methods (*i.e.*CoLight and Efficient-CoLight) achieve better adaptability by modeling neighboring interactions. However, other RL methods remain sensitive to network characteristics.

LLM-based agents further improve cross-network adaptability through enhanced reasoning over traffic states. CoLLMLight achieves the best ATT on Jinan 2 and the best ATT, AQL, and AWT on Hangzhou, but requires repeated LLM inference during online control. In contrast, EvolveLight conducts LLM reasoning through control policy development with lightweight controllers, achieving the best or the second-best performance on all datasets. A detailed analysis of inference cost is in Appendix A.5. Moreover, EvolveLight consistently surpasses frontier LLMs, demonstrating that our post-training strategy further enhances agent specialization and downstream traffic optimization.

4.3 Self-Evolving Capability Comparison

Figure 2 and Table 3 evaluate continual adaptation over 24 deployment hours on the unseen Guangzhou network. After the initial demand peak, most baselines exhibit increasing ATT, AQL, and AWT as traffic patterns evolve, indicating limited adaptation beyond their original policies. In contrast, EvolveLight stabilizes congestion metrics after its initial adaptation period and achieves superior performance across ATT, TP and AWT. Moreover, its throughput during the afternoon peak exceeds the morning peak, demonstrating that accumulated deployment feedback and evolved

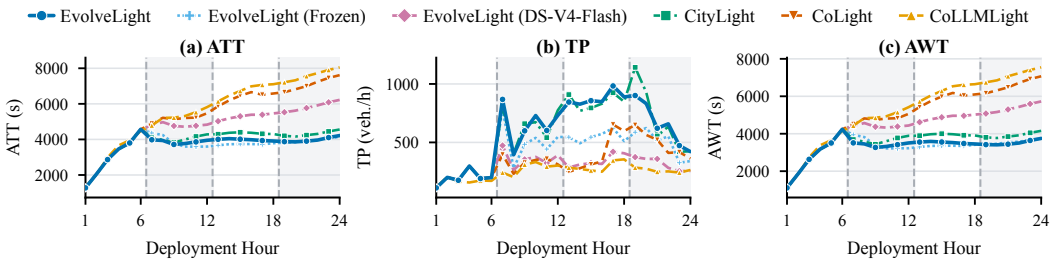

Figure 2: Hourly self-evolving performance on the Guangzhou dataset over 24 hours.

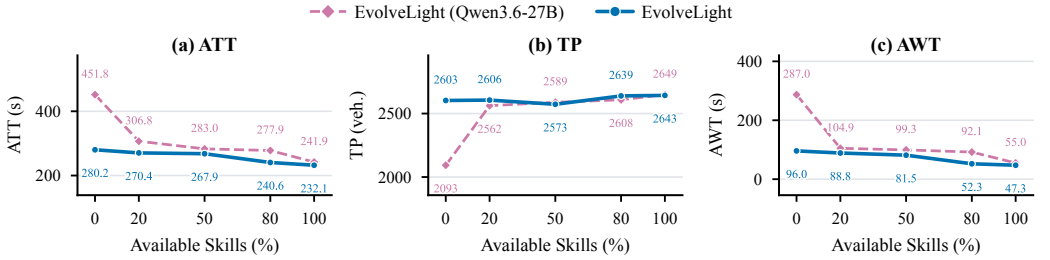

Figure 3: Traffic performance under different levels of initial Skill Library availability.

skills improve future control performance rather than merely maintain existing policies. CityLight achieves the best self-evolving baseline performance, which indicates that its connectivity-aware neighbor modeling gradually helps it adapt to the evolving traffic patterns in long-term deployment. However, its gap with EvolveLight is still significant (7.48% on ATT and 9.72% on AWT), indicating that the self-evolution solely with parametric updates is not enough in long-term adaptation.

We further evaluate a frozen variant of EvolveLight (EvolveLight-Frozen), which selects the best-performing deployment strategy and keeps it fixed throughout 24 hours. The frozen variant maintains strong overall performance, demonstrating that EvolveLight can generate high-quality controllers that generalize across changing traffic conditions. However, it achieves lower throughput than the evolving version, confirming the importance of continual adaptation during long-term deployment. Notably, we analyze the deployed strategies and find that EvolveLight maintains a heterogeneous controller portfolio, combining timing-based, transportation, and RL-based methods across different regions and periods. This shows that EvolveLight adaptively selects control strategies according to regional traffic characteristics rather than relying on a fixed controllers or uniformly deploying the most complex model. A detailed analysis of the strategy distribution is in Appendix A.6.

Table 3: Performance at the 24th deployment hour on Guangzhou. TP denotes the total number of arrived vehicles over 24 hours.

Method	ATT	TP	AQL	AWT
MaxPressure	9269.06	7,531	729.99	8696.43
CoLight	7614.10	8,502	625.17	7068.45
Efficient-CoLight	7958.57	7,771	585.36	7413.08
CityLight	4558.42	14,218	491.69	4153.86
LLMLight	7955.13	6,018	438.37	7463.27
CoLLMLight	8051.71	6,000	445.31	7544.92
EvolveLight (DS-V4-Flash)	6222.69	7,301	**338.60**	5725.40
EvolveLight	**4217.60**	**14,256**	442.56	**3750.14**

We further evaluate skill evolution under different levels of prior capability availability on the New York dataset by retaining 0%, 20%, 50%, 80%, or 100% of the initialized skill library in Figure 3. The untrained Qwen3.6-27B agent is highly sensitive to missing skills, with performance degrading substantially as library coverage decreases. In contrast, post-trained EvolveLight remains robustness with only up to 20% of performance degradation. These results indicate that our decoupled cooperative post-training not only enables the great development of downstream traffic efficiency but also enables the development of high-quality skills from scratch.

4.4 Ablation Studies

We ablate the post-training strategy on Jinan 1, Jinan 2, and Hangzhou, comparing EvolveLight with three variants: the Qwen3.6-27B backbone, supervised cold-start (SCS), multi-agent RL without

Table 4: Ablation study of post-training strategies.

Method	Jinan 1				Jinan 2				Hangzhou			
	ATT	TP	AQL	AWT	ATT	TP	AQL	AWT	ATT	TP	AQL	AWT
Qwen3.6-27B	321.68	5690	102.92	59.00	309.30	4127	52.38	43.32	352.32	4048	76.78	53.58
SCS	319.67	5710	100.37	57.55	308.39	4127	51.36	42.49	348.54	4098	71.15	49.59
W/O Decouple	320.02	5705	100.51	57.64	308.43	4127	51.38	42.51	347.81	4088	67.90	47.74
EvolveLight	**315.59**	**5728**	**97.73**	**56.04**	**305.83**	**4132**	**48.44**	**40.09**	**344.38**	**4101**	**65.94**	**46.37**

reward decoupling (W/O Decouple). As shown in Table 4, SCS consistently improves upon the untrained backbone, confirming the value of role-specific demonstrations. In contrast, W/O Decouple yields only marginal and inconsistent gains over SCS. It slightly improves the traffic on Hangzhou but degrades or matches SCS on Jinan 1 and Jinan 2. This suggests that jointly updating all agents provides noisy supervision and can entangle role-specific contributions. EvolveLight achieves the best result for every metric on all three datasets. Relative to the backbone, it reduces ATT, AQL, and AWT by up to 2.3%, 14.1%, and 13.5%, respectively, while increasing throughput by up to 1.3%. These consistent gains support our role-wise optimization design, which successfully separates and optimizes the cooperative downstream task performance and role-specific capabilities.

5 Related Work

Traffic Signal Control. Classical TSC optimizes fixed-time plans or applies traffic-responsive rules such as Max-Pressure (Varaiya, 2013). Deep RL instead learns control policies from environment feedback. PressLight incorporates pressure into reward, MPLight scales decentralized control, and CoLight coordinates intersections through graph attention (Wei et al., 2019a; Chen et al., 2020; Wei et al., 2019b). Recent LLM-based systems improve interpretability and flexibility through direct decision making or program synthesis (Lai et al., 2025; Yuan et al., 2026; Lei et al., 2026).

LLM Agents. Recent LLM agents increasingly automate data science and scientific modeling. DS-Agent develops ML solutions through case-based reasoning, MM-Agent structures real-world modeling workflows, and MLE-STAR combines web retrieval with targeted code refinement (Guo et al., 2024; Liu et al., 2026a; Nam et al., 2026). Recent skill-oriented agents further convert experience into reusable capabilities. For example, Agent Workflow Memory induces task routines, while SkillWeaver synthesizes and debugs executable skills (Wang et al., 2025b; Zheng et al., 2025). In urban computing, USTBench identifies weaknesses in long-horizon planning and reflection, UrbanDS organizes heterogeneous datasets as graph-linked skills, and TrafficClaw performs closed-loop control across mobility subsystems (Lai et al., 2026a; Zhou et al., 2026; Lai et al., 2026b).

Multi-Agent Post-Training. Multi-agent post-training learns coordination rather than relying on frozen-model prompting. MALT derives role-specific SFT/DPO data by propagating terminal rewards through heterogeneous reasoning trees (Motwani et al., 2024). MAPoRL jointly optimizes discussion agents with verifier-shaped rewards, while MAGRPO uses centralized group-relative advantages for multi-turn cooperation (Park et al., 2025; Liu et al., 2026c). MARFT formalizes asynchronous heterogeneous agents through a flexible Markov game (Liao et al., 2025). Stronger-MAS introduces agent- and turn-wise GRPO with global and role-specific rewards (Zhao et al., 2026).

6 Conclusion and Future Work

We presented *EvolveLight*, an LLM-based agentic framework that enables continual adaptive TSC through a closed loop of traffic diagnosis, control policy development, deployment feedback, and skill evolution. By coordinating planning, modeling, and skill agents, EvolveLight dynamically develops control strategies and accumulates reusable capabilities through a persistent utility-driven skill library. We further introduced DCMA-RL, which leverage supervised cold-start training and role-wise cooperative optimization to preserve agent specialization while incorporating delayed downstream traffic performance feedback into learning. Extensive experiments across five real-world traffic datasets demonstrate better or competitive performance over existing methods and superior adaptability during 24-hour self-evolving deployment as experience and reusable skills accumulate. Future work will investigate broader real-world deployment scenarios with additional operational constraints and more advanced lifelong skill learning mechanisms, further extending agentic self-evolution toward adaptive urban intelligence.

AI USE STATEMENT

In this work, we used generative AI tools solely to improve the clarity and coherence of author-written text. We did not use generative AI to formulate the research ideas or methods, conduct experiments, analyze data, produce results, or substantiate scientific claims. The authors reviewed and verified all AI-assisted revisions and take full responsibility for the final content of this work, including all text, claims, and artifacts.

ETHICS STATEMENT

This work studies traffic signal control in simulation and does not involve human participants or the collection of personally identifiable information. No personally identifiable information are included in the datasets. Improved traffic control may reduce congestion, travel delay, and associated emissions. However, deploying automatically developed controllers in real transportation systems could introduce safety, reliability, and distributional risks across road users and neighborhoods. Real-world deployment should undergo rigorous validation, monitoring, and approval by qualified transportation authorities, with appropriate human oversight and fail-safe control mechanisms.

REPRODUCIBILITY STATEMENT

To support reproducibility, we release the source code and post-trained LLM weights at `https://anonymous.4open.science/r/EvolveLight-8341/`. The paper specifies the simulation environment, traffic datasets, baseline methods, evaluation metrics, Skill Library initialization procedure, and agentic post-training configuration. It also reports the reward design, role-wise optimization procedure, and principal hyperparameters required to reproduce the experiments.

REFERENCES

Pablo Alvarez Lopez, Angelo Banse, Michael Behrisch, Jakob Erdmann, Yun-Pang Flötteröd, Robert Hilbrich, Ronald Nippold, and Peter Wagner. Simulation of urban mobility (sumo). Zenodo, version 1.26.0, January 2026. URL `https://zenodo.org/records/18406080`.

Chacha Chen, Hua Wei, Nan Xu, Guanjie Zheng, Ming Yang, Yuanhao Xiong, Kai Xu, and Zhenhui Li. Toward a thousand lights: Decentralized deep reinforcement learning for large-scale traffic signal control. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 3414–3421, 2020.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026. URL `https://arxiv.org/abs/2606.19348`.

GLM-5-Team. GLM-5: From vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026. URL `https://arxiv.org/abs/2602.15763`.

Siyuan Guo, Cheng Deng, Ying Wen, Hechang Chen, Yi Chang, and Jun Wang. Ds-agent: automated data science by empowering large language models with case-based reasoning. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=nZeVKeeFYf9`.

Peter Koonce et al. Traffic signal timing manual. Technical report, United States. Federal Highway Administration, 2008.

Siqi Lai, Zhao Xu, Weijia Zhang, Hao Liu, and Hui Xiong. Llmlight: Large language models as traffic signal control agents. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 2335–2346, 2025.

Siqi Lai, Yansong Ning, Zirui Yuan, Zhixi Chen, and Hao Liu. USTBench: Benchmarking and dissecting spatiotemporal reasoning capabilities of LLMs as urban agents. In *International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=ETzBStUFJy.

Siqi Lai, Pan Zhang, Yuping Zhou, Jindong Han, Yansong Ning, and Hao Liu. Trafficclaw: A generalizable llm agent in the unified physical environment for urban traffic control. *arXiv preprint arXiv:2604.17456*, 2026b.

Da Lei, Feng Xiao, Lu Li, and Yuzhan Liu. Signalclaw: Llm-guided evolutionary synthesis of interpretable traffic signal control skills. *arXiv preprint arXiv:2604.05535*, 2026.

Junwei Liao, Muning Wen, Jun Wang, and Weinan Zhang. Marft: Multi-agent reinforcement fine-tuning. *arXiv preprint arXiv:2504.16129*, 2025.

Fan Liu, Zherui Yang, Cancheng Liu, Tianrui Song, Xiaofeng Gao, and Hao Liu. Mm-agent: Llm as agents for real-world mathematical modeling problem. In *Advances in Neural Information Processing Systems*, volume 38, pp. 20881–20934, 2026a.

Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, et al. Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization. *arXiv preprint arXiv:2601.05242*, 2026b.

Shuo Liu, Zeyu Liang, Xueguang Lyu, and Christopher Amato. Llm collaboration with multi-agent reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 32150–32158, 2026c.

Sumeet Ramesh Motwani, Chandler Smith, Rocktim Jyoti Das, Rafael Rafailov, Ivan Laptev, Philip HS Torr, Fabio Pizzati, Ronald Clark, and Christian Schroeder de Witt. Malt: Improving reasoning with multi-agent llm training. 2024.

Jaehyun Nam, Jinsung Yoon, Jiefeng Chen, Jinwoo Shin, Sercan Arik, and Tomas Pfister. Mle-star: Machine learning engineering agent via search and targeted refinement. In *Advances in Neural Information Processing Systems*, volume 38, pp. 116692–116712, 2026.

Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 30215–30248, 2025.

Qwen Team. Qwen3.6-27B: Flagship-level coding in a 27B dense model, April 2026. URL https://qwen.ai/blog?id=qwen3.6-27b.

SiliconFlow. SiliconFlow model platform, 2026. URL https://siliconflow.cn/models. Accessed: 2026-09-26.

Pravin Varaiya. Max pressure control of a network of signalized intersections. *Transportation Research Part C: Emerging Technologies*, 36:177–195, 2013.

Leizhen Wang, Peibo Duan, Hao Wang, Yue Wang, Jian Xu, Nan Zheng, and Zhenliang Ma. Evolvesignal: A large language model powered coding agent for discovering traffic signal control algorithms. *arXiv preprint*, 2025a.

Zora Zhiruo Wang, Jiayuan Mao, Daniel Fried, and Graham Neubig. Agent workflow memory. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63897–63911. PMLR, 13–19 Jul 2025b. URL https://proceedings.mlr.press/v267/wang25bx.html.

Hua Wei, Chacha Chen, Guanjie Zheng, Kan Wu, Vikash Gayah, Kai Xu, and Zhenhui Li. Presslight: Learning max pressure control to coordinate traffic signals in arterial network. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1290–1298, 2019a.

Hua Wei, Nan Xu, Huichu Zhang, Guanjie Zheng, Xinshi Zang, Chacha Chen, Weinan Zhang, Yanmin Zhu, Kai Xu, and Zhenhui Li. Colight: Learning network-level cooperation for traffic signal control. In *Proceedings of the 28th ACM international conference on information and knowledge management*, pp. 1913–1922, 2019b.

Hua Wei, Guanjie Zheng, Vikash Gayah, and Zhenhui Li. Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. *ACM SIGKDD explorations newsletter*, 22(2):12–18, 2021.

Qiang Wu, Liang Zhang, Jun Shen, Linyuan Lü, Bo Du, and Jianqing Wu. Efficient pressure: Improving efficiency for signalized intersections. *arXiv preprint arXiv:2112.02336*, 2021.

Zirui Yuan, Siqi Lai, and Hao Liu. Collmlight: Cooperative large language model agents for network-wide traffic signal control. In *International Conference on Learning Representations*, volume 2026, pp. 66687–66715, 2026.

Z.ai. GLM-5.3: Frontier coding with emergent cyber capabilities, August 2026. URL `https://z.ai/blog/glm-5.3`.

Jinwei Zeng, Chao Yu, Xinyi Yang, Wenxuan Ao, Qianyue Hao, Jian Yuan, Yong Li, Yu Wang, and Huazhong Yang. Citylight: a neighborhood-inclusive universal model for coordinated city-scale traffic signal control. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 4036–4044, 2025.

Yujie Zhao, Lanxiang Hu, Yang Wang, Minmin Hou, Hao Zhang, Ke Ding, and Jishen Zhao. Stronger-MAS: Multi-agent reinforcement learning for collaborative LLMs. In *International Conference on Learning Representations*, 2026. URL `https://openreview.net/pdf/285b2536e21c0713523e56ea1c3284197173195b.pdf`.

Boyuan Zheng, Michael Y Fatemi, Xiaolong Jin, Zora Zhiruo Wang, Apurva Gandhi, Yueqi Song, Yu Gu, Jayanth Srinivasa, Gaowen Liu, Graham Neubig, et al. Skillweaver: Web agents can self-improve by discovering and honing skills. *arXiv preprint arXiv:2504.07079*, 2025.

Zhilun Zhou, Jianghao Yu, Yuming Lin, Sun Yongquan, Depeng Jin, Yong Li, et al. Urbands: A graph-guided llm multi-agent system for data-intensive urban tasks. *arXiv preprint arXiv:2607.26724*, 2026.

CONTENTS

1 Introduction **1**

2 Preliminaries **2**

3 Method **3**

3.1 EvolveLight Framework 3

3.2 Spatiotemporal Skill Library Management 4

3.3 Decoupled Cooperative Multi-Agent RL 5

4 Experiments **6**

4.1 Experimental Setup 6

4.2 Overall Performance Comparison 7

4.3 Self-Evolving Capability Comparison 7

4.4 Ablation Studies 8

5 Related Work **9**

6 Conclusion and Future Work **9**

A Appendix **14**

A.1 Road Network Description 14

A.2 Baseline Methods 14

A.3 Implementation Details 15

A.4 Skill Library Utility Score Distribution 15

A.5 Inference Cost Analysis 15

A.6 Strategy Distribution Analysis 16

A.7 Agent Prompt Templates 17

A.7.1 Planning Agent 17

A.7.2 Modeling Agent 20

A.7.3 Skill Agent 23

A APPENDIX

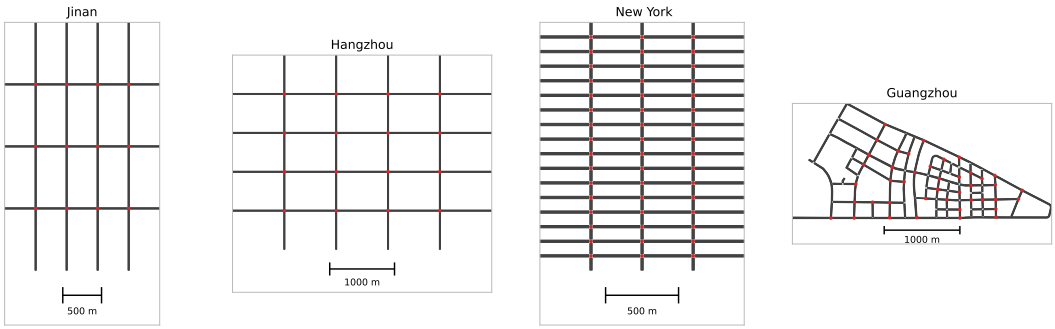

Figure 4: The snapshot of the road networks.

A.1 ROAD NETWORK DESCRIPTION

We evaluate four road networks. Jinan, Hangzhou, New York, and Guangzhou are constructed from OpenStreetMap and real-world traffic data. The snapshot of the road networks are shown in Figure 4.

- **Jinan**: This 3×4 network covers 12 four-leg intersections in the Dongfeng sub-district of Jinan, China. At each intersection, the east–west approaches are 400 m long and the north–south approaches are 800 m long. Traffic demand is derived from roadside-camera records.
- **Hangzhou**: This 4×4 network covers 16 four-leg intersections in the Gudang sub-district of Hangzhou, China. At each intersection, the east–west approaches are 800 m long and the north–south approaches are 600 m long. Traffic demand is derived from roadside-camera records.
- **New York**: We use the 48-intersection variant of the Upper East Side, Manhattan network released with CoLight. Its traffic demand is derived from public taxi-trip records by mapping trip endpoints to intersections and routing trips through the network (Wei et al., 2019b).
- **Guangzhou**: Our Guangzhou network contains 44 intersections and is used only for transfer and long-horizon adaptation evaluation. Unlike the regular-grid benchmarks above, it combines three-leg and four-leg intersections, nonuniform road lengths, and traffic demand that evolves over the 24-hour simulation horizon. Traffic demand is derived from roadside-camera records.

A.2 BASELINE METHODS

We compare EvolveLight with transportation-based controllers, RL-based controllers, LLM-based TSC agents, and frontier LLM backbones. The methods are summarized below.

- **FixedTime**: A nonadaptive controller that repeats a predetermined cycle with fixed green times, independently of the observed traffic state.
- **MaxPressure**: A decentralized, training-free controller that selects the phase with the largest pressure, computed from the imbalance between upstream and downstream queues (Varaiya, 2013).
- **MPLight**: A scalable decentralized RL controller that shares a phase-competition network across intersections. It adopts the FRAP architecture and represents traffic with the current phase and movement pressure while optimizing an intersection-pressure reward (Chen et al., 2020).
- **CoLight**: A multi-agent RL controller that uses graph attention to learn communication weights among neighboring intersections, thereby coordinating phase decisions at the network level (Wei et al., 2019b).
- **Efficient-XLight**: Efficient-XLight variants of MPLight and CoLight that replace conventional vehicle-count or pressure observations with efficient pressure, which normalizes queue imbalance by the numbers of incoming and outgoing lanes (Wu et al., 2021).
- **CityLight**: A parameter-sharing universal policy for heterogeneous city-scale intersections. Its Neighbor Influence Encoder represents traffic-flow relations and connectivity, while its attention-based Neighbor Influence Aggregator combines the competitive and cooperative effects of adjacent intersections (Zeng et al., 2025).

- **LLMLight**: An LLM-based controller that converts structured local traffic observations into natural-language prompts and directly selects signal phases through reasoning, with optional instruction fine-tuning for TSC (Lai et al., 2025).
- **CoLLMLight**: A cooperative LLM-agent framework that augments intersection-level reasoning with communication among neighboring agents for network-wide signal coordination (Yuan et al., 2026).
- **Qwen3.6-27B**: An open-weight 27-billion-parameter dense model with a hybrid architecture that interleaves Gated DeltaNet and full-attention layers. It supports a native 262K-token context and is optimized for reasoning, tool use, and agentic coding (Qwen Team, 2026).
- **DeepSeek-V4-Flash**: A 284-billion-parameter mixture-of-experts model with 13 billion active parameters and a one-million-token context window. Its hybrid sparse-attention design targets efficient long-context reasoning and agentic execution (DeepSeek-AI, 2026).
- **GLM-5.3**: A 744-billion-parameter mixture-of-experts model with approximately 40 billion active parameters. Built on the GLM-5 architecture, it emphasizes complex coding and long-horizon agent tasks, with the 5.3 release deriving its improvements from additional post-training (GLM-5-Team, 2026; Z.ai, 2026).

A.3 Implementation Details

Supervised Cold-Start. We collect 2,026 demonstration trajectories with DeepSeek-V4-Flash and retain high-quality trajectories through the rejection sampling of Eq. equation 10. All three agents share the Qwen3.6-27B backbone and are fine-tuned with LoRA (rank 64, scaling factor 128, applied to all linear layers). We optimize with AdamW at learning rate 1×10^{-5} with cosine decay, a warmup ratio of 0.03, weight decay 0.0, gradient clipping 1.0, and betas $(0.9, 0.999)$. Training uses a global batch size of 8 for 1 epoch with a maximum sequence length of 32,768 tokens, and 5% of the trajectories are held out for validation.

Decoupled Cooperative Multi-Agent RL. Each training step samples a joint group of $G = 8$ trajectories. We use PPO clip $\epsilon = 0.2$, a KL penalty coefficient $\beta = 0.01$ with the low-variance KL estimator, and an entropy-pinning regularizer $0.06\,|H(\pi_\theta) - 0.30|$ that stabilizes the policy entropy around 0.30. Rollouts are sampled at temperature 0.8 and validation uses temperature 0.3. Each trajectory is budgeted at 6,144 prompt tokens and 26,624 response tokens. Reasoning turn budgets are 15 actions for planning, 10 for modeling, and 5 for skill development, with at most 5 deployment validations per time window.

Skill Library. The utility reward of Eq. equation 7 uses the network-wide travel time as the downstream metric with $\gamma = 0.25$, *i.e.*a 25% relative improvement maps to $r = 1$, and the EMA of Eq. equation 8 uses $\alpha = 0.25$. For utility-aware retrieval in Eq. equation 9, the relevance margin is $\delta = \max(0.05,\ 0.1\,\rho_{\max})$, the freshness bonus is set to $b_{\text{fresh}} = 0.08$, and the exploration coefficient is $c = 0.04$. Retrieval returns the top-5 skills, where semantic relevance is a BM25 score ($k_1 = 2.2$, $b = 0.75$) computed over skill names, descriptions, documentation, and tool metadata, augmented by a $0.1\times$ confidence term.

A.4 Skill Library Utility Score Distribution

Figure 5 reveals that EvolveLight learns a non-uniform utility hierarchy among accumulated skills. Notably, the highest-scoring skills consistently involve neighboring spatiotemporal analysis, including upstream–downstream pressure estimation, downstream capacity modeling, and downstream congestion estimation. These skills explicitly capture congestion propagation and spatial dependencies across intersections, leading to more reliable downstream improvements. In contrast, skills relying primarily on isolated intersection-level signals receive substantially lower utilities, suggesting limited transferability under heterogeneous traffic conditions. This pattern indicates that the evolving skill library autonomously identifies coordinated spatiotemporal reasoning as a critical reusable capability for TSC, consistent with the importance of network-level interactions in traffic control. Moreover, the consistent utility ordering across skills demonstrates that the utility-aware scoring mechanism effectively distinguishes valuable capabilities from ineffective ones, enabling successful refinement and retrieval of high-performing skills throughout deployment.

A.5 Inference Cost Analysis

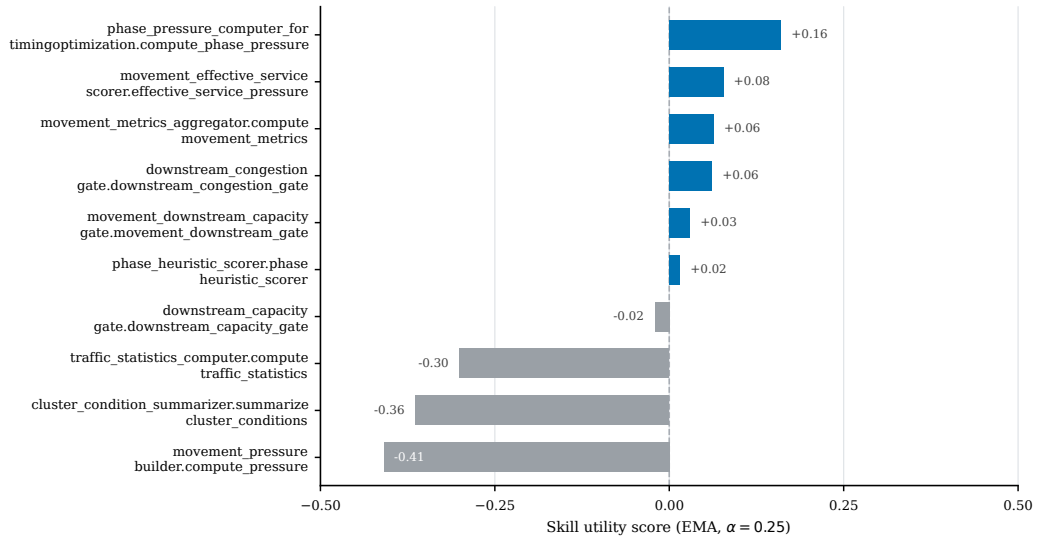

Figure 5: Skill-library utility scores of representative skills in the self-evolving skill library. Scores are exponential moving averages with $\alpha = 0.25$.

We compare EvolveLight with real-time LLM-based control methods in terms of inference cost and control performance during 24-hour deployment on the Guangzhou dataset. The cost is estimated using the SiliconFlow API (SiliconFlow, 2026)[1] with the same model parameters as the training LLMs in (Lai et al., 2025; Yuan et al., 2026). As shown in Figure 6, EvolveLight achieves the most favorable cost–performance trade-off, obtaining the lowest average travel time (4217.6 s) with only \$2.76 LLM inference cost. In contrast, LLMLight and CoLLMLight require 19.2× and 37.3× higher inference budgets while achieving 47.0% and 47.6% worse travel time, respectively. This advantage originates from a different utilization paradigm of LLMs. Real-time LLM controllers repeatedly invoke language models during every control step, causing inference cost to scale with intersections and simulation steps. EvolveLight instead performs LLM reasoning on control policy development, while deploying lightweight controllers for real-time execution. These results demonstrate that EvolveLight enables practical long-horizon LLM-driven traffic adaptation without sacrificing control performance.

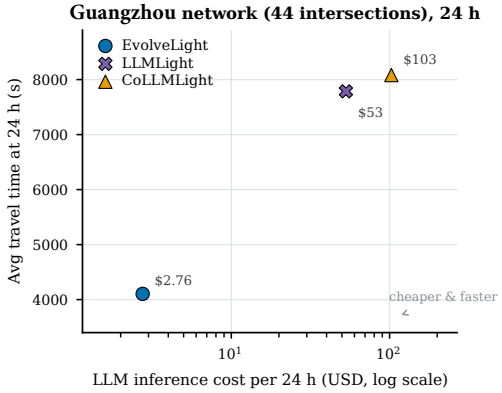

Figure 6: LLM inference cost versus control performance over 24-hour deployment on the Guangzhou dataset.

A.6 Strategy Distribution Analysis

We analyze the spatial distribution of strategies deployed by EvolveLight on the Guangzhou network in Figure 7. EvolveLight maintains a heterogeneous controller portfolio, combining timing-based, transportation-based, and RL-based methods across different regions and periods. Comparing the strategy shares in Figures 7(a–b) with the average queue lengths in Figure 7(c) reveals a clear traffic-dependent allocation pattern: congested regions with longer queues are assigned larger shares of transportation-based and RL-based controllers, whereas free-flow regions are predomi-

[1] https://cloud.siliconflow.cn/me/models

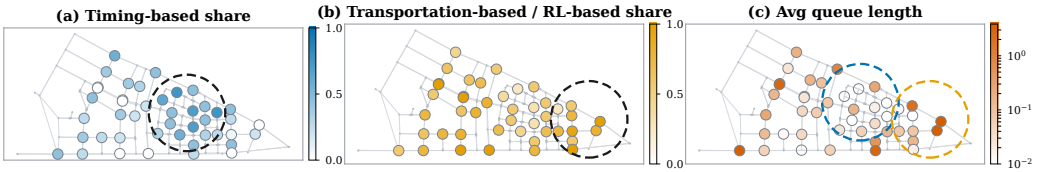

Figure 7: The deployed strategy distribution in Guangzhou dataset.

nantly served by timing-based strategies. This spatial correspondence indicates that EvolveLight correctly matches controller complexity to local traffic conditions, deploying adaptive controllers where congestion requires responsive coordination while retaining efficient timing-based control where traffic remains unconstrained.

A.7 Agent Prompt Templates

We provide the prompt templates used by EvolveLight in their runtime order: the Planning Agent first constructs a deployment plan, and the Modeling Agent then realizes and validates the requested controllers. The Skill Agent is not a mandatory third stage; it is invoked on demand by either of the first two agents when a required reusable capability is absent or needs revision. Angle-bracketed fields denote values injected at runtime. Repeated action results, retry diagnostics, and turn-budget reminders are represented by explicit placeholders.

A.7.1 Planning Agent

The Planning Agent receives network-level evidence and iteratively alternates between analysis and executable deployment planning. In skill-enabled runs, the skill actions shown below are appended to its action space.

Planning Agent—system prompt

```
You are the Planning Agent for EvolveLight traffic optimization.

Incremental workflow:
Analysis and deployment planning build up over iterations. First,
complete basic functional DATA_ANALYSIS and a valid DEPLOYMENT_PLAN.
Later attempts refine the cluster strategy, evidence depth, and
modeling priorities using previous feedback and cached results. Do not
wait for perfect analysis before producing a workable plan.

Return exactly one executable action per response:
  DATA_ANALYSIS('''<raw Python code>''')
  DEPLOYMENT_PLAN('''<raw Python code>''')

When skill-agent mode is enabled, the following are also available:
  SKILL_RETRIEVAL('''{
    "query": "...", "categories": [...], "top_k": 5
  }''')
  SKILL_DEVELOPMENT('''{
    "goal": "...", "categories": [...],
    "requirements_doc": "..."
  }''')
  SKILL_UPDATE('''{
    "skill_id": "...", "issue": "...",
    "requirements_doc": "..."
  }''')

Action-format rules:
- Replace every placeholder with real code or JSON.
```

```
- Do not use Markdown fences or a leading language tag.
- Write raw Python directly inside the triple quotes.

DATA_ANALYSIS:
- Gather cluster-level evidence sufficient for a sound deployment
  plan.
- Inspect only loaded variables; do not read files or mutate
  controllers.
- Available evidence includes traffic states, observations,
  intersection
  ids, timing plans, control phases, spatial and cluster context,
  previous strategies, previous deployment plans, previous modeling
  feedback, and the dynamic incumbent baseline.
- Use summarize_intersection_conditions(...) or
  iter_lane_metrics(...) for robust lane aggregation.
- Planning, Modeling, and Skill Agents share the analysis cache.
- Wait for the Action Result before producing DEPLOYMENT_PLAN.

DEPLOYMENT_PLAN:
- Compare every candidate with the dynamic incumbent. A candidate
  becomes
  the new incumbent only if its validated avg_queue_len and
    avg_travel_time
  are both strictly lower. Treat waiting time as an important
    secondary
  outcome.
- First compute a partition of all intersections into clusters; then
  assign each cluster exactly one strategy:
      "TimingOptimization" | "Heuristic" | "RL".
- Derive clusters in code from intersection ids, traffic evidence,
  flow
  direction, and spatial adjacency. Never hardcode city-specific
    lists.
- Cover every intersection exactly once. Multi-intersection clusters
  must
  be spatially contiguous.
- Scale the cluster count with network size and preserve strongly
  coupled
  corridors; place fixed/adaptive boundaries at weak edges.
- Use either Heuristic or RL as the adaptive family, never both in the
  same deployment. If Heuristic has no clear advantage, prefer RL.
- Zero timing-controlled intersections is valid. In automatic
  allocation,
  normally begin near 10% TimingOptimization and retain an adaptive
  majority. Never produce an all-timing plan.
- Explain cluster rationale, modeling priorities, and how previous
  Modeling Agent feedback is addressed.

The code must end by assigning exactly:
  result = {
    "deployment_clusters": [
      {
        "cluster_id": <str>,
        "intersections": <list[str]>,
        "strategy": "TimingOptimization" | "Heuristic" | "RL",
        "rationale": <short str>
      },
      ...
    ],
    "summary": <str>
  }

Skill use:
- Retrieve relevant skills before an analysis or planning action.
```

```
- Call an authorized tool as
    skill_tools.call(skill_id, tool_name, **kwargs).
- Request only small, reusable components. A development request must
  specify purpose, inputs, outputs, and acceptance criteria.
- If a skill fails or lacks required behavior, update that skill,
  retrieve
  the new version, and then continue planning.

Output limits:
Print at most 100 words per action. Print only compact network/cluster
aggregates and a small top-k bottleneck list. Save detailed tables and
intermediate artifacts to cache.
```

Planning Agent—initial user prompt

```
Network: <number_of_intersections> intersections
User Requirements: <user_requirements>
Current Time Window: <window_index>
  (<window_start_seconds>s - <window_end_seconds>s)

Previous Active Strategy:
<previous_strategy_context>

Dynamic Incumbent Baseline:
<incumbent_source_window_metrics_and_deployment_counts>

Previous Modeling Agent Feedback:
<previous_modeling_feedback>

Spatial Network Context:
<spatial_network_summary>

Current Cluster Context (reference only):
<cluster_context_summary>

TimingOptimization allocation:
<automatic_or_required_ratio_guidance>

Strategy guidance:
Choose either Heuristic or RL alongside any TimingOptimization
  clusters;
never use both adaptive families or an all-timing plan. If Heuristic
  has
no clear advantage, prefer RL. Reuse validated heuristic features when
moving to RL. Beat the incumbent and retain TimingOptimization only
  where
it improves performance.

<skill_library_state_and_retrieval_guidance>

Available Traffic States (<state_count> states):
  - Window starting at <time>s: <state_reference>
  ...

Use current evidence and cached results. Start with DATA_ANALYSIS only
when more evidence is needed.
```

Planning Agent—iterative follow-up

```
Iteration <iteration> completed.
<remaining_turns_or_final_turn_notice>
Deployment test validations remaining:
  <remaining_validations>/<validation_budget>
```

```
Action Result:
<bounded_action_result>

<modeling_summary_and_metrics_vs_incumbent>
<previous_window_performance_feedback>
<skill_action_result_or_repair_guidance>

Next required step:
<produce_or_revise_DEPLOYMENT_PLAN;
 run_DATA_ANALYSIS_only_if_more_evidence_is_needed>
```

A.7.2 MODELING AGENT

After a valid deployment plan is produced, the Modeling Agent develops one shared artifact for each requested strategy family. Its required action advances in the order TimingOptimization, Heuristic or RL, and finally training and simulation; absent strategy families are skipped.

Modeling Agent—system prompt

```
You are the Modeling Agent for traffic-signal controllers.

Incremental workflow:
Complete one working artifact per required strategy family, then call
TRAIN_AND_SIMULATION. Each attempt changes at most one or two elements
(a feature set, scoring rule, or small architecture component). Begin
from the supplied baseline and extend prior best-policy code only
  after
validation beats the incumbent.

Return exactly one triple-quoted action per response:
  DATA_ANALYSIS('''<raw Python code>''')
  TIMING_PLAN('''<raw Python code>''')
  HEURISTIC_PLAN('''<raw class code>''')
  RL_PLAN('''<raw class code>''')
  TRAIN_AND_SIMULATION('''<concise summary>''')

When skill-agent mode is enabled, SKILL_RETRIEVAL,
SKILL_DEVELOPMENT, and SKILL_UPDATE are also available.

Follow the injected "Required next action" and action-specific
  guidance.
Model each strategy family once:
- TIMING_PLAN covers all TimingOptimization intersections in one
  result.
- HEURISTIC_PLAN defines one shared class for all Heuristic clusters.
- RL_PLAN defines one shared graph-aware class for all RL clusters.
- Every deployment intersection must have its required artifact before
  TRAIN_AND_SIMULATION.

DATA_ANALYSIS may inspect loaded traffic state, cluster conditions,
spatial context, and the shared cache, but may not read files or
  mutate
controller state.

Controller priorities:
- Adaptive control decides every 20 s. Holding extends the current
  green
  by 20 s; switching gives 3 s of yellow and 2 s of all-red, followed
    by
  15 s of the new green.
- Score candidate phases using movement-specific service demand,
  waiting
```

```
  cost, and downstream receiving capacity. Prevent spillback.
- Add only a small movement-aligned, phase-specific green-wave bonus
  when
  reliable arrivals fall within the candidate green interval.
- Never add the same coordination scalar to every candidate action.
- Build replaceable per-movement indicators while retaining current
  phase
  and movement vehicle count as base features.
- prepare_features must keep the signature
    prepare_features(self, intersection_states, adj_matrix,
                     target_intersection_idx).
- Heuristic choose_action must keep the signature
    choose_action(self, intersection_states, adj_matrix,
                  available_phases, target_intersection_idx).
- Heuristic and RL controllers must use adj_matrix for bounded direct-
  neighbor and, when useful, two-hop context.
- TimingOptimization intersections are absent from adaptive graph
  inputs.
- RL starts from a full-graph baseline: MLP embedding, one-hop
  neighbor
  mean aggregation, and a per-node Q head. forward(...) returns
    Q-values
  for every graph node in one pass.
- RL includes current-phase context. Its training reward is fixed to
  -queue_length; do not define a reward function in RL_PLAN.

Action-specific requirements:

TIMING_PLAN:
- Estimate phase demand from flow, queue, waiting, density, arrivals,
  and
  spillback risk; avoid arbitrary equal splits.
- Provide every target intersection. Green durations must sum to the
  reported cycle length.
- Assign result = {
    intersection_id: {
      "duration": {phase: green_seconds, ...},
      "cycle_length": total_green_seconds
    }, ...
  }.

HEURISTIC_PLAN:
- Define a no-argument class implementing the exact prepare_features
  and
  choose_action interfaces above.
- Candidate scores should encode effective service, downstream gating,
  and bounded graph context.
- Return an integer index into available_phases.

RL_PLAN:
- Define one shared graph-aware torch.nn.Module.
- forward returns [batch, nodes, actions]; choose_action slices the
  target
  node and returns a Python int in the available action range.
- self.feature_dim must equal the final dimension produced by
  prepare_features.
- Infer batch, node, and neighbor dimensions from tensor shapes; never
  hardcode graph size or confuse node count with top-k neighbor count.

TRAIN_AND_SIMULATION:
- Call only after modeling progress is complete.
- Train RL when present and validate the complete deployment.
- Report queue length, waiting time, and arrived-vehicle travel time.
```

```
Skill use:
- Retrieve relevant modeling skills before constructing an artifact.
- Pin the displayed skill version and use JSON-compatible arguments.
- A skill may be called through skill_tools.call(...) or bound once
  with
  skill_tools.bind(...). Convert tensors/arrays to lists before calls
    and
  reconstruct them afterward.
- Request only a feature constructor, graph extractor, phase scorer,
  timing calculation, tensor encoder, or similarly narrow component.
- A skill action does not satisfy the required modeling action.

Output limits:
Print at most 100 words per action; cache detailed evidence.
```

Modeling Agent—initial user prompt

```
Deployment summary:
<planning_agent_deployment_summary>

Last validation:
<candidate_metrics_vs_dynamic_incumbent>

Previous feedback:
<previous_modeling_feedback>

Deployment:
  TimingOptimization: <compact_intersection_list>
  Heuristic: <compact_intersection_list>
  RL: <compact_intersection_list>

Deployment clusters:
  <cluster_id>: <algorithm>; strategy=<strategy>;
      <compact_intersection_list>
  ...

Required next action:
<DATA_ANALYSIS | TIMING_PLAN | HEURISTIC_PLAN |
 RL_PLAN | TRAIN_AND_SIMULATION>

<action_specific_guidance>
<full_reference_example_for_current_action>
<previous_best_code_for_current_action>

Recent traffic states (<number_of_windows> windows):
<traffic_state_summary>

Intersection observations:
<compact_observation_summary>

Spatial network context:
<compact_spatial_summary>

Core variables:
  traffic_states, cluster_context, cluster_conditions,
  intersection_observations, spatial_network_context,
  deployment_clusters, timing_intersections,
  heuristic_intersections, rl_intersections,
  intersection_control_phases, current_timing_plans,
  default_cycle_length, iter_lane_metrics,
  save_cache, load_cache, list_cache

Control phases: <phase_summary>
Timing plans: <strategy_timing_summary>
```

```
Available cache keys: <cache_keys>

<skill_retrieval_and_runtime_guidance>
<modeling_progress>
<next_cluster_instruction>
```

Modeling Agent—iterative follow-up

```
Action Result:
<bounded_action_result>

<artifact_validation_or_runtime_error>
<skill_action_result_or_repair_guidance>
<updated_modeling_progress>

Required next action:
<next_missing_artifact_or_TRAIN_AND_SIMULATION>
```

A.7.3 SKILL AGENT

The Skill Agent is instantiated only for development or update requests. Retrieval itself is handled directly by the library and does not start a separate Skill Agent conversation.

Skill Agent—system prompt

```
You are the Skill Agent for EvolveLight.

Your job is to analyze traffic/controller evidence when necessary and
turn reusable engineering ideas into executable, versioned skills. A
skill must generalize beyond one experiment or intersection. Never
hardcode intersection ids, experiment paths, checkpoints, or a final
scenario-specific policy. Do not use network access, file access,
subprocesses, threads, dynamic imports, exec, or eval.

Develop small component skills only. Narrow requests for an entire
  agent
workflow, complete control policy, deployment planner, incumbent
  manager,
training pipeline, or broad traffic-optimization system to one
  concrete
reusable utility.

Return exactly one action per response:
  DATA_ANALYSIS('''<raw Python code>''')
  SKILL_DEVELOPMENT('''<complete JSON object>''')
  SKILL_UPDATE('''{
    "skill_id": "<id>",
    "version": <optional_integer>,
    "reason": "<why>"
  }''')

Write raw Python or JSON directly inside the triple quotes. Replace
  every
placeholder and emit no Markdown or explanatory text outside the
  action.

DATA_ANALYSIS:
- Use only when the supplied request, evidence, and cache are
  insufficient.
- Available context includes traffic states, simulation history,
  observations, intersection ids, spatial context, deployment plans
    and
```

```
  clusters, timing plans, control phases, modeling artifacts, the
    request,
  and evidence.
- Share the analysis cache with the Planning and Modeling Agents.
- Use iter_lane_metrics(...) and
  summarize_intersection_conditions(...) for robust aggregation.
- Inspect data only. Wait for the Action Result before developing a
  skill.

SKILL_UPDATE:
- Use when a related skill may be relevant and its complete previous
  version is needed.
- Provide skill_id and optional version. After the system returns the
  old
  version, emit SKILL_DEVELOPMENT with the same skill_id to publish
    the
  next immutable version.

SKILL_DEVELOPMENT:
- Return one JSON object:
  {
    "summary": <what_was_built_or_repaired_design_and_limits>,
    "skills": [<skill>, ...]
  }
- Prefer one focused skill. Create multiple skills only when each has
  independent reuse value.
- Compact skill schema:
  {
    "name": <short_human_name>,
    "category": "analysis" | "deployment" | "modeling",
    "description": <one_clear_reusable_purpose>,
    "source_code": <Python_source_as_JSON_string>,
    "doc": <detailed_markdown_usage_documentation>
  }
- The framework fills identifiers, versions, entrypoints, schemas,
  evidence, confidence, status, tags, cost, and persistence metadata.
- A repaired skill retains its skill_id, explains the root cause and
  fix,
  adds regression coverage, and publishes a new immutable version.
- Review related skills before creating a new id; prefer extension or
  versioning over duplication.

Valid skill families:
1. Traffic-data construction, cleaning, aggregation, diagnostics,
   statistics, bottleneck detection, and cache-ready summaries.
2. Small deployment primitives such as connectivity checks, corridor
   labels, congestion scores, cluster validation, or one-step
     suitability.
3. Modeling feature constructors for normalized traffic, graph,
  multi-hop,
   receiving-capacity, and spillback features.
4. Reusable phase scoring, bounded graph reasoning, arrival
  estimation,
   phase gating, and tie breaking.
5. Volume estimation, critical-flow analysis, cycle calculation, green
   splits, phase demand, and timing feasibility checks.
6. Reusable RL feature constructors and torch modules such as graph
   encoders, message-passing blocks, movement encoders, and dueling
     heads.

Implementation rules:
- Runtime tools are deterministic, stateless, bounded in cost, and
  independent of conversation state.
- Allowed dependencies are Python builtins, math, statistics, NumPy,
```

```
  pandas, and torch.
- The top-level tool accepts and returns JSON-compatible values.
- Define at least one top-level Python function. The first function is
  the
  entrypoint unless a legacy entrypoint is supplied.
- source_code uses ordinary JSON newline escapes, not double escaping.
- Static validation occurs at publication; runtime failures are
  repaired
  through SKILL_UPDATE.

Output limits:
Print at most 100 words. Never print complete source code, full skill
  JSON,
large fixtures, full tables, or full intersection lists.
```

Skill Agent—initial user prompt

```
Develop executable skills from the request and evidence below.

Source agent: <planning_or_modeling>

Request:
<JSON_request_with_goal_categories_requirements_and_optional_existing_
  skill>

Related existing skills
(prefer retaining/versioning over duplication):
<top_five_semantically_relevant_skill_summaries>

Evidence:
<traffic_controller_and_execution_evidence>
```

Skill Agent—iterative follow-up

```
<DATA_ANALYSIS | SKILL_UPDATE | VALIDATION> Action Result:
<bounded_result_or_complete_existing_skill>

<format_retry_validation_errors_or_repair_requirements>
<remaining_turn_budget>

Return exactly one next action. On successful validation, control
  returns
to the requesting Planning or Modeling Agent.
```