Chat2Map: A ReAct-based Agent Framework for Automated Web Map Generation from Natural Language Instructions

Hongping Zhang [1], Peilong Ma [2],Cong Wang[1], Lei Ding[1], Zhen Wang,[1] Heng Li[1]

[1] National Geomatics Center of China, Beijing, China- (zhanghongping,wangcong, dinglei, wangzhen, liheng)@ngcc.cn
[2] Nanjing Normal University, School of Geography, 210023 Nanjing, Jiangsu, China - mpl_gis@nnu.edu.cn

Keywords: Large Language Models, WebGIS, Geospatial Visualization, ReAct Agent, Code Generation.

Abstract
Web map creation remains difficult for non-specialists because it requires the combined use of geospatial data understanding, cartographic design, front-end development, and platform-specific mapping APIs. Although large language models have recently improved natural-language-driven code generation, they still perform unreliably in web mapping tasks, where failures often arise from hallucinated API usage, heterogeneous input data structures, and the gap between code plausibility and rendered map quality. This paper presents Chat2Map, an agent framework for automated web map generation from natural-language instructions. Rather than treating the task as one-shot text-to-code generation, Chat2Map formulates web map authoring as a grounded workflow that integrates skill and tool grounding, runtime schema construction for uploaded GeoJSON and JSON files, and an iterative generate-execute-diagnose-repair loop based on runtime and visual feedback. To validate the framework, we implemented a working prototype with Tianditu as a reference API setting and evaluated it on 30 web map generation tasks covering both directive-based generation and data-driven visualization. The results show that Chat2Map achieved 80% code executability and a 70% task success rate on directive-based tasks, while also maintaining a 70% task success rate on data-driven tasks where baseline models failed to produce usable outputs. These findings suggest that reliable web map generation depends not only on stronger code models, but also on how model reasoning is connected to domain knowledge, explicit data understanding, and real execution environments.

1 INTRODUCTION

Interactive web maps have become a common medium for publishing, exploring, and communicating spatial information in domains such as public services, urban management, environmental monitoring, and thematic storytelling (Roth, 2013; Netek et al., 2023). Compared with static cartographic products, web maps support zooming, querying, filtering, comparison, and other forms of user interaction, making them increasingly important not only for expert analysis but also for public-facing communication. At the same time, maps are now used not only as analytical interfaces but also as narrative and visual communication media, especially in story-oriented and thematic applications (Caquard and Cartwright, 2014; Roth, 2020). As web technologies and geospatial data infrastructures continue to evolve, more spatial information is expected to be delivered through interactive web-based map applications rather than through static figures alone.

Despite this growing importance, creating a usable web map remains a highly expertise-intensive task. Developers must work across several knowledge domains simultaneously, including geospatial data organization, coordinate handling, thematic symbolization, front-end interaction design, and browser-side implementation. In practice, web map development often requires assembling data services, interface components, and platform-specific map libraries into a single operational application, which increases both technical complexity and development cost (Zavala-Romero et al., 2014; Kong et al., 2015). Moreover, many real tasks require not only displaying geographic features, but also building legends, side panels, pop-up windows, filter controls, and coordinated views so that the final product becomes a meaningful cartographic interface rather than a simple map canvas. As a result, many domain experts who understand spatial problems well are still unable to independently create customized web maps for communication and decision support.

Recent progress in large language models (LLMs) has opened a new possibility for reducing this barrier. Natural-language-driven code generation is improving rapidly, and LLMs are becoming increasingly capable of producing front-end components, data-processing scripts, and complete application prototypes (Ghaemi et al., 2024). This broader trend has encouraged the idea that software systems can be described in natural language and then partially implemented by intelligent assistants. Similar expectations have also begun to appear in geospatial settings, where recent studies have explored natural-language-driven interfaces for spatial analysis and GIS-related tasks (Mansourian and Oucheikh, 2024; Li and Ning, 2023). For web maps, this possibility is especially attractive because many users can clearly describe what they want to show on a map, even if they do not know how to implement it through conventional programming.

However, automated web map generation remains much more challenging than generic web coding. First, geospatial applications depend on domain-specific APIs and service conventions that are often weakly represented in general model training data, which makes hallucinated functions, invalid parameter settings, and incorrect workflow composition more likely in practice (Hou et al., 2025; Chen et al., 2025). Second, web map generation must work with heterogeneous data inputs whose structures are often not immediately obvious to a general-purpose model. A user may upload GeoJSON directly, or provide a JSON file with nested fields, wrapped payloads, and mixed attribute structures. Small misunderstandings in file schema or geometry access paths can therefore cause the generated page to fail or produce misleading visual results. Third, a web map must be judged not only by whether the code compiles, but also by whether the rendered page is meaningful as a map. A result may be technically executable while still suffering from blank basemaps, broken layer configuration, weak thematic hierarchy, or interaction states that do not match the user’s intended task. Recent benchmark and evaluation work has further shown that geospatial code generation remains a difficult setting for current LLMs, especially when domain

correctness and execution reliability are both required (Hou et al., 2025; Chen et al., 2025; Xu et al., 2025).

These limitations suggest that web map authoring should not be treated simply as a one-shot natural-language-to-code task. Instead, it should be formulated as a grounded development workflow in which the model must reason with platform knowledge, uploaded data, external tools, and execution feedback. This view is consistent with the broader shift toward agentic AI, where language models are embedded in reasoning-and-acting loops rather than used as single-pass text generators (Yao et al., 2022; Wang et al., 2024). It also aligns with recent efforts in geospatial AI that connect language models with tools, workflows, and domain constraints for more reliable task completion (Li and Ning, 2023; Akinboyewa et al., 2025). From this perspective, the key problem is not only how to generate code from a request, but how to ensure that the generated map can run in a real environment, interpret user data correctly, and remain aligned with cartographic intent after rendering.

In response, this study presents Chat2Map, an agent framework for automated web map generation from natural-language instructions. Rather than relying on free-form code generation alone, Chat2Map organizes map-development knowledge through skill and tool grounding, converts uploaded GeoJSON and JSON inputs into explicit runtime schemas, and uses an iterative generate-execute-diagnose-repair loop to improve both code validity and rendered map quality. In this design, the model is supported by a bounded capability space instead of being asked to solve the task from raw prompting alone. At the same time, uploaded geospatial data are transformed into interpretable runtime structures so that subsequent generation and debugging are based on explicit evidence rather than guessed field names or access paths. Candidate web map pages are then executed in a real browser environment, where runtime errors and rendered outputs provide feedback for further revision.

The main idea behind Chat2Map is that reliable web map generation depends on the integration of domain grounding, data grounding, and environment grounding. Domain grounding ensures that map-related APIs, references, and tools are used in a controlled and auditable way. Data grounding ensures that uploaded geospatial inputs are understood through explicit runtime schemas before visualization logic is generated. Environment grounding ensures that generated outputs are tested in a real execution setting and can be revised according to observable failures or visual deficiencies. Together, these mechanisms move the task beyond simple code completion and toward a practical workflow for executable and inspectable web map creation.

To demonstrate this idea, we implemented Chat2Map as a working web prototype with Tianditu as a reference API setting. The prototype allows users to describe mapping needs in natural language, upload data, inspect generated code, observe rendered outputs, and iteratively refine results within a unified interaction environment. The remainder of this paper introduces the framework design, reports representative results, and discusses how the proposed approach supports more dependable and shareable web map generation.

2 METHODOLOGY

Chat2Map addresses the above challenges through an agent-oriented workflow that combines reasoning, tool use, execution, and revision in a closed loop. Instead of treating web map authoring as a one-step translation from natural-language instructions to code, the framework formulates the task as a grounded development process in which the model operates with platform knowledge, uploaded data, and environmental feedback (Figure 1). This design follows the broader ReAct paradigm, in which language models alternate between reasoning and action, rather than producing a final answer in a single pass (Yao et al., 2022). It also aligns with the growing literature on LLM-based autonomous agents, which emphasizes planning, external tool invocation, and iterative interaction with task environments as key mechanisms for improving reliability (Wang et al., 2024). In the present study, this workflow is realized through three tightly connected methodological components.

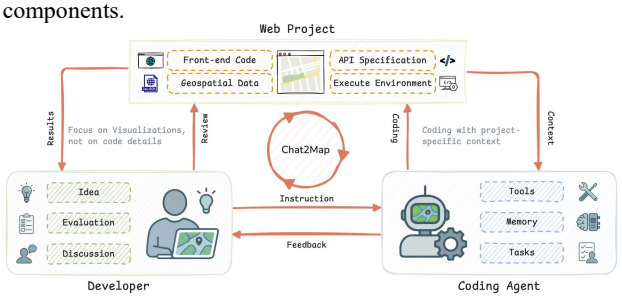

Figure 1. Chat2Map framework architecture showing the interaction between developer and coding agent through instruction-feedback loops. The web project integrates front-end code, geospatial data, API specifications, and execution environment. The ReAct cycle enables iterative refinement where developers focus on visualizations while the agent handles project-specific coding with contextual awareness.

The first component is capability grounding through skills, tools, and platform-specific references. In many existing code-generation settings, the model is expected to infer the correct development strategy directly from the prompt. For web maps, however, this is often unreliable because the task depends on specialized APIs, visual design decisions, and interaction conventions that are weakly represented in general-purpose model training corpora. Chat2Map therefore constructs a bounded capability space in which only the packages relevant to the current task are activated, such as map API usage, interface design, chart integration, or repair operations. These packages provide procedural guidance, reference materials, and executable actions that help the agent reason within a constrained domain instead of searching freely over an unconstrained code space. This strategy is consistent with recent work arguing that tool use and structured orchestration can improve the performance of LLM systems on complex tasks that require external operations or domain-specific actions (Qin et al., 2023). It is also related to recent geospatial agent research, which increasingly treats language models as coordinators of workflows, tools, and contextual resources rather than as standalone text generators (Li and Ning, 2023; Chen et al., 2024; Akinboyewa et al., 2025). In Chat2Map, the model is therefore used as the core of an orchestration process that links user intent, documentation, workspace state, and callable tools into a grounded runtime context for map construction (Figure 2).

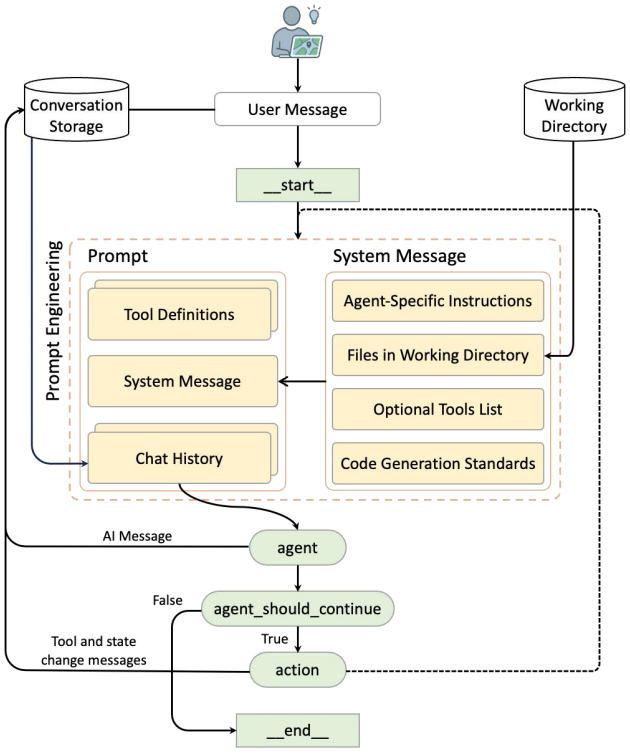

Figure 2. Detailed agent workflow showing prompt engineering components. User messages initiate a cyclic process where the system message (containing tool definitions, agent instructions, working directory files, optional tools, and code generation standards) guides the agent through decision-making (agent_should_continue) and action execution, with conversation storage and working directory maintaining state across iterations.

The second component is schema-aware processing for uploaded geospatial files. Web map generation is strongly affected by how input data are organized, yet uploaded files often expose their useful structure only implicitly. User-provided inputs may arrive as standard GeoJSON, or as JSON objects whose meaningful spatial content is nested inside wrapper fields and payload paths. When a model guesses these structures incorrectly, even otherwise plausible code can fail during loading, layer construction, or interaction binding. To reduce this source of error, Chat2Map analyzes uploaded files before generation and converts them into explicit runtime schemas. These schemas summarize the effective data root, geometry structure, field composition, and candidate attributes for visual encoding, interaction, or filtering. In effect, this step turns heterogeneous data inputs into an explicit intermediate representation that the agent can reuse across generation, debugging, and repair. This design is motivated by recent studies showing that data-structure ambiguity is a major source of failure in geospatial code generation and that current LLMs remain sensitive to file-format and schema misalignment in multi-step spatial tasks (Hou et al., 2025; Xu et al., 2025). By making input structure explicit before map code is generated, Chat2Map reduces dependence on speculative field inference and creates a more stable basis for downstream reasoning.

The third component is an iterative generate-execute-diagnose-repair loop for improving both runtime validity and rendered map quality. After constructing the grounded context and runtime schema, the system generates a candidate web map page and executes it in a real browser environment. The framework then collects two kinds of evidence. The first is runtime feedback, such as JavaScript errors, failed network requests, broken initialization procedures, or missing layers. The second is visual feedback derived from rendered map states, which helps identify problems such as blank basemaps, misplaced layouts, weak visual hierarchy, clipped interface components, or incomplete thematic presentation. Based on these signals, the agent revises the code and re-runs the page until the output becomes executable and reasonably aligned with the intended map expression. This execution-centered strategy is supported by recent work in geospatial evaluation, which suggests that runnable behavior and multimodal inspection are more informative than text-only matching when judging generated spatial code (Chen et al., 2025; Wu et al., 2025). It also reflects an important cartographic consideration: a generated map should not be judged solely by whether it runs, but also by whether the rendered result is coherent as a map interface and supports the requested communicative task.

Taken together, these three components allow Chat2Map to move beyond unconstrained natural-language-to-code generation. The framework instead supports a domain-grounded, schema-aware, and browser-validated workflow for producing web maps that are not only syntactically plausible, but also executable, inspectable, and suitable for further sharing and refinement. In this sense, Chat2Map should be understood less as a generic code model and more as an agent framework for reliable web map authoring under real geospatial constraints.

3 SYSTEM PROTOTYPE

To demonstrate that Chat2Map can function as a practical web map authoring environment rather than only a conceptual framework, we implemented it as a working prototype system. The prototype is designed to connect natural-language interaction, code generation, browser-side execution, and result sharing within one operational workflow. In this way, the system allows users not only to describe mapping requirements, but also to inspect generated code, observe rendered outputs, and iteratively refine the resulting web maps.

The prototype uses Tianditu as a reference API setting for web map generation. This choice does not mean that the framework is limited to a single platform. Rather, Tianditu provides a concrete test environment in which platform-specific map services, API calls, and interaction logic can be grounded and executed. Through this implementation, Chat2Map serves as a system-level realization of the methodological ideas presented in the previous section, especially the integration of capability grounding, schema-aware data understanding, and execution-based repair.

3.1 System architecture

The system architecture of Chat2Map is organized around three connected parts: the user workspace, the system backend, and the agent orchestration layer (Figure 3). These parts jointly support the end-to-end process from user request to runnable and shareable web map output.

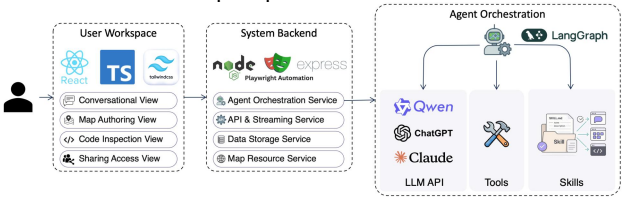

Figure 3. System architecture of the Chat2Map prototype. The system connects a browser-based user workspace, a backend service layer, and a LangGraph-based agent orchestration layer. The user workspace supports conversation, map authoring, code inspection, and sharing access; the backend coordinates request handling, storage, streaming, and browser automation; and the agent orchestration layer integrates LLM APIs, tools, and skills to support grounded web map generation and iterative repair.

The user workspace is the front-end environment through which users interact with the system. It is implemented with React and TypeScript and provides multiple coordinated views for conversation, map authoring, code inspection, and sharing access. Through this workspace, users can describe map requirements in natural language, upload data files, inspect intermediate outputs, and continue refinement through follow-up prompts. This design reduces the need to switch repeatedly between separate tools such as chat interfaces, browsers, code editors, and sharing platforms.

The system backend is responsible for connecting user interactions with the generation and execution pipeline. Implemented with Node.js and Express, the backend manages agent orchestration services, API and streaming services, data storage, and map resource access. It acts as the operational layer that receives user input, stores intermediate artifacts, coordinates model calls, and returns generated outputs to the user interface. In addition, the backend supports browser automation through Playwright, allowing generated map pages to be executed and inspected in a real browser environment rather than being evaluated only at the text level.

The agent orchestration layer serves as the core coordination mechanism of the system. It links multiple LLM backends, tools, and skill packages through a LangGraph-based control flow. Within this layer, the system can select appropriate capabilities according to the current task, assemble contextual information, invoke external tools, and trigger iterative repair when execution or rendering problems occur. This architecture reflects the central idea of Chat2Map: reliable web map generation depends not on a single code model alone, but on the coordinated interaction among models, tools, skills, and runtime feedback. Figure 3 therefore illustrates how the prototype transforms the methodological framework into an operational system for real web map development.

3.2 User interface and interaction flow

The user-facing prototype is designed as an integrated authoring environment that supports the full workflow of map generation, inspection, and sharing (Figure 4). Rather than exposing only a single map-editing screen, the interface includes a landing page, an authoring workspace, and a public gallery, together forming a continuous interaction flow for users with different needs.

The landing page provides the entry point to the system and communicates its main purpose: generating professional web map applications from natural-language instructions. This page lowers the initial barrier to entry by presenting the system as a creation environment rather than as a conventional development interface. Once users enter the authoring workflow, they are brought to the main workspace, where most of the core interaction takes place.

The integrated authoring workspace combines several functions in one screen. Users can enter natural-language instructions, optionally upload data, inspect execution traces, view the rendered map, and examine the generated code. This arrangement is important because web map development typically requires repeated movement between description, generation, execution, and debugging. By placing conversation, rendering, and code side by side, the prototype makes the generation process more transparent and supports incremental refinement. In the example shown in Figure 4, the user request, generated interface, rendered map result, and code output are all visible within the same workspace, allowing the user to understand how the system is translating a requirement into an executable web map application.

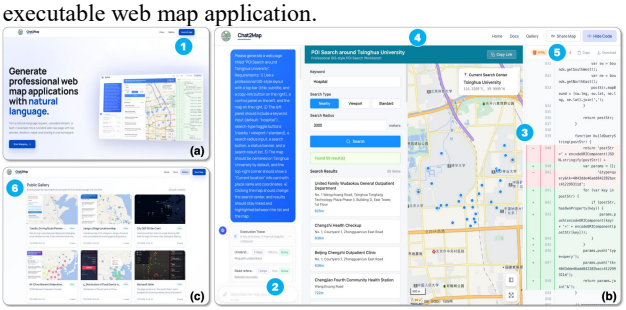

Figure 4. Main user-facing interfaces and interaction flow of the Chat2Map prototype. The figure shows (a) the landing page, (b) the integrated authoring workspace, and (c) the public gallery. The numbered markers indicate a typical user workflow: entering the creation process, providing map requirements and optional data, generating code, rendering and inspecting the map, sharing the generated artifact, and browsing published outputs in the gallery.

Figure 4 also indicates a typical interaction sequence supported by the prototype. Users first enter the creation workflow from the landing page. They then provide a map request and, when necessary, upload data files. The system generates code, renders the candidate web map, and returns both the interactive output and the corresponding implementation to the workspace. After reviewing the result, users can share the generated artifact through the built-in sharing function. Published outputs can then be browsed in the public gallery, which serves as a repository of generated map applications and helps demonstrate the shareability of the overall workflow.

The gallery is therefore not merely a visual supplement, but an important part of the prototype design. It extends Chat2Map beyond one-off generation by allowing completed maps to be preserved, revisited, and shared with others. This feature supports the broader goal of turning web map generation into a practical and reusable authoring process rather than a transient interaction with a language model.

4 RESULTS

4.1 Overall performance

We evaluated Chat2Map on 30 web map generation tasks covering two major settings: directive-based generation and data-driven visualization. The evaluation focused on whether the generated outputs could run successfully, whether they satisfied the requested mapping task, and whether the final results were usable as interactive web maps. In other words, the assessment was not limited to code-level correctness alone, but also considered whether the final product functioned as a meaningful web map application. Overall, the results indicate that Chat2Map can substantially improve the reliability of

natural-language-driven web map creation in comparison with unconstrained code generation baselines.

For directive generation tasks, Chat2Map achieved 80% code executability and a 70% task success rate. These tasks required the system to interpret user instructions that described layout structure, map composition, interaction logic, and thematic intent without depending primarily on uploaded datasets. In this setting, one of the main challenges for baseline models was the frequent production of invalid or hallucinated API calls, especially when the requested functionality depended on platform-specific map operations. Chat2Map reduced this problem by grounding generation in platform-related skills, tool support, and structured execution feedback, thereby improving both the syntactic validity and the practical usability of the produced code. This result suggests that even in tasks driven mainly by textual instructions, bounded capability organization can make web map generation more stable and dependable.

For data-driven visualization tasks, Chat2Map also achieved a 70% task success rate, while the compared baseline models failed to complete these tasks successfully. This difference is especially important because data-driven web map construction is not only a coding problem, but also a data interpretation problem. The system must identify how uploaded files are structured, determine valid geometry and attribute access paths, and then translate that understanding into appropriate map layers, visual encodings, and interactive behaviors. The results suggest that schema-aware preprocessing and grounded generation are particularly valuable in this setting, where small misunderstandings in file structure can easily lead to broken rendering, missing features, or logically incorrect visual outputs. In this sense, the performance gap between Chat2Map and the baselines highlights the importance of explicit data understanding in automated geospatial application development.

In addition to the aggregate statistics, the generated cases show that Chat2Map can support a wide range of geospatial visualization scenarios. As illustrated in Figure 5, the framework was able to produce different types of map applications, including innovation evaluation maps for the Yangtze River Delta, classified visualizations of urban underground pipe networks, navigation route maps between campuses, province-level annual temperature change maps, demolition-site information maps, tourist route maps, public service center distribution maps, and flood-disaster visualization pages. These examples indicate that the framework is not limited to one narrow task type. Instead, it can handle both instruction-driven and data-driven requests, and can support multiple forms of geospatial expression, including point distribution, route depiction, thematic comparison, and information-rich interactive presentation. The diversity of these outputs also suggests that the framework is capable of adapting to different combinations of map content, interface layout, and interaction requirements.

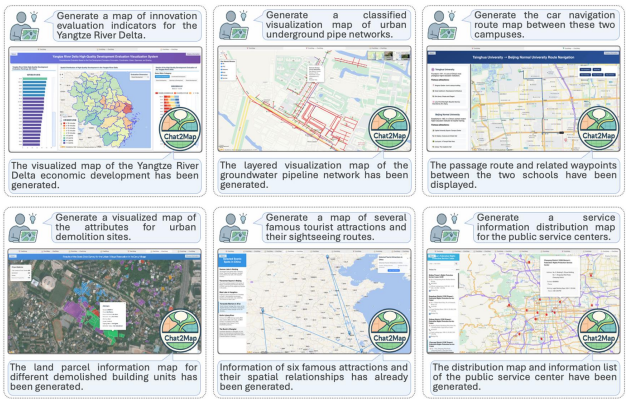

Figure 5. Representative examples of maps generated by Chat2Map across diverse application scenarios. Top row (left to right): Yangtze River Delta innovation evaluation indicators, classified urban underground pipe network visualization, car navigation route between two campuses, annual average temperature changes across provinces. Bottom row: urban demolition site attributes map, tourist attraction sightseeing routes, public service center distribution map, flood disaster information visualization. Each example demonstrates the framework's capability to interpret natural language instructions and generate professional-grade interactive maps.

4.2 Representative application

A representative directive-based example is the “POI Search around Tsinghua University” application generated by Chat2Map. In this task, the user requested a professional GIS-style search interface with a top title bar, a left-side control panel, a central map view, and a code panel for inspection. The system needed to translate this textual description into a complete interactive web map page that supported keyword-based POI retrieval, search-type switching, radius input, result listing, and synchronized map updates. Chat2Map generated a runnable interface in which the search controls, map display, and result cards were coherently organized within a unified layout. The map was centered on the target area, retrieved POI points were rendered correctly, and the result list remained linked with the map view. This example illustrates that Chat2Map can handle not only map rendering itself, but also the surrounding interface logic required for a usable web map application. It therefore provides a concrete demonstration of how the framework supports directive-based generation under platform-specific API constraints.

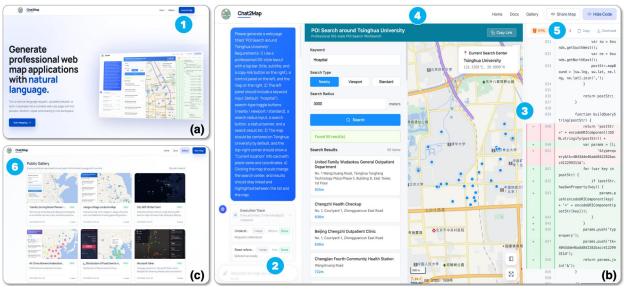

Figure 6. A representative directive-based web map application generated by Chat2Map. Given the request “POI Search around Tsinghua University,” the system produced a GIS-style search

interface with a left-side control panel, a central interactive map, a linked result list, and an inspectable code panel. The example demonstrates that Chat2Map can support not only map rendering itself, but also the surrounding interface logic and platform-specific interaction design required for a usable web map application.

4.3 Ablation and cross-model analysis

We also conducted ablation experiments to examine the contribution of key components in the framework. When the constraint mechanisms were removed, the task success rate dropped sharply to 36.7%, indicating that bounded capability organization and structured control are central to reliable map generation. Replacing the tuned coding model with a base model also reduced performance, lowering task success to 63.3%. These results suggest that Chat2Map does not depend on a single factor alone. Instead, its effectiveness comes from the combination of model capability, domain-specific grounding, structured data understanding, and iterative repair. The ablation results therefore support the design rationale of the framework rather than attributing performance gains to one isolated module.

To further test the generality of the framework, we examined its behavior with multiple large language models, including GPT-4, Claude 3, Llama 3, and Qwen-Max. All tested models achieved success rates above 70% within the Chat2Map workflow. This result suggests that the framework is not tied to one proprietary model family. Rather, it provides a transferable orchestration logic that can improve web map generation across different underlying model backends, as long as they can participate in the grounded generation and feedback-repair loop. This cross-model consistency is important because it indicates that the proposed framework may remain useful even as the underlying model ecosystem continues to evolve.

Taken together, these findings show that Chat2Map is effective not only in improving code executability, but also in supporting more dependable end-to-end web map production. The results support the main claim of this study: automated web map generation becomes substantially more reliable when natural-language interaction is combined with domain grounding, schema-aware data handling, and iterative execution-based repair.

5 CONCLUSIONS

This study presented Chat2Map, an agent framework for automated web map generation from natural-language instructions. Rather than treating web map creation as a simple text-to-code problem, Chat2Map organizes the task as a grounded workflow that combines domain-specific skills and tools, schema-aware understanding of uploaded geospatial data, and iterative repair based on runtime and visual feedback. In this way, the framework supports the generation of web maps that are not only syntactically plausible, but also executable, inspectable, and suitable for further refinement.

The results suggest that this workflow can improve the reliability of automated web map development across both directive-based and data-driven tasks. By constraining model behavior through platform knowledge, making uploaded GeoJSON and JSON inputs explicitly interpretable, and validating outputs in a real browser environment, Chat2Map reduces common failures such as hallucinated API usage, incorrect data access, and visually incomplete map states. The system prototype and representative examples further demonstrate that the framework can support a range of practical web mapping scenarios rather than only narrow template-style tasks.

Overall, Chat2Map shows that progress in automated geospatial application development depends not only on stronger language models, but also on how those models are connected to domain knowledge, executable tools, and real runtime environments. This work provides a practical step toward more accessible and shareable WebGIS authoring, and suggests a promising direction for future research on agentic geospatial systems.

References

Akinboyewa, T.; Li, Z.; Ning, H.; Lessani, M. N., 2025: GIS Copilot: towards an autonomous GIS agent for spatial analysis. International Journal of Digital Earth, 18(1), 2497489. Available at: https://doi.org/10.1080/17538947.2025.2497489.

Caquard, S.; Cartwright, W., 2014: Narrative cartography: From mapping stories to the narrative of maps and mapping. The Cartographic Journal, 51(2), 101-106. Available at: https://doi.org/10.1179/0008704114Z.000000000130.

Chen, G.; Jiao, H.; Hou, S.; Liu, Z.; Xie, L.; Wu, S.; Wu, H.; Guan, X.; Gui, Z., 2025: GeoJSEval: An automated evaluation framework for large language models on JavaScript-based geospatial computation and visualization code generation. ISPRS International Journal of Geo-Information, 14(10), 382. Available at: https://doi.org/10.3390/ijgi14100382.

Chen, Y.; Wang, W.; Lobry, S.; Kurtz, C., 2024: An LLM agent for automatic geospatial data analysis. arXiv preprint arXiv:2410.18792. Available at: https://arxiv.org/abs/2410.18792.

Ghaemi, H.; Alizadehsani, Z.; Shahraki, A.; Corchado, J. M., 2024: Transformers in source code generation: A comprehensive survey. Journal of Systems Architecture, 153, 103193. Available at: https://doi.org/10.1016/j.sysarc.2024.103193.

Hou, S.; Shen, Z.; Liang, J.; Jiao, H.; Zhao, A.; Qing, Y.; Peng, D.; Gui, Z.; Guan, X.; Xiang, L.; Wu, H., 2025: Can large language models generate geospatial code? Geo-spatial Information Science, 1-35. Available at: https://doi.org/10.1080/10095020.2025.2535523.

Kong, N.; Zhang, T.; Stonebraker, I., 2015: Evaluation of web GIS functionality in academic libraries. Applied Geography, 60, 288-293. Available at: https://doi.org/10.1016/j.apgeog.2014.11.017.

Li, Z.; Ning, H., 2023: Autonomous GIS: the next-generation AI-powered GIS. International Journal of Digital Earth, 16(2), 4668-4686. Available at: https://doi.org/10.1080/17538947.2023.2278895.

Mansourian, A.; Oucheikh, R., 2024: ChatGeoAI: Enabling geospatial analysis for public through natural language, with large language models. ISPRS International Journal of Geo-Information, 13(10), 348. Available at: https://doi.org/10.3390/ijgi13100348.

Netek, R.; Pohankova, T.; Bittner, O.; Urban, D., 2023: Geospatial analysis in web browsers-comparison study on WebGIS process-based applications. ISPRS International

Journal of Geo-Information, 12(9), 374. Available at: https://doi.org/10.3390/ijgi12090374.

Qin, Y.; Liang, S.; Ye, Y.; Zhu, K.; Yan, L.; Lu, Y.; Lin, Y.; Cong, X.; Tang, X.; Qian, B.; Zhao, S.; Hong, L.; Tian, R.; Xie, R.; Zhou, J.; Gerstein, M.; Li, D.; Liu, Z.; Sun, M., 2023: ToolLLM: Facilitating large language models to master 16000+ real-world APIs. arXiv preprint arXiv:2307.16789. Available at: https://arxiv.org/abs/2307.16789.

Roth, R. E., 2013: Interactive maps: What we know and what we need to know. Journal of Spatial Information Science, 6. Available at: https://doi.org/10.5311/JOSIS.2013.6.105.

Roth, R. E., 2020: Cartographic design as visual storytelling: Synthesis and review of map-based narratives, genres, and tropes. The Cartographic Journal, 58(1), 83-114. Available at: https://doi.org/10.1080/00087041.2019.1633103.

Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; Zhao, W. X.; Wei, Z.; Wen, J., 2024: A survey on large language model based autonomous agents. Frontiers of Computer Science, 18(6). Available at: https://doi.org/10.1007/s11704-024-40231-1.

Wu, H.; Shen, Z.; Hou, S.; Liang, J.; Jiao, H.; Qing, Y.; Zhang, X.; Li, X.; Gui, Z.; Guan, X.; Xiang, L., 2025: AutoGEEval: A multimodal and automated evaluation framework for geospatial code generation on GEE with large language models. ISPRS International Journal of Geo-Information, 14(7), 256. Available at: https://doi.org/10.3390/ijgi14070256.

Xu, L.; Zhao, S.; Lin, Q.; Chen, L.; Luo, Q.; Wu, S.; Ye, X.; Feng, H.; Du, Z., 2025: Evaluating large language models on geospatial tasks: A multiple geospatial task benchmarking study. International Journal of Digital Earth, 18(1). Available at: https://doi.org/10.1080/17538947.2025.2480268.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K.; Cao, Y., 2022: ReAct: Synergizing reasoning and acting in language models. arXiv preprint arXiv:2210.03629. Available at: https://arxiv.org/abs/2210.03629.

Zavala-Romero, O.; Ahmed, A.; Chassignet, E. P.; Zavala-Hidalgo, J.; Fernandez Eguiarte, A.; Meyer-Baese, A., 2014: An open-source Java web application to build self-contained web GIS sites. Environmental Modelling & Software, 62, 210-220. Available at: https://doi.org/10.1016/j.envsoft.2014.08.029.