URBANAGENT: A TOOL-AUGMENTED AGENT FOR CROSS-SYSTEM URBAN TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern cities rely on an increasing number of digital services to operate, but residents' daily needs are still difficult to meet. Services are fragmented and have little interoperability, placing a heavy operational burden on users. Existing digital platforms, urban foundation models, and intelligent assistants each address only isolated aspects of an urban task. But they struggle to reliably convert complex natural-language requests into executable cross-system workflows. We propose UrbanAgent, a tool-augmented agent framework for cross-system urban tasks. It couples the cognitive and reasoning capabilities of a large language model with a toolset supporting code execution, API calls, and Model Context Protocol. Through one adaptive closed loop, it clarifies missing information before acting, grounds tool use in live observations, and aligns the final response with observed evidence and task constraints. To address the evaluation gap, we introduce UrbanEval, a benchmark specifically designed for cross-system urban request. Unlike prior benchmarks that assess either general tool use or urban knowledge and reasoning, UrbanEval evaluates both task results and execution quality, including required tool coverage, dependency validity, and evidence traceability. Experimental results indicate that UrbanAgent reaches a 71% task success rate, 10points above the strongest baseline. This lead holds across GPT-5-mini, Gemini-2.5-flash, DeepSeek-V4-flash, and Qwen3-235B-A22B.

1 INTRODUCTION

Modern cities have deployed extensive digital systems for transportation, urban planning, environment, energy, and public services Zheng et al. (2014).Yet this digitalization has not translated into convenience. These functions are scattered across separate systems with independent data, interfaces, and procedures. Even a simple daily request involves multiple aspects. Figure 1 illustrates a representative request that combines weather-related route planning with restrictions on restaurants along the way. Existing services can satisfy individual subtasks, but the user must manually transfer intermediate results, compare alternatives, and verify the final combination. Such requests are typically multi hop and compositional, spanning multiple systems, locations, and constraints He et al. (2025). The manual coordination required is especially burdensome for older residents, who need more time to find the right function in feature-rich interfaces and make more errors Yu & Chattopadhyay (2024).

Existing research addresses separate components of the coordination problem. Integrated urban platforms such as Hangzhou City Brain aggregate municipal data and services into a unified operating layer for traffic, public safety, and environmental management Zhang et al. (2019). However each deployment remains bounded by a single domain or operating organization and does not act on an individual resident's cross-system request. Urban foundation models such as UrbanGPT and CityGPT support spatiotemporal prediction and spatial reasoning, but do not interact with real-time service systems Li et al. (2024); Feng et al. (2024). Agent systems go a step further and take actions on real applications and services, yet each agent is confined either to a single application's interface or to a single predefined task. GUI agents such as AppAgent and Mobile-Agent focus on interactions within a single application Zhang et al. (2025); Wang et al. (2024), while urban vertical agents such as UrbanKGent and LLMLight execute specific domain tasks, including knowledge graph construction and traffic signal control Ning & Liu (2024); Lai et al. (2024). Existing approaches integrate

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.

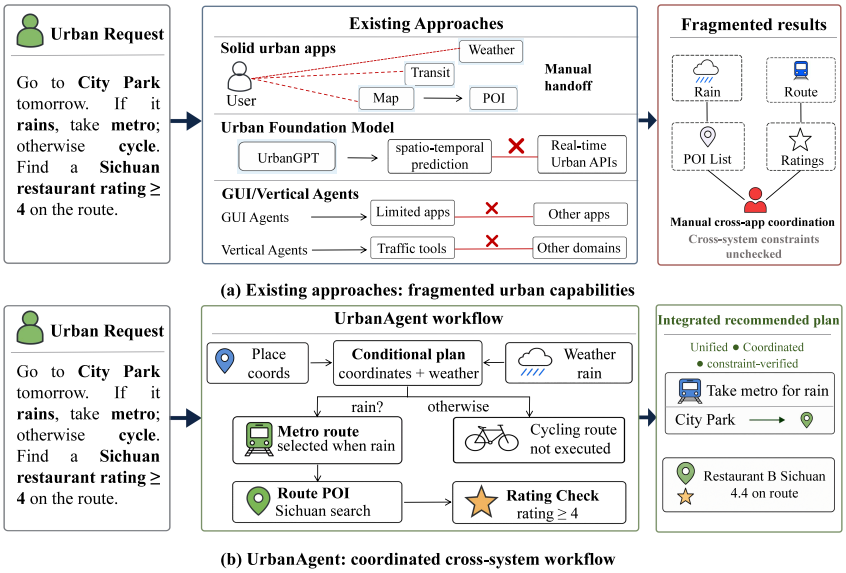


Figure 1: Fragmented urban services vs UrbanAgent. (a) Existing approaches each cover one fragment of a compositional request. (b) UrbanAgent executes the same request as one coordinated, evidence-aligned cross-system workflow.

urban data, reason about cities, or act within a single application or domain, but do not coordinate heterogeneous services around a complete user request.

Tool-augmented LLM agents can coordinate independent services through standardized tool interfaces. They select and invoke tools, then pass intermediate results between them. Prior work has applied such agents to web navigation and software engineering Yao et al. (2023); Qin et al. (2024); Zhou et al. (2024); Xie et al. (2024); Jimenez et al. (2024). These advances make general tool agents a promising basis for executing cross-system urban tasks. However, direct transfer is not feasible. Urban requests are usually expressed as high-level goals rather than complete execution paradigms. Fulfilling them may require real-time environmental conditions, spatial relations, transportation options, and local-service information from providers with different geographic coverage. The workflow also emerges during execution because later calls may depend on places or conditions returned earlier. Although none of these challenges is unique in isolation, their combination makes urban workflows particularly fail. When a user asks for the "nearest" service without providing a location, an agent may silently assume one. Then it can retrieve a valid place, obtain current weather, and compute an accurate route for the wrong location. Each tool call may appear successful, but their combination produces a plausible but unusable recommendation. Therefore, reliable execution requires clarification before action, consistency checks, ordered tool use, and constraint verification.

To address this problem, we propose UrbanAgent, a tool-augmented agent framework for open urban tasks. It owns four components including cognition, a reasoning–execution core, tool invocation, and synthesis. They clarify missing critical task inputs before taking action, select each operation based on previous observations, rely on call consumption of upstream values, recover from empty or geo-invalid returns, and align the final response with collected evidence and task constraints. Every action and statement is grounded in tool observations rather than the model's prior knowledge.

Furthermore, we find no benchmark evaluates agents on coordinating heterogeneous urban services across domains. General tool-use benchmarks test function calls in non-urban settings Patil et al. (2025); Qin et al. (2024); Liu et al. (2023). Urban benchmarks test city knowledge, reasoning, or simulation, not live service execution Feng et al. (2025); Zhou et al. (2025). Agentic local life search stays within a single domain category He et al. (2025). To bridge this gap, we construct UrbanEval. The requests range from ambiguous tasks that require clarification to constrained tasks with ordered dependencies. Because a plausible answer can conceal an incomplete or invalid execution trace, UrbanEval assesses both task results and execution quality, including tool coverage, dependency order, and evidence grounding.

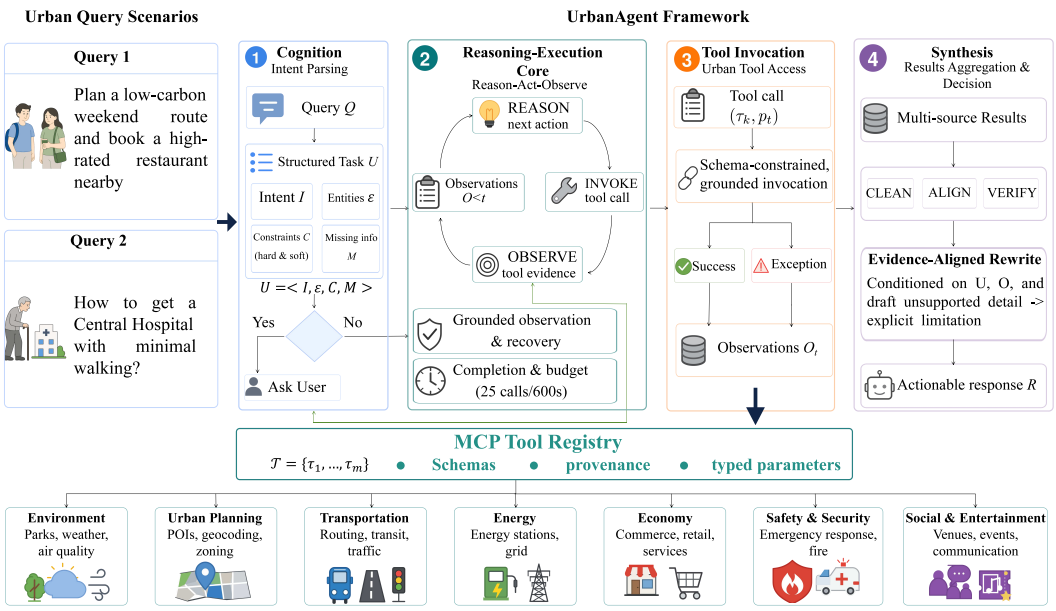


Figure 2: Framework of UrbanAgent. (1) Cognition maps a query to a structured task $\mathcal{U}$ and requests clarification when a key task parameter is missing. (2) The reasoning–execution core selects the next action from prior observations, and (3) tool invocation executes that action through the unified registry $\mathcal{T}$; the two alternate until the agent commits a preliminary answer $\hat{\mathcal{R}}$. (4) Synthesis aligns the draft with multi-source observations, verifies task constraints, and commits the final answer $\mathcal{R}$.

To address these gaps, we make the following contributions:

- We propose UrbanAgent to bridge the gap between high-level user requests and fragmented urban services. It clarifies missing parameters, preserves cross-service dependencies, and grounds the final response in validated tool observations.
- We introduce UrbanEval to make cross-system urban execution evaluable. It pairs 250 requests with trace-aware criteria that verify missing handling, required capability coverage, dependency preservation, and evidence support. Therefore, it distinguishes between truly completed workflows and plausible answers produced through incomplete execution.
- We show that UrbanAgent improves cross-system task completion over general orchestration baselines under matched models and tools. With GPT-5-mini, UrbanAgent completes 64.0% of executable tasks vs 53.5% for the strongest baseline, and it achieves the highest TSR across all tested base models.

2 URBANAGENT

UrbanAgent addresses complex tasks in dynamic urban environments that require coordination across multiple services. When presented with a natural-language request q, the system consults a tool registry $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_m\}$ to gather observations about the urban environment and generate a response $\mathcal{R}$ that meets the request and its constraints.

These tasks present four challenges: 1) requests may be ambiguous or omit essential information; 2) subsequent actions may depend on changing urban conditions and earlier query results; 3) services differ in their interfaces, geographic coverage, and return formats; and 4) relevant search results may still violate spatial, temporal, or service constraints. UrbanAgent addresses these challenges through four modules: cognition and intent parsing, reasoning and execution, tool invocation, and synthesis. The following subsections will provide a detailed exposition of the mechanism design for each module.

2.1 COGNITION AND INTENT PARSING

Natural-language requests often include ambiguous expressions or lack essential details, such as a specific location or date. The cognition and intent parsing module clarifies the task and determines the required information. Leveraging the semantic understanding of the language model, this module maps the original request q to the structured representation $\mathcal{U}$:

$$\mathcal{U} = f_{\text{cog}}^{\theta}(q) = \langle \mathcal{I}, \mathcal{E}, \mathcal{C}_h, \mathcal{C}_s, \mathcal{M} \rangle. \tag{1}$$

Here, $\mathcal{I}$ specifies the task objective and expected output, such as place retrieval, route planning, or service recommendation. $\mathcal{E}$ contains entities and parameters supplied in the request, including locations and time windows. Together, they guide tool selection and argument construction. The module separates user requirements into hard constraints $\mathcal{C}_h$ and soft preferences $\mathcal{C}_s$. Hard constraints define the essential physical and operational requirements a solution must meet—such as spatial distance limits, time windows, and service availability. In contrast, soft preferences, like favoring lower-carbon travel, help compare and rank candidate results.

The module also identifies missing information that requires user clarification and records it in $\mathcal{M}$. If $\mathcal{M}$ is nonempty, UrbanAgent asks for the missing details before calling tools. Otherwise, it proceeds with $\mathcal{U}$ as the shared task representation for execution and synthesis.

2.2 REASONING AND EXECUTION

Urban conditions change over time, and later actions often depend on information obtained from earlier queries. Destination selection may depend on search results, a travel mode may depend on current weather, and an empty return may require a revised search. These dependencies require decisions based on the latest available observations and adjustments to subsequent actions as new feedback arrives. We therefore use a ReAct-style controller (Yao et al., 2023) that interleaves reasoning with tool use and selects subsequent actions from the task representation and accumulated observations.

Let t index controller turns, and T denote the final turn. At turn t, the controller receives the original request q, the task representation $\mathcal{U}$, the execution context C_t, and the tool schemas. It selects an action according to

$$\begin{aligned} &A_t \sim \pi_\theta(\cdot \mid q, \mathcal{U}, C_t, \text{Schema}(\mathcal{T})), \\ &A_t \in \{B_t, \text{Commit}(\hat{\mathcal{R}})\}. \end{aligned} \tag{2}$$

Here, B_t is an ordered batch of tool calls, and $\text{Commit}(\hat{\mathcal{R}})$ submits a preliminary answer. The conditional action distribution π_θ is induced by the base model, its execution instructions, and the function-calling interface. The context C_t includes previous calls, compact return summaries, and runtime feedback, allowing the controller to revise its decisions using the latest results. The controller works with the tool invocation module in a repeated loop of reasoning, calling tools, observing results, and adjusting subsequent actions. After each batch, the returned observations inform the next decision. When the controller commits, it summarizes the accumulated observations as O_T and passes them to synthesis together with the preliminary answer $\hat{\mathcal{R}}$.

2.3 TOOL INVOCATION

Urban services differ in their interfaces, geographic coverage, and response formats, making them difficult to combine within a single task. Execution across services therefore requires selecting providers appropriate for the target region and constructing calls according to their interfaces. The tool invocation module $\text{Exec}_{\mathcal{T}}$ provides a common calling interface for these services by registering remote Model Context Protocol (MCP) services and local computational tools in a shared registry (Anthropic, 2024; Model Context Protocol, 2025; Ouyang et al., 2025). Each tool τ_k has a capability description, a source provider, and an input schema. We denote the set of arguments satisfying that schema by $\mathcal{P}_k$. These descriptions and schemas guide the controller in selecting services and constructing call arguments.

Let i index executed tool calls, k_i identify the tool selected for call i, and s_i denote its clock time. The module executes the query selected by π_θ; given arguments $p_i \in \mathcal{P}_{k_i}$, the call returns

$$y_i = \mathrm{Exec}_{\mathcal{T}}(\tau_{k_i}, p_i; W(s_i)). \tag{3}$$

Here, $W(s_i)$ denotes the urban state at clock time s_i, and y_i is the service response under that state. The agent accesses information about weather, locations, and service availability through these responses rather than observing the full urban state directly. Each return, including any failure feedback, is passed to the reasoning module to support subsequent decisions.

2.4 SYNTHESIS

Even after execution has collected substantial relevant information, the preliminary results may still fail to satisfy the user's mandatory requirements. A candidate location may exceed a distance limit, a route may fall outside a required time window, or a service may be unavailable on the requested date. These cases illustrate why semantic relevance to a query does not establish a plan's feasibility in the actual urban environment. A usable plan must respect real-world spatial, temporal, and service conditions as well as the user's stated requirements. The synthesis module therefore rechecks mandatory conditions against the evidence collected during execution before organizing the final response.

At the end of execution, the system extracts an observation view from the complete tool-call trace $O_T = \mathrm{ObsView}(H_T)$. This view summarizes the tool returns and their status. The synthesis module receives this view together with the task representation and preliminary answer:

$$\mathcal{R} = f^{\theta}_{\mathrm{syn}}(\mathcal{U}, O_T, \hat{\mathcal{R}}). \tag{4}$$

The model is instructed to assess hard constraints before considering soft preferences. Hard constraints determine which candidates are feasible, while soft preferences guide comparisons among candidates that satisfy those conditions. It uses the observations in O_T to check candidate results against the requirements in $\mathcal{C}_h$, then considers $\mathcal{C}_s$ when comparing and presenting the results. Synthesis uses the collected observations to revise the draft, aiming to produce a response that reflects both the available evidence and the user's requirements.

3 URBANEVAL BENCHMARK

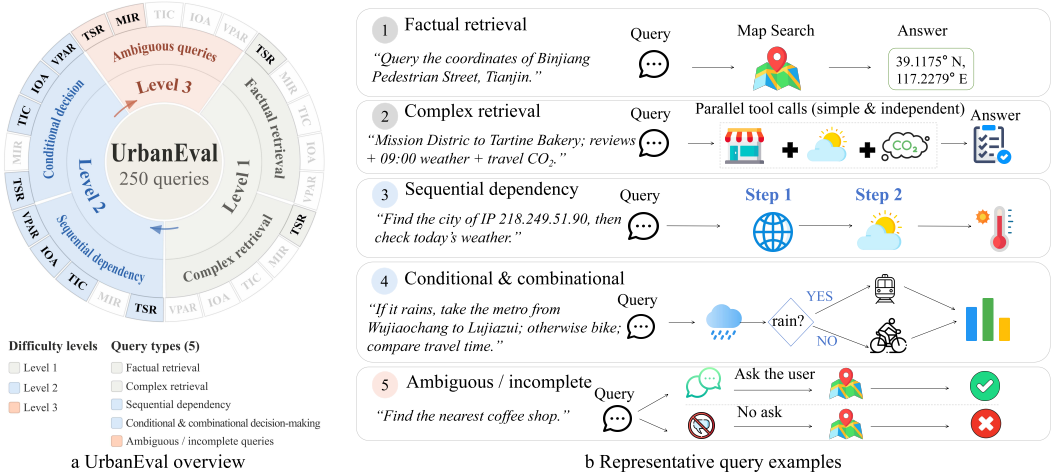
Existing tool-use benchmarks are not designed around cross system urban workflows (Patil et al., 2025; Qin et al., 2024; Liu et al., 2023), while urban benchmarks focus on scene understanding or simulation rather than live service execution (Feng et al., 2025; Zhou et al., 2025). Agentic local life search stays within a single domain category He et al. (2025).UrbanEval instead tests high level requests coordinating heterogeneous urban services across domains.

UrbanEval contains 250 natural-language requests spanning geographic locations across multiple countries and cities. These requests require agent to interact with external services exposed through MCP servers across seven urban-computing domains (Zheng et al., 2014). They cover a wide range of urban service scenarios, extending from geospatial queries and route planning to local service search and recommendations, weather and disaster information retrieval, and carbon emission estimation. UrbanEval does not simply equate a high number of tool calls with a high difficulty. Instead, it defines difficulty based on the information dependencies and decision structures required to complete the request. As shown in Figure 3, Level 1 comprises factual retrieval (FR) and complex retrieval across independent services (CR), testing information acquisition and composition. Level 2 comprises sequential dependency (SD) and conditional or combinatorial decision-making (CC), testing dependency-aware execution and decisions under constraints. Level 3 consists of ambiguous or incomplete intent (AI), testing whether an agent identifies task-critical omissions before acting. The five query types are evenly represented, with 50 requests each.

Each request is paired with a human-authored checklist of independently verifiable criteria rather than a single reference answer. The checklist specifies the required outcomes, tool capabilities, and dependencies between tool calls. For ambiguous or incomplete requests, it also identifies the

Table 1: UrbanEval's three-layer metric mechanism. TIC, IOA, and the efficiency measures are computed deterministically from recorded traces; TSR, MIR, and VPAR are scored by an LLM judge (Qwen3-Max) under a shared rubric. Process metrics are reported on Level 2, and VPAR only on runs having at least one tool call.

Layer	Metric	Definition
Result	TSR	Task success rate — a correct, complete, constraint-satisfying answer
	MIR	Missing-information identification rate — guess-free flagging of a missing task-critical input
Process	TIC	Tool invocation completeness — coverage of the required tool capabilities
	IOA	Invocation order accuracy — dependency pairs called in the required order
	VPAR	Valid proactive augmentation rate — useful augmentation or verification beyond the minimum
Efficiency	Tokens per query	Mean over all queries
	Tokens per success	Total tokens divided by the number of successful tasks, excluding infrastructure failures from both



a UrbanEval overview b Representative query examples

Figure 3: Overview of UrbanEval. (a) The 250 queries span five task categories and three difficulty levels. Every category is scored by result-layer metrics, and Level 2 additionally by process-layer metrics. (b) Representative requests and tool workflows for the five categories.

task-critical information that must be obtained from the user before execution. These criteria are checked against evidence appropriate to the information being evaluated. Stable facts, such as a district's official name, are checked against fixed reference records. Time-varying facts, such as current weather, are checked against tool observations obtained during the run. Execution-derived quantities, such as an estimated arrival time, must be supported by the relevant inputs and outputs in the recorded trajectory. This evidence-based evaluation distinguishes supported task completion from answers that are merely plausible.

Based on these checklists and recorded trajectories, UrbanEval assesses performance at three layers (Table 1). The result layer measures task success (TSR) and the identification of missing task-critical information (MIR). The process layer measures required tool coverage (TIC), dependency-respecting invocation order (IOA), and useful augmentation or verification beyond the minimum requirements (VPAR). The efficiency layer reports token consumption per query and per successful task.

Table 2: Main results on UrbanEval for tool-augmented systems, grouped by base model. GPT-5-mini defines the controlled main setting; the other base models evaluate generalization under the same tool set, budget, and protocol.

		Results		Efficiency		Process		
Base model	Method	TSR ↑	MIR ↑	Tok. ↓	T/Succ. ↓	TIC ↑	IOA ↑	VPAR ↑
GPT-5-mini (main)	Native tool-calling	0.61	0.92	26.7	42.8	0.83	0.99	0.83
	ReAct	0.57	0.94	**21.8**	**37.3**	0.75	0.96	0.84
	Plan-and-Execute	0.47	0.90	133.1	272.8	0.79	0.96	0.84
	AutoGen	0.61	0.96	24.5	40.1	0.86	0.98	0.83
	UrbanAgent	**0.71**	**0.98**	83.8	118.3	**0.89**	**1.00**	**0.96**
Gemini-2.5-flash	Native tool-calling	0.37	0.62	78.1	210.0	0.85	**1.00**	0.79
	ReAct	0.37	0.70	**46.2**	125.5	0.86	**1.00**	0.82
	Plan-and-Execute	0.39	0.72	79.4	202.6	**0.92**	0.94	0.90
	AutoGen	0.31	0.18	89.3	286.2	0.87	0.99	**0.93**
	UrbanAgent	**0.53**	**0.98**	52.9	**99.4**	0.85	0.98	0.86
DeepSeek-V4-flash	Native tool-calling	0.50	0.96	64.0	129.0	0.92	0.97	0.96
	ReAct	0.48	0.96	**58.7**	**122.3**	**0.95**	**0.99**	0.95
	Plan-and-Execute	0.42	0.82	112.9	268.8	0.88	0.97	**0.98**
	AutoGen	0.21	0.20	107.8	508.6	0.91	0.98	0.92
	UrbanAgent	**0.52**	**0.98**	104.3	199.0	0.85	**0.99**	0.94
Qwen3-235B-A22B	Native tool-calling	0.30	0.82	**26.2**	**88.6**	0.69	0.82	0.46
	ReAct	0.28	0.86	31.4	113.9	0.68	0.76	0.53
	Plan-and-Execute	0.32	0.84	83.3	263.5	0.74	0.70	0.72
	AutoGen	0.08	0.06	128.6	1607.8	**0.78**	0.74	0.62
	UrbanAgent	**0.41**	**0.96**	40.4	98.9	0.68	**0.95**	**0.86**

4 EXPERIMENTS

In this section, we aim to answer the following Research Questions (RQs). **RQ1 (Effectiveness and cost):** Does UrbanAgent improve end-to-end task success, and at what token cost? **RQ2 (Process):** Which execution behaviors, tool coverage, dependency order, augmentation, and clarification, accompany its gains? **RQ3 (Robustness):** Does the advantage persist across closed and open base models? In addition, other results from the experiments are also analyzed.

4.1 EXPERIMENTAL SETUP

Baselines. We evaluate four tool-augmented baselines that represent established orchestration strategies: Native tool-calling, which delegates action selection to the model's function-calling interface; ReAct, which interleaves reasoning and tool use Yao et al. (2023); Plan-and-Execute, which plans before execution Wang et al. (2023a); and AutoGen, which pairs an assistant with a tool executor Wu et al. (2024). We also evaluate ten Single LLMs without tool access or live observations.

Settings. To compare orchestration rather than access, every tool-augmented system within a base-model block receives the same queries, MCP tools, tool descriptions, JSON schemas, temperature, and budget (25 LLM calls or 600 seconds per query). When a tool call returns an error, the agent receives it and may retry. So the error alone does not count as a task failure. We follow the UrbanEval protocol. Qwen3-Max at temperature 0 scores them with system identities masked to limit bias Wang et al. (2023b); Zheng et al. (2023), grading TSR and MIR against fixed per-query criteria and VPAR against a fixed trajectory rubric. To keep the benchmark neutral, we froze the UrbanEval queries and schema before finalizing UrbanAgent, and annotated the schema independently of its development. Because these responses vary over time, the schema grades time-varying facts against each run's own returns rather than a fixed key.

4.2 Main Results

Controlled comparison (RQ1). Over all 250 queries UrbanAgent attains an overall TSR of **71%**, against 61% for the strongest baselines, Native tool-calling and AutoGen (Table 2). The 50 ambiguous or incomplete intent queries test clarification rather than execution, so we isolate the 200 executable queries separately: UrbanAgent completes 128 (64.0%) against 107 (53.5%) for Native tool-calling, a 10.5-point gain at matched model and tool access. The gain incurs a higher token cost. 118.3k per successful task is about three times the 37.3–42.8k of Native tool-calling, ReAct, and AutoGen.(Figure 4b). The higher token cost buys reliability on the hard requests. On the 200 executable queries UrbanAgent solves 21 more than Native tool-calling.

The margin concentrates in two categories (Figure 4a): UrbanAgent exceeds the strongest baseline by 16 points on factual retrieval (0.68 vs. 0.52) and 12 on conditional or combinatorial decision-making (0.68 vs. 0.56). It narrows on sequential dependency, where the leading systems fall between 0.68 and 0.72, and reverses on complex retrieval across independent services, where AutoGen edges UrbanAgent 0.50 to 0.48.

A stronger base model does not bridge the gap. The ten Single LLMs (Table 3) include the strongest available models. But all fall within a narrow 0.22–0.26 overall TSR. Their successes are almost entirely clarification: 70–91% arise on the ambiguous or incomplete intent category, while no executable category exceeds 0.22 and the best scores on the conditional or combinatorial decision-making and sequential dependency categories reach only 0.12. Without tools, even a frontier model flags a missing input but cannot satisfy a request that depends on live urban observations. Tool use closes this gap: on GPT-5-mini, TSR rises from 0.22 without tools to 0.71 with UrbanAgent, a 49-point difference. Scale alone does not close this gap. The newest models stay within the same narrow band as the older ones. The bottleneck is access to live data, not model capacity.

4.3 Analysis

Locating the advantage (RQ2). By separating results from process, UrbanEval also localizes the 10.5-point gap. On GPT-5-mini the tool-augmented systems already coincide on whether the right tools are called in the right order: TIC is 0.89 for UrbanAgent against 0.86 for the strongest baseline, and IOA 1.00 against 0.99. Clarification is equally saturated: MIR ranges from 0.90 to 0.98, and every system identifies the missing input and requests it. UrbanAgent departs only in never acting on an assumed value, returning no assumption across the 50 ambiguous or incomplete intent queries against 5–8 for the baselines. Coverage, ordering, and clarification thus do not explain the gap. There are two behaviors can explain including augmentation beyond the minimal toolchain and completion of constrained tasks.

Proactive augmentation. Among the process metrics, only augmentation separates the systems: VPAR reaches 0.96 for UrbanAgent against 0.84 for the strongest baseline. Valid augmentation takes four forms: precision refinement (narrowing a coarse result to an exact one), expanded retrieval (evidence gathered beyond the minimal toolchain), error recovery (a failed call retried with new arguments or another tool), and cross-validation (a fact confirmed from two independent calls). The judge credits at least one on 95 of the 99 Level-2 runs in which UrbanAgent issued a tool call; the remaining run issued none, leaving nothing to augment. Refinement appears in 85 trajectories and expanded retrieval in 54, error recovery in 8 and cross-validation in 10; the categories overlap, so a trajectory may bear several. UrbanAgent thus supplements and cross-checks steps that a minimal, correctly ordered toolchain omits. Refinement is the most common form. A coarse match becomes the exact value that the request needs.

Task completeness. The second locus is completion, which the failure classes expose. Each failed run receives one dominant class: a factual error (a claim its observations do not support), a requirement mismatch (a result that violates a stated constraint), task incompleteness (a required step left undone), or other. UrbanAgent fails 72 of the 200 executable queries, against 93 for Native tool-calling, 104 for ReAct, 127 for Plan-and-Execute, and 96 for AutoGen, and the classes diverge most on incompleteness (Figure 4c): UrbanAgent leaves 20 tasks incomplete against 31–45 for the baselines. One factual retrieval query illustrates the pattern: it requests hotels near Kunming South Railway Station with availability from 12–15 July 2026. All four baselines retrieve hotels but none verifies availability for those dates. Native tool-calling and AutoGen defer the check, ReAct returns no verified hotel, and Plan-and-Execute spends 25 LLM calls on candidates and routes yet never

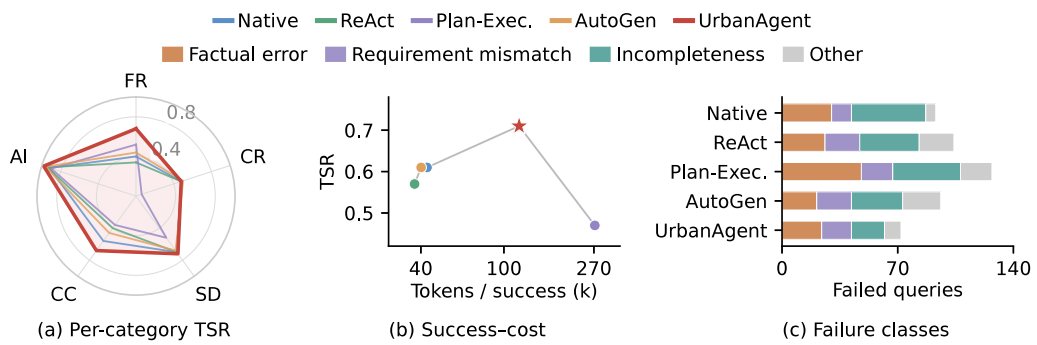


Figure 4: Main results and analysis on GPT-5-mini. (a) TSR by query type: factual retrieval (FR), complex retrieval across independent services (CR), sequential dependency (SD), conditional or combinatorial decision-making (CC), and ambiguous or incomplete intent (AI). (b) TSR vs tokens per successful task. (c) Failure counts by class including factual error, requirement mismatch, incompleteness, and other over the 200 executable queries.

Model	TSR ↑	MIR ↑	Tok. ↓	T/Succ. ↓
GPT-5-mini	0.22	**1.00**	1.6	7.2
Gemini-2.5-flash	**0.26**	0.94	0.5	1.8
DeepSeek-V4-flash	0.25	0.92	0.5	1.9
Qwen3-235B-A22B	0.24	0.96	**0.3**	**1.2**
GPT-5	**0.26**	**1.00**	2.4	9.2
Claude-Opus-4.5	0.25	**1.00**	0.4	1.7
Gemini-3-Pro	**0.26**	0.94	0.9	3.4
DeepSeek-R1	0.22	0.88	0.6	2.8
Kimi-K2.5	**0.26**	0.92	0.9	3.4
GLM-5	0.25	0.88	1.6	6.4

Table 3: Results for ten single LLMs prompted directly, without tools or live observations. The process metrics therefore do not apply.

queries rooms. UrbanAgent checks several nearby hotels for the dates and reports that none has rooms, completing the query the baselines abandon. This case is typical of constrained retrieval. UrbanAgent tests each candidate before it answers.

Robustness across base models (RQ3). UrbanAgent ranks first in TSR across all four base models and first in MIR within each tool-augmented block (Table 2; a few tool-free Single LLMs reach a higher MIR of 1.00, Table 3). The rank is stable but the margin is not: +10 points on GPT-5-mini, +14 on Gemini-2.5-flash, and +9 on Qwen3-235B-A22B, yet +2 on DeepSeek-V4-flash. The margin tracks how strong the baselines already are. On DeepSeek-V4-flash the best baseline already reaches 0.50 TSR.

Limitations. UrbanAgent is more token-intensive, and its gains vary across tasks and backbones; evaluation with time-varying services and an LLM judge does not establish generalization to unseen cities or domains.

5 Conclusion

We propose UrbanAgent, a tool-augmented agent that turns a natural-language urban request into an executable cross-system workflow. Meanwhile, we built UrbanEval, a 250-query benchmark that scores not only whether a request succeeds but how it is executed. Under GPT-5-mini, UrbanAgent reaches 71% TSR, 10 points above the strongest baseline, and completes 64.0% of executable requests against 53.5%. Future work will isolate each component's contribution and add user studies and in-the-wild deployment.

AI USE STATEMENT

Separate from the language models evaluated as agent backbones and judges in the experiments, generative AI assistants were used to provide feedback on methodological framing and experimental presentation; assist with software implementation, refactoring, and testing; support literature discovery and summarization; suggest manuscript organization; and edit prose for readability. The authors reviewed AI-assisted outputs against the cited sources, executable artifacts, and reported results. The authors take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

UrbanEval is intended as a research benchmark, not as a safety certification, medical decision system, or substitute for authoritative public-service information. Its release contains benchmark requests and public-place or service references, but no participant contact details or service credentials. Live-service use should comply with provider terms, and geographic coverage and provider availability can introduce regional bias. Any deployment involving health, mobility, emergency, or other consequential decisions requires independent verification, privacy safeguards, and appropriate human oversight.

REPRODUCIBILITY STATEMENT

Sections 2 and 3 describe the UrbanAgent workflow and the UrbanEval benchmark, including its evaluation criteria and metrics. Section 4.1 specifies the experimental settings and evaluation configuration. The supplementary material provides an end-to-end execution trace. The accompanying code and data release includes a reference implementation aligned with the archived artifacts, prompts, benchmark queries and evaluation schemas, a tool manifest, and verification scripts. It also includes frozen evaluation verdicts for all 250 queries and matching raw execution records. The release therefore supports inspection of the available evidence and fully reconstructs the original runs. Live reruns depend on access to external services and their changing responses.

REFERENCES

Anthropic. Introducing the model context protocol. Anthropic News, November 2024. URL https://www.anthropic.com/news/model-context-protocol.

Jie Feng, Yuwei Du, Tianhui Liu, Siqi Guo, Yuming Lin, and Yong Li. Citygpt: Empowering urban spatial cognition of large language models, 2024.

Jie Feng, Jun Zhang, Tianhui Liu, Xin Zhang, Tianjian Ouyang, Junbo Yan, Yuwei Du, Siqi Guo, and Yong Li. Citybench: Evaluating the capabilities of large language models for urban tasks. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 5413–5424, 2025.

Hang He, Chuhuai Yue, Chengqi Dong, Mingxue Tian, Zhenfeng Liu, Jiajun Chai, Xiaohan Wang, Yufei Zhang, Qun Liao, Guojun Yin, et al. Localsearchbench: Benchmarking agentic search in real-world local life services, 2025.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, 2024.

Siqi Lai, Zhao Xu, Weijia Zhang, Hao Liu, and Hui Xiong. Llmlight: Large language models as traffic signal control agents, 2024.

Zhonghang Li, Lianghao Xia, Jiabin Tang, Yong Xu, Lei Shi, Long Xia, Dawei Yin, and Chao Huang. Urbangpt: Spatio-temporal large language models, 2024.

Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents, 2023.

Model Context Protocol. Model context protocol specification. Official specification, March 2025. URL https://modelcontextprotocol.io/specification/2025-03-26. Version dated 2025-03-26.

Yansong Ning and Hao Liu. Urbankgent: A unified large language model agent framework for urban knowledge graph construction, 2024.

Chaoqian Ouyang, Ling Yue, Shimin Di, Libin Zheng, Linan Yue, Shaowu Pan, Jian Yin, and Min-Ling Zhang. Code2mcp: Transforming code repositories into mcp services, 2025.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. ToolLLM: Facilitating large language models to master 16000+ real-world apis. In *International Conference on Learning Representations*, 2024.

Junyang Wang, Haiyang Xu, Jiabo Ye, Ming Yan, Weizhou Shen, Ji Zhang, Fei Huang, and Jitao Sang. Mobile-agent: Autonomous multi-modal mobile device agent with visual perception, 2024.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 2609–2634, 2023a.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators, 2023b.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, et al. Autogen: Enabling next-gen llm applications via multi-agent conversations. In *First conference on language modeling*, 2024.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

Ja Eun Yu and Debaleena Chattopadhyay. Reducing the search space on demand helps older adults find mobile ui features quickly, on par with younger adults. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1–22, 2024. doi: 10.1145/3613904.3642796.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yanda Li, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pp. 1–20, 2025. doi: 10.1145/3706598.3713600.

J. Zhang, Xian-Sheng Hua, J. Huang, X. Shen, J. Chen, Q. Zhou, Z. Fu, and Y. Zhao. City brain: practice of large-scale artificial intelligence in the real world. *IET Smart Cities*, 1(1):28–37, 2019.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Yu Zheng, Licia Capra, Ouri Wolfson, and Hai Yang. Urban computing: concepts, methodologies, and applications. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(3):1–55, 2014.

Baichuan Zhou, Haote Yang, Dairong Chen, Junyan Ye, Tianyi Bai, Jinhua Yu, Songyang Zhang, Dahua Lin, Conghui He, and Weijia Li. URBench: A comprehensive benchmark for evaluating large multimodal models in multi-view urban scenarios. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 10707–10715, 2025.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024.