The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume L-4/W1-2026
Free and Open Source Software for Geospatial (FOSS4G) 2026 – Academic Track "Bridging Geospatial Technology and Humanity",
30 August–5 September 2026, Hiroshima, Japan

Towards Automated Map JSON Style from Spatial Vector Data Using MCP

Arissara Sompita

Platform Engineering Support, ELEMNT Earth, 252 SPE Tower, Phahonyothin Rd., Phaya Thai, Bangkok 10400, Thailand – arissara.sompita@gmail.com

Keywords: Map Style JSON, MapLibre, Model Context Protocol, Large Language Models, Ollama, Cartographic Design

Abstract

GIS data is inherently multi-dimensional, involving space, time, and attributes, and interpreting it usually requires considerable time and expertise. Many users of web-mapping platforms struggle at the visualization stage, where they must interpret the data, choose appropriate visualization methods, and define map elements such as size, colour, symbols, transparency, and overall composition. They must also understand map rendering through Map Style JSON, which typically demands significant technical knowledge and design experience. To reduce these barriers, we propose a system that combines the Model Context Protocol (MCP) with open Large Language Models (LLMs) served locally through Ollama to automatically generate Map Style JSON conformant with the MapLibre Style Specification directly from vector-based spatial data. The system is built around purpose-built MCP tools that inspect the data, build and validate data-driven styling expressions, and are designed to embed cartographic design principles such as semantic colour selection and the perceptually grounded use of visual variables. In a preliminary evaluation on consumer hardware, using a 77-polygon GeoJSON dataset and a vector tile service, the system recoloured a map's fill from a plain-language request and returned a style that MapLibre rendered without manual correction. The evaluation also shows that the local open model acts mainly as a natural-language front-end, while correctness and renderer compatibility depend on the MCP tool layer. Relying on open models running locally, the approach aligns with the open geospatial ecosystem and integrates with other open-source tools.

1. Introduction

Geospatial data underpins decision-making across domains as varied as urban planning, environmental monitoring, transportation, and public health. Yet the value of such data is only realised when it is communicated effectively, and communication depends on visualization. GIS data is inherently multi-dimensional: every feature combines spatial geometry, one or more temporal references, and a set of thematic attributes. Turning this multi-dimensional record into a clear, truthful map is a non-trivial task that has traditionally been the province of trained cartographers and GIS specialists.

For the growing number of non-expert users who publish maps on the web, the visualization stage is often the hardest part of the workflow. They must first interpret their data and decide what story it should tell, then choose an appropriate visualization method, and finally translate that intent into concrete map elements: the size of symbols, the choice of colour hue and value, the use of transparency, the selection of icons, and the overall composition of the map. Each of these decisions draws on cartographic knowledge that most users do not possess.

A further obstacle is technical. Modern open web-mapping libraries such as MapLibre GL JS render maps from a declarative style document, the Map Style JSON, which specifies what data to draw, in what order, and with which paint and layout properties. Authoring this document by hand requires understanding its nested structure, the data-driven expression syntax, and the way styling properties bind to feature attributes. For users whose expertise lies in their own domain rather than in cartography or software, this represents a steep and often prohibitive learning curve.

These challenges can be reduced with tools that deeply understand the user's data and are capable of automatically designing map visualizations. In this paper we propose such a tool: a Model Context Protocol (MCP) server combined with open Large Language Models (LLMs) running locally through Ollama, which automatically generates Map Style JSON compliant with the MapLibre Style Specification directly from vector-based spatial data. The contribution is twofold. First, the system automates the production of valid, ready-to-use MapLibre styles from a user's data and a natural-language request. Second, it is designed to embed cartographic design principles—semantic colour selection, perceptually grounded use of visual variables, support for multi-class datasets, and modern colour palettes—so that the resulting styles aim to be not merely specification-valid but also legible and cartographically sound. Because the architecture uses open models and can run entirely on local infrastructure through Ollama, it is consistent with the values of the open geospatial ecosystem and integrates naturally with other open-source tools.

The remainder of this paper is organised as follows. Section 2 reviews related work on automated map styling, the MapLibre Style Specification, the Model Context Protocol and open LLMs, and cartographic design principles. Section 3 describes the methodology and system architecture. Section 4 reports the implementation. Sections 5 and 6 present experiments and conclusions.

2. Related Work and Background

2.1 Automated Map Styling

Automating cartographic design has been a long-standing ambition of the discipline, and the recent rise of geospatial artificial intelligence (GeoAI) has renewed interest in the topic. In a systematic review of the field, Kang et al. (2024) show that GeoAI can accelerate previously complex cartographic design tasks including symbolization, generalization, and colour assignment and can enable forms of cartographic creativity that were not previously practical, while also raising ethical questions that the community must address. Their synthesis makes clear that the automation of styling decisions, as opposed to data processing alone, is an active and maturing research frontier.

The arrival of large language models has accelerated this trend by allowing users to express mapping intent in natural language. Zhang et al. (2024) propose MapGPT, an autonomous framework that treats a map as an integration of distinct elements each handled by a dedicated tool and uses an LLM to orchestrate those tools from conversational input, allowing users to adjust elements such as colour and position through dialogue. This decomposition of map-making into tool-mediated steps coordinated by an LLM is closely related to the approach we adopt, but MapGPT targets general map production with proprietary models, whereas our work focuses specifically on

generating MapLibre-conformant Style JSON for vector data and is designed to run on open models served locally. Our system therefore differs in its emphasis on an open, standards-compliant output artifact and a locally deployable model backend.

2.2 MapLibre Style Specification

MapLibre GL JS is an open-source TypeScript library that renders interactive maps from vector tiles in the browser using GPU-accelerated WebGL rendering (MapLibre Contributors, 2024). It originated as a community fork of mapbox-gl-js after that project moved to a non-open-source licence, and it is distributed under a permissive BSD licence, which has made it a foundation for open web mapping.

The appearance of a MapLibre map is governed by the MapLibre Style Specification, a declarative format in which a style is a JSON object defining the data sources to draw, the order in which layers are rendered, and the paint and layout properties applied to each layer. A central feature of the specification is its support for data-driven expressions, which allow styling properties to be computed as functions of feature attributes and zoom level. This expression language is what makes rich thematic cartography possible in MapLibre, but it is also the part of the specification that is hardest for non-experts to author by hand. Our system targets this specification directly, generating both valid layer definitions and the expressions that bind styling to data, so that the output can be consumed without modification by any MapLibre-based application.

2.3 MCP and Open LLMs (Ollama)

The Model Context Protocol (MCP) is an open standard, introduced by Anthropic (2024), for connecting LLM applications to external data and tools in a uniform way. MCP addresses the combinatorial "M×N" integration problem connecting many models to many tools by defining a common interface built on JSON-RPC, in which servers expose primitives such as tools (actions the model can invoke), resources (read-only context), and prompts (reusable templates), and communicate with hosts either over standard input/output for locally spawned servers or over HTTP for remote ones. Because the protocol is model-agnostic, a single MCP server can be driven by different LLM backends without modification, which is the property we exploit to keep our system open and portable.

Complementing MCP, Ollama is an open-source runtime that makes it straightforward to serve open-weight LLMs locally. It manages model downloading, quantization, and GPU offloading, and exposes an OpenAI-compatible HTTP interface, allowing applications written against common LLM SDKs to be redirected to a local model with minimal change. Running inference locally provides data sovereignty, avoids per-call API costs, and enables offline operation properties that are particularly valuable for geospatial workflows that may involve sensitive or proprietary data. Together, MCP and Ollama allow our system to combine a standardised tool interface with locally hosted open models, so that the entire pipeline can run within an organisation's own infrastructure.

2.4 Cartographic Design Principles

For automatically generated styles to be useful, they must respect the established principles that make maps legible. The theoretical foundation for these principles is Bertin's system of visual variables (Bertin, 1983), which identified the graphic dimensions position, size, shape, value, hue, orientation, and texture—through which data can be encoded, and distinguished those suited to ordered or quantitative data from those suited to qualitative differences. MacEachren (1995) extended this work into a perceptual and cognitive account of how maps are read, providing a framework, grounded in visual perception, for choosing symbolization that matches the nature of the data and the reader's perceptual capabilities. These works establish, for example, that colour value and size convey magnitude, whereas colour hue is best reserved for categorical distinctions.

Colour selection in particular has a well-developed practical literature. Harrower and Brewer (2003) introduced ColorBrewer, a tool and an underlying set of palettes that organise colour schemes into sequential schemes for ordered data, diverging schemes that emphasise a critical mid-point, and qualitative schemes for categorical data, with attention to the number of classes and to constraints such as colour-blind safety. These principles directly inform our system: the choice between sequential, diverging, and qualitative palettes is driven by the semantics and distribution of the attribute being mapped, and the assignment of specific hues—such as blue for water bodies or graduated greens for vegetation—follows widely understood conventions. By encoding this body of knowledge into the rules that guide the LLM, the system aims to produce styles that are not only specification-valid but also cartographically sound.

3. Methodology and System Architecture

3.1 System Overview

The proposed system is organised into four zones that together turn a natural-language request and a vector dataset into a rendered map (Figure 1). The browser zone hosts the user-facing application; the server zone bridges the browser and the language model tooling; the MCP server zone performs the data interpretation and style generation; and the external zone provides the data, models, and tiles that the system consumes.

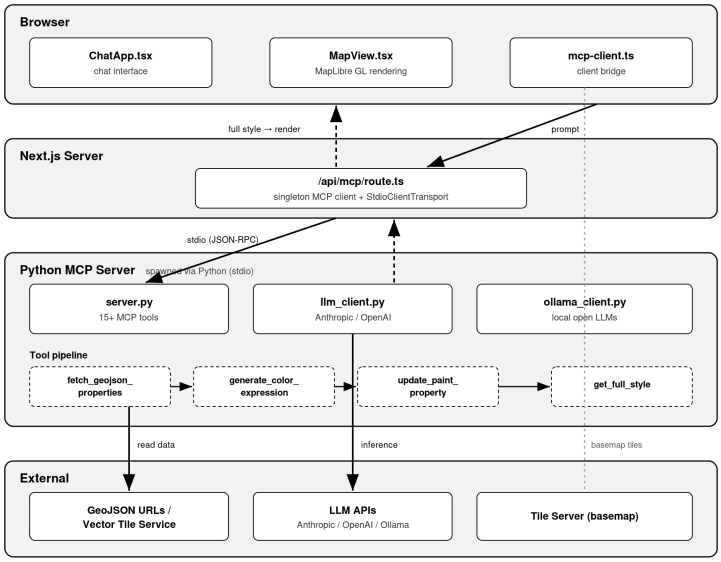

Figure 1. System architecture, showing the browser, Next.js server, Python MCP server, and external zones, and the request flow that turns a prompt into a rendered MapLibre style.

In the browser zone, a chat component (ChatApp) captures the user's request and displays the conversation, while a map component (MapView), built on MapLibre GL, renders the resulting style. A thin client module (mcp-client) mediates communication between the interface and the server. In the Next.js server zone, an API route maintains a singleton MCP client that connects to the MCP server over a standard input/output transport (StdioClientTransport); keeping a single long-lived client avoids the cost of repeatedly spawning the server process. The Python MCP server is the core of the system: it is launched as a spawned Python process communicating over stdio, and it exposes more than fifteen tools through which the model inspects data and constructs styles. The server is model-agnostic, with client modules that allow it to be driven by Anthropic, OpenAI, or local Ollama models. Finally, the external zone comprises the GeoJSON URLs and vector tile services that supply the spatial data, the LLM APIs that provide inference, and the tile server that supplies basemap tiles.

A typical request flows through the system as follows: the user's prompt is passed to the LLM, which selects an appropriate tool; the system first inspects the data's properties, then generates a styling expression, applies it to the relevant layer, and finally retrieves the complete style, which is returned to the frontend for MapLibre to render.

3.2 MCP Tools

The MCP server exposes its functionality as a set of discrete tools that the LLM invokes in sequence. Conceptually these tools fall into five roles—inspection, generation, update, validation, and retrieval—illustrated here by the tools fetch_geojson_properties, generate_color_expression, update_paint_property, validate_style, and get_full_style. These names are illustrative: the prototype exposes a larger set of more than fifteen finer-grained tools.

A data-inspection step precedes styling: a tool reads the attribute schema and value distributions of the supplied GeoJSON or vector-tile source, so that subsequent decisions are grounded in the actual data rather than in assumptions. A generation tool then produces the styling artifact—most importantly a data-driven colour expression—based on the inspected properties and the cartographic rules described below. An update tool applies the generated property to the appropriate layer, modifying the working style in place. A validation tool checks the resulting document against the MapLibre Style Specification so that only specification-conformant output is returned. A final retrieval tool returns the complete style document to the client. Decomposing the task into small, composable tools allows the LLM to reason about one decision at a time and makes each step independently testable.

3.3 Spatial Data Interpretation

Because high-quality styling depends on understanding the data, the system reads directly from the feature properties of GeoJSON sources and vector tiles. For each candidate attribute it determines the data type and the distribution of values, distinguishing categorical attributes (a bounded set of labels, such as land-use classes) from continuous attributes (numeric ranges, such as population density or vegetation index). This characterisation determines the family of visualization that is appropriate and, in turn, the kind of MapLibre expression that must be generated—for example, a match expression for categorical data or an interpolation expression for continuous data. Handling multiple classification classes explicitly is a design requirement of the system, since many real datasets carry several thematic dimensions.

3.4 Cartographic Rules and Colour Selection

The styling decisions made by the system are governed by an encoded set of cartographic rules derived from the principles reviewed in Section 2.4. The selection of a colour scheme is driven by the semantics and statistical character of the attribute: continuous, ordered attributes are mapped with sequential palettes; attributes with a meaningful mid-point are mapped with diverging palettes; and categorical attributes are mapped with qualitative palettes, following the ColorBrewer framework (Harrower and Brewer, 2003). Semantic conventions are applied where they exist—blue hues for water, graduated greens for vegetation—so that the map agrees with readers' expectations. Beyond colour, the rules govern transparency, symbol and icon choice, and the assignment of visual variables to data so that magnitude is encoded through value and size and categories

through hue, consistent with Bertin (1983) and MacEachren (1995). The system supports the principal MapLibre rendering types—fill, circle, line, heatmap, and 3-D extrusion—and selects among them according to geometry type and the user's stated intent.

4. Implementation

The implementation proceeded in five stages, moving from study of the underlying technologies to a working, testable system.

The first stage was a study of the relevant open-source tools Ollama for local model serving and MapLibre for rendering, among others together with a detailed study of the structure of MapLibre expressions and of how data is bound to and rendered on the map. Understanding the expression language was essential, since it is the mechanism through which thematic styling is achieved and therefore the primary artifact the system must generate correctly.

The second stage was the design of the overall system, including the System Architecture described in Section 3 and the division of responsibilities among the browser, server, MCP server, and external zones.

The third stage was the design of the MCP and its tools. This required encoding an understanding of the MapLibre expression structure into the tools, implementing the ability to read attributes from the properties of GeoJSON data and vector tiles, and packaging the server as a spawned Python process communicating over standard input/output so that it could be driven uniformly by different model backends.

The fourth stage was the development of the chat user interface, so that users could issue requests in natural language and immediately see the resulting design reflected on the map. Coupling the conversation directly to the rendered output provides the feedback loop that makes iterative refinement possible.

The fifth stage was testing the system against representative commands. Tests used real data delivered both as GeoJSON URLs and through a vector tile service, and exercised the system's ability to generate colours in a requested tone together with appropriate transparency, and to switch the rendering representation among fill, circle, heatmap, extrusion, and line types. These tests confirmed that the system could take a dataset and a natural-language request and return a valid, cartographically appropriate MapLibre style.

5. Experiments and Results

To assess whether the system can produce valid and cartographically appropriate MapLibre styles from natural-language requests, it was exercised on representative vector datasets and a set of styling commands. This section describes the experimental setup and reports the observed results.

5.1 Experimental Setup

The experiments were conducted entirely on commodity consumer hardware to demonstrate that the system can run locally without specialised infrastructure. The test machine was a MacBook Pro 13-inch (2022) with an Apple M2 chip and 16 GB of unified memory, running macOS Sequoia 15.6.1. Language-model inference was served locally through Ollama using the open-weight qwen2.5-coder model, which was selected for its strength in generating and reasoning about structured, code-like output such as MapLibre Style JSON and data-driven expressions. Running the model on unified memory of this size confirms that the approach is practical on a single laptop rather than requiring a dedicated GPU server.

Two categories of spatial data were used as input. The first was a GeoJSON dataset of 77 polygon features representing the provincial boundaries of Thailand, used to test attribute-driven styling and classification. The second was a vector tile service, served from the Vallaris Maps platform operated by GISTDA, containing more than 300 features, used to test the system's ability to read feature properties from tiled sources rather than from a single GeoJSON document. Together these inputs cover the two principal ways in which vector data reaches a MapLibre map. Both were loaded into a prototype chat interface ("Ellen", a map-style assistant), which reported the number of features loaded and the available source layers before any styling command was issued.

The evaluation focused on a natural-language command representative of a common styling task: a request to recolour the polygon fill (for example, "Change data to pink color"), optionally with adjusted transparency. This command exercises the colour-selection logic (Section 3.4) together with the data-interpretation step that identifies the layer geometry before styling (Section 3.3), and requires the system to resolve the target layer and write a valid paint property that MapLibre can render. A more demanding command—classifying the polygon features by the region attribute so that provinces are grouped and coloured by administrative region, which requires generating a match-style expression keyed on a categorical attribute—was also targeted; as reported below, making this reliable across the tool layer proved more involved and is not yet complete.

5.2 Results

Overall, the results were modest: the locally hosted open model performed less well than expected when asked to drive styling directly. It could interpret intent reasonably—recognising, for instance, that "Change data to pink color" referred to a colour edit—but on its own it could not reliably produce output that was geometry-aware and schema-compliant with MapLibre GL JS, and unaided attempts frequently yielded invalid or unusable styles. As a result, what the user perceived as the model "styling the map" was in practice carried out by a layer of purpose-built MCP tools that compensated for these limitations. These tools handled the deterministic work: inspecting the vector data to identify its geometry type, resolving the target layer and paint property, building or validating the styling expression, and writing the result into a Style JSON document that MapLibre GL JS could render without manual correction. Notably, each capability had to be implemented as a dedicated tool, and achieving even modest reliability required a substantial amount of tool logic: the recolour operation was implemented and worked, whereas more complex operations such as classifying features by a categorical attribute proved considerably harder to make reliable and were not completed in the present prototype.

The recolour task is a representative example. Given the request "Change data to pink color," the system recognised the GeoJSON layer as polygonal, updated the fill-color property of the polygon layer (named data-fill) to pink (#FFC0CB), and confirmed the change in the chat. The 77-province dataset, initially in its default green styling (Figure 2, before), was recoloured to a uniform pink fill while retaining its boundaries (Figure 3, after). Although this simple case completed successfully, it did so only because the supporting tools carried out the operation; the language model itself contributed little beyond intent recognition.

In summary, the open-weight LLM functioned mainly as a natural-language front-end, while correctness, geometry-awareness, and renderer compatibility depended almost entirely on the surrounding MCP tool layer. The relatively weak standalone performance of the local model meant that the principal engineering effort shifted from prompt design toward building robust, well-scoped MCP tools—an important practical limitation to consider when reproducing this approach with locally hosted language models.

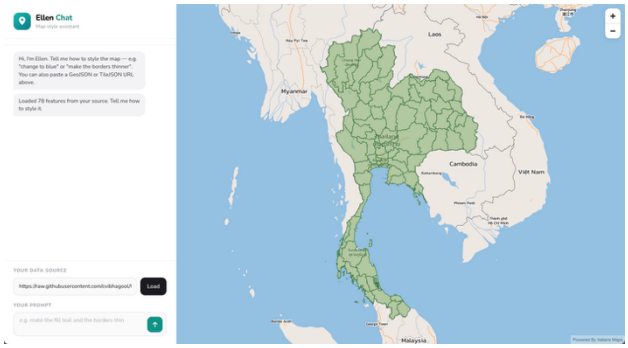

Figure 2. Default (before) styling of the 77-province GeoJSON dataset.

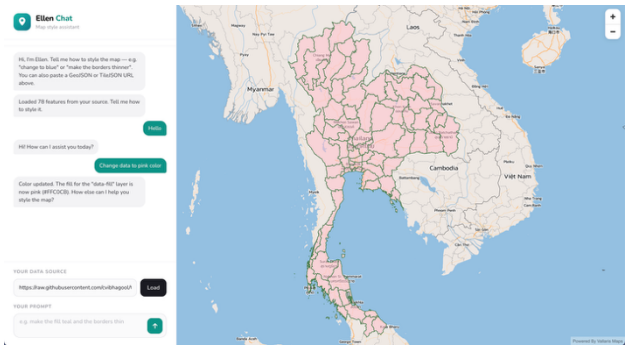

Figure 3. Result (after) of the "Change data to pink color" command.

5.3 Discussion

The experiments indicate that the proposed system can turn a plain-language request into a valid, render-ready MapLibre style for a simple styling task while running entirely on a single consumer laptop. Two observations are worth emphasising. First, serving an open-weight, code-oriented model (qwen2.5-coder) locally through Ollama on a machine with 16 GB of unified memory was sufficient to generate structured Style JSON and data-driven expressions, supporting the claim that the approach is practical without dedicated GPU infrastructure or proprietary APIs. Second, decomposing the task into discrete MCP tools—inspect, generate, update, validate, retrieve—allowed the model to reason about one styling decision at a time and made it possible to reject specification-invalid output before it reached the client.

Several limitations remain. Open models of this size can misinterpret ambiguous prompts or generate expressions that are syntactically valid but cartographically sub-optimal, which is the motivation for the validation step and any subsequent correction cycles. Capabilities beyond simple recolouring—most notably classifying features by a categorical attribute, which the system is designed to support through match expressions—are not yet reliably implemented across the tool layer and remain the principal item of ongoing work. Highly complex requests, or attributes whose semantics are not obvious from their names, are more likely to require refinement through further dialogue. Finally, the present evaluation is qualitative and small in scale; a larger and more systematic study—covering more datasets, more rendering types, and a comparison between open models and proprietary backends—would be needed to characterise accuracy and performance rigorously. These directions are discussed further in the conclusions.

6. Conclusions

This paper presented a system that combines the Model Context Protocol with open Large Language Models served locally through Ollama to automatically generate Map Style JSON conformant with the MapLibre Style Specification from vector-based spatial data. The system exposes data inspection and style generation as a set of composable MCP tools and is designed to encode cartographic design principles—semantic colour selection, the ColorBrewer scheme families, and the perceptually grounded use of visual variables—so that the styles it returns aim to be not only specification-valid but also legible, directly from a user's data and a natural-language request. A preliminary evaluation on consumer hardware, using a 77-polygon GeoJSON dataset of Thai provincial boundaries and a vector tile service with more than 300 features, showed that the system could recolour a map's fill in response to a plain-language request, returning a style that MapLibre rendered without manual correction. It also showed that the open-weight model functioned mainly as a natural-language front-end, while correctness and renderer compatibility depended on the MCP tool layer.

The work makes progress toward the goals set out at the outset: it demonstrates a viable path to automating the generation of Map Style JSON, aims to lower the barrier to high-quality map visualization for users without GIS expertise, and—because it relies on open models and an architecture that runs locally through Ollama—aligns with the principles of the open geospatial ecosystem and can be integrated with other open-source tools.

Future work will extend the system in several directions: completing reliable support for categorical classification, such as grouping features by an administrative attribute; a larger and more systematic evaluation comparing open and proprietary model backends on accuracy and speed; support for additional data types such as raster and time-series data; richer handling of multi-attribute and multi-layer styling; and investigation of model fine-tuning or prompt strategies to improve the reliability of expression generation for complex requests.

References

Anthropic, 2024. Introducing the Model Context Protocol. www.anthropic.com/news/model-context-protocol (15 June 2026).

Bertin, J., 1983. *Semiology of Graphics: Diagrams, Networks, Maps*. University of Wisconsin Press, Madison, WI.

Harrower, M., Brewer, C.A., 2003. ColorBrewer.org: An online tool for selecting colour schemes for maps. *The Cartographic Journal*, 40(1), 27–37. doi.org/10.1179/000870403235002042.

Kang, Y., Gao, S., Roth, R.E., 2024. Artificial intelligence studies in cartography: a review and synthesis of methods, applications, and ethics. *Cartography and Geographic Information Science*, 51(4), 599–630. doi.org/10.1080/15230406.2023.2295943.

MacEachren, A.M., 1995. *How Maps Work: Representation, Visualization, and Design*. Guilford Press, New York.

MapLibre Contributors, 2024. MapLibre GL JS. maplibre.org/maplibre-gl-js/docs/ (15 June 2026).

Zhang, Y., He, Z., Li, J., Lin, J., Guan, Q., Yu, W., 2024. MapGPT: an autonomous framework for mapping by integrating large language model and cartographic tools. *Cartography and Geographic Information Science*, 51(6). doi.org/10.1080/15230406.2024.2404868.