CATALYST: CATalogue-driven LLM sYstem for Spatial Tiling*

Tarlan Bahadori
University of California, Riverside
Riverside, California, USA
tbaha001@ucr.edu

Ahmed Eldawy
University of California, Riverside
Riverside, California, USA
eldawy@ucr.edu

ABSTRACT

Geospatial data is a large and growing fraction of public open data, yet it remains difficult to discover, combine, visualize, and style at scale. We demonstrate CATALYST, a catalogue-driven and LLM-assisted system for interactive exploration of large geospatial repositories. Given a natural-language prompt, CATALYST retrieves relevant datasets through semantic search, derives needed information through joins or overlays when necessary, summarizes datasets with bounded enriched-schema aggregates, visualizes the resulting layers using a tile-based index, and generates cartographically meaningful styles through declarative styling or optional sandboxed code generation. The demo lets users issue natural-language visualization requests, inspect retrieved datasets and aggregate summaries, refine map styles conversationally, and generate executable styling code. Measurements on datasets up to 42.8 million features and 1.16 billion vertices show linear preprocessing, while the LLM-facing payload remains bounded by schema width rather than dataset size.

ACM Reference Format:
Tarlan Bahadori and Ahmed Eldawy. 2026. CATALYST: CATalogue-driven LLM sYstem for Spatial Tiling. In *Proceedings of (SIGSPATIAL '26).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Open data initiatives and advances in data collection technologies have led to an unprecedented growth in publicly available data, with geospatial data comprising a significant portion of it. For instance, Data.gov hosts nearly 300,000 geospatial datasets, accounting for more than 73% of all datasets on the platform. These datasets support applications in environmental monitoring, transportation, precision agriculture, urban planning, public health, and scientific decision-making.

Despite the wide availability and usefulness of geospatial datasets, three challenges limit their effective use. First, users often struggle to identify datasets relevant to a specific application because metadata can be incomplete, inconsistent, or missing. Second, exploratory analysis usually requires interactive map visualization, but geospatial datasets can reach tens or hundreds of gigabytes, making naive rendering impractical. Third, many analysis tasks require more than selecting and styling a single dataset: users may need to combine multiple datasets through overlays, attribute joins, or spatial joins to derive the information needed for a visualization. This is difficult because users must understand dataset schemas, compatible attributes, spatial relationships, and appropriate visual encodings, especially when datasets contain many unfamiliar attributes or lack descriptive documentation.

*This work is supported in part by the National Science Foundation (NSF) under grant IIS-2046236.

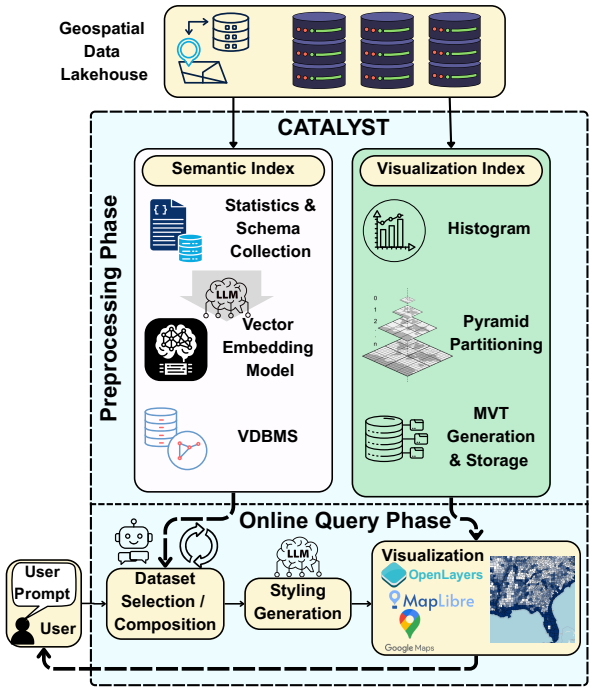

Figure 1: Overview of the CATALYST pipeline: enriched-schema aggregates and embeddings support semantic search, while tile histograms and visualization indexes enable scalable vector-tile rendering..

This paper demonstrates CATALYST, a catalogue-driven and LLM-assisted system designed to help researchers to explore geospatial datasets[1]. Rather than requiring users to select a dataset manually, CATALYST supports intent-driven prompts such as *"show buildings in high earthquake risk zones"* or *"visualize theft-related crime hotspots."* Given such prompts, CATALYST performs semantic retrieval over an enriched catalogue to identify relevant datasets and attributes. It then uses a tile-based visualization index to load and render the selected dataset on an interactive map, supporting low-latency visualization at large scale. Finally, it uses an LLM to generate map styling specifications, completing an end-to-end workflow from natural-language intent to interactive visualization. The source code and live demo are available at https://github.com/tarlaun/catalyst and https://starmap.cs.ucr.edu/catalyst.

Figure 1 gives a high-level overview of CATALYST, which runs in two phases: an *offline preprocessing* phase and an *online query* phase. During preprocessing, CATALYST connects to a geospatial data lakehouse and performs two complementary operations.

[1]A demonstration video is available at https://bit.ly/4f1Juv0

First, it computes enriched-schema aggregates that summarize each dataset's attributes, geometry, and value distributions. These summaries are converted into textual descriptors and embedded into a vector DBMS to support semantic retrieval. Second, it constructs a tile-based visualization index by building tile histograms and a multi-resolution pyramid, allowing the system to precompute dense tiles while generating sparse tiles on demand. Unlike raster-only methods [7], CATALYST generates vector tiles, enabling client-side styling during exploration.

During the online query phase, a user enters a free-text prompt. CATALYST embeds the prompt, retrieves candidate datasets, and sends only compact enriched-schema summaries to the LLM. The LLM then selects suitable attributes and generates either a structured style specification or executable visualization code. The conversation-driven interface allows users to refine dataset selection, styling choices, and generated code without manually inspecting the entire catalogue.

2 SYSTEM DESIGN

As illustrated in the system pipeline (Figure 1), our framework consists of an offline preprocessing phase and an online query phase. During preprocessing, we construct two complementary indexes in parallel: (1) an enriched-schema semantic index and (2) a tile histogram index, alongside multi-resolution MVT generation. The online phase leverages these indexes to support efficient dataset discovery, styling, and visualization.

2.1 Preprocessing Phase: Semantic Index

The first component in the offline step is building a semantic index over datasets, designed to support scalable discovery and attribute selection. For each dataset, we compute an *enriched schema* that captures attribute names, inferred types, and compact statistical summaries (e.g., distributions, top-k values, and numeric ranges) [3]. We then build a semantic index on this enriched schema by first generating a textual descriptor that combines dataset metadata with these summaries. This descriptor is embedded using a vector embedding model and stored in a PostgreSQL vector database (VDBMS). This index acts as a semantic index over the dataset repository, enabling similarity-based retrieval.

At query time, a user request is embedded and matched against this index to retrieve the top-k most relevant datasets. These candidates are passed to the LLM, which selects the most appropriate dataset and attribute(s) for visualization. This *search-then-reason* design avoids sending the entire catalogue to the LLM, reducing both latency and token usage.

2.2 Preprocessing Phase: Visualization Index

To support scalable interactive visualization, we adopt the standard pyramid tile structure used in modern web-based maps, where the spatial domain is recursively partitioned into tiles across multiple zoom levels (typically 20 levels). While fully materializing this pyramid enables low-latency access, it incurs prohibitive storage overhead due to the *exponential growth* in the number of tiles.

For scalable visualization, we construct a *tile histogram index* over the pyramid to capture data distribution across tiles. For each tile (z, x, y), we estimate (i) the number of intersecting features and (ii) the expected tile size in bytes. This is computed by mapping features to tiles and aggregating statistics such as vertex counts, resulting in a sparse summary of spatial density across zoom levels.

Using the tile histogram index, we classify tiles into *dense* and *sparse* categories. Dense tiles are those likely to exceed size thresholds, while sparse tiles are small and inexpensive to generate on demand. This classification enables a hybrid materialization strategy inspired by prior work on pre-rendered tiles [7], which we extend to support vector tiles (MVT).

In parallel with index construction, we perform pyramid partitioning and tile generation. For each dense tile, we extract the corresponding subset of features and apply bounded-size reservoir sampling to ensure that tile sizes remain within practical limits. Dense tiles are materialized offline as MVT files and stored in an on-disk index for efficient retrieval.

To support on-demand generation of sparse tiles, we additionally build a spatial index over all features. At query time, this index allows the system to efficiently retrieve features intersecting a requested tile and generate the corresponding MVT dynamically. Together, these components enable low-latency visualization while balancing storage overhead and computation cost.

2.3 Online Query Phase

At query time, CATALYST processes a user prompt and visualizes the result through the following steps:

(1) It embeds the user prompt and retrieves candidate datasets from the semantic index.
(2) It sends the prompt and compact enriched-schema summaries to the LLM for final dataset and attribute selection.
(3) It loads the selected dataset through the visualization index and displays the corresponding vector tiles in the browser.
(4) It uses the LLM to generate either a structured style specification or executable visualization code.

The user can refine the result through follow-up prompts. The session manager preserves the current dataset, selected attributes, styling mode, and previous instructions, allowing the LLM to interpret revisions such as *"use a red palette," "show only the top five classes,"* or *"apply a log scale"* without requiring the user to restate the full request.

2.3.1 Safe and Robust LLM Styling. To balance efficiency, security, and flexibility in map styling, CATALYST supports two styling methods, *declarative* and *executable*. Declarative styling follows a predefined set of styles that use selected attributes, renderer mode, palette, opacity, and related parameters. For more flexible styling, CATALYST also supports executable styling, expressed as JavaScript code that defines per-feature behavior. For safety, executable styling is optional and requires user approval after code review.

For additional security, the generated code executes inside an isolated map runtime. The runtime is served in a sandboxed `iframe` from an origin distinct from the parent application, preventing access to the parent DOM, cookies, local storage, or session state. Communication is limited to a small `postMessage` protocol for setting datasets, applying styles, and running approved code.

The LLM interface also includes robustness checks for malformed outputs. Structured responses are parsed defensively, falling back from strict JSON to a tolerant extractor for markdown-fenced or

YAML-like outputs. Categorical styles are normalized by removing invalid or duplicate colors and filling missing colors from a fixed qualitative palette. These safeguards reduce common LLM-output failures while preserving interactive map rendering.

2.3.2 Multi-Dataset Query and Tiled Joins. Many user requests require reasoning over more than one dataset. For example, a user may ask to *"show buildings by fire hazard severity"* or *"color parks based on dominant vegetation."* Such prompts require CATALYST to identify multiple relevant datasets, infer their roles, and decide whether to render them as coordinated overlays or derive a new layer through a join or aggregation.

Given a prompt, CATALYST retrieves candidate datasets from the semantic index and passes their enriched-schema summaries to the LLM. The LLM produces a structured plan that assigns each dataset a role, such as target layer, context layer, or aggregation layer, and selects the appropriate operation. The plan may specify a multi-layer overlay, an attribute join over a shared key, or a spatial join using predicates such as `intersects`, `within`, or `nearest`.

For derived layers, CATALYST uses a tile-aligned join strategy. Instead of loading the entire target dataset, running one global join, and rebuilding the full tile pyramid, the system reuses the target's existing spatial partitions. Each partition is joined only with source features overlapping its tile bounding box, the derived attribute is appended, and the augmented partition is written back with bounding-box columns for predicate pushdown. Since target partitions are disjoint and source features relevant to a target row must overlap that partition's bounding box, the per-partition join preserves the same result while avoiding global re-partitioning. This design makes multi-dataset queries practical in the interactive demo. Overlay queries are rendered as synchronized vector-tile layers, while join and aggregation queries produce derived layers that can be styled like ordinary datasets. Generated tiles for derived layers are served on demand and can be cached lazily, allowing users to express cross-dataset spatial questions without manually locating datasets, writing spatial SQL, or rebuilding an entire visualization pyramid up front.

3 EVALUATION

We evaluate the preprocessing scalability of CATALYST on subsets of OSM21-Parks [6] at increasing scales. The measurements are end-to-end preprocessing runs: the system builds the visualization index, generates tiles, and computes enriched-schema aggregates inline during tiling. The experiments were run on an Apple M3 Pro machine with 11 cores and 18 GB RAM.

Table 1 shows that preprocessing remains close to linear as the input grows from 2.5 million to 42.8 million features and from 118.7 million to 1.16 billion vertices. Throughput remains near 18–20K rows/s across the full range, and peak memory remains around 3 GB through the 10M-feature scale before increasing to 7.51 GB for the largest input.

For cross-dataset joins, we compared the count aggregation using the full derive path and the tile-aligned derive path on the LA buildings layer with 3.88M polygons. The tile-aligned version completed in 58.7 s, while the full path took 2,417.6 s because it re-partitions the derived dataset and rebuilds the full MVT pyramid. Both methods preserve the same 3,879,476 output rows, but the tiled approach makes the derived result queryable immediately through on-demand tiles.

Table 1: End-to-end preprocessing scalability on OSM-Parks. Input size reports the source GeoParquet size. The aggregate pass is computed inline during tiling.

File Size	Features	Vertices	Time (s)	Rows/s	Peak Mem. (GB)
1.61 GB	2,490,471	118,706,621	136.95	18,185	3.04
2.90 GB	4,980,942	213,810,099	252.51	19,726	3.10
3.87 GB	7,471,413	288,741,888	375.64	19,890	3.14
5.56 GB	9,961,884	408,161,460	513.06	19,417	3.15
12.80 GB	19,923,768	816,322,920	1,151.96	17,296	4.96
17.60 GB	42,769,620	1,155,340,647	2,307.07	18,539	7.51

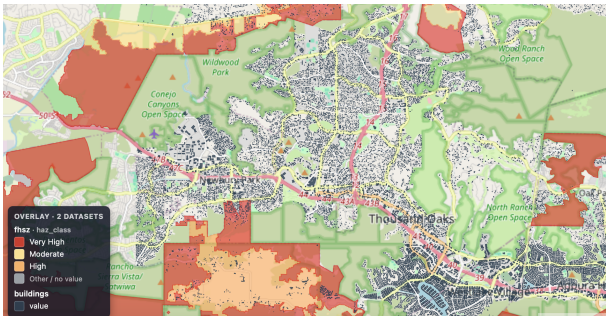

Figure 2: Visualization for "Show buildings fire hazard severity." CATALYST overlays buildings with Fire Hazard Severity Zones and colors them by hazard class using a red scale.

4 DEMONSTRATION SCENARIOS

We demonstrate CATALYST through interactive scenarios that highlight dataset discovery, aggregate-aware styling, conversational refinement, and executable code generation. We will preload the system with various datasets in Riverside for demonstration. The current implementation can use either Google Gemini as a hosted LLM backend or Ollama for local model execution, although the architecture is model-agnostic and can support alternative LLMs.

4.1 Scenario 1: Single-Dataset Exploration

In the first scenario, the user enters a prompt that can be answered using a single dataset, such as "Color Riverside based on vegetation type." CATALYST embeds the prompt and retrieves candidate datasets from the semantic index. The assistant then selects the most relevant dataset, identifies the attribute that best matches the user's intent, and generates an appropriate map style. For this prompt, CATALYST selects the Riverside vegetation dataset and uses the vegetation-category attribute for styling. The system generates a categorical color map, where each major vegetation type is assigned a distinct color. Users can also inspect the selected dataset's enriched metadata, including attribute names, value summaries, and top-k categories, to understand why the attribute was chosen and how the style was produced. After the initial map is generated, the user can refine it conversationally. For example, the user may ask to change the color palette, show only the most common vegetation classes, or add labels for selected categories.

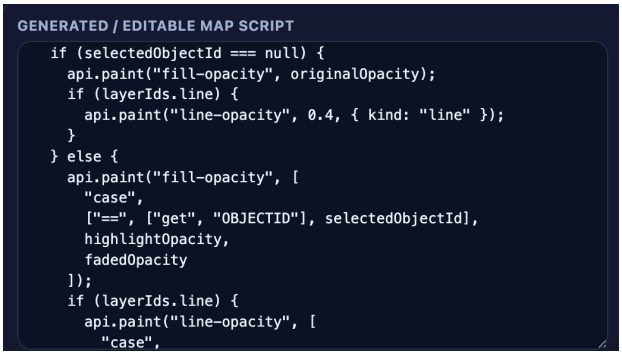

Figure 3: Generated styling code for inspection and editing.

This scenario introduces the basic CATALYST workflow: natural-language request, semantic dataset retrieval, attribute selection, enriched-schema inspection, and interactive map styling.

4.2 Scenario 2: Multi-Dataset Exploration

In the multi-dataset scenario, the user enters a prompt such as "Show buildings fire hazard severity." Unlike a single-layer styling request, this prompt requires information from two datasets: a buildings layer and a fire-hazard-zone layer. CATALYST first retrieves candidate datasets from the semantic index, then asks the AI assistant to produce a structured multi-dataset plan.

The backend validates the plan and renders buildings with Fire Hazard Severity Zones as coordinated overlay layers. CATALYST then computes the enriched-schema summary for the derived layer and uses it to generate a structured style. The resulting map highlights buildings inside hazard zones while preserving the surrounding buildings as spatial context. The generated visualization is shown in Figure 2. Additionally, users can see explanation from the assistant, for example: *"I have overlaid the building footprints on top of the Fire Hazard Severity Zones (FHSZ). The hazard zones are color-coded from moderate to very high risk, allowing you to easily identify which buildings and communities are located in high-exposure areas."*

This scenario shows how CATALYST turns natural-language cross-dataset questions into interactive maps by retrieving relevant layers, inferring their roles, and executing the needed join or overlay to generate multi-dataset visualizations.

4.3 Scenario 3: Visualization Code Generation

In the third scenario, the user asks CATALYST to generate executable visualization code for a request that goes beyond a single declarative style. For example, the user may enter: "Highlight Riverside's vegetation patterns by coloring each vegetation type differently and using opacity to make large areas stand out while fading small patches." This prompt requires a bivariate style: vegetation type controls the fill color, while patch area controls opacity. A fixed one-attribute style template is insufficient because the visualization depends on multiple attributes, conditional logic, and a logarithmic area transformation to prevent small patches from visually dominating the map.

Another prompt the user may enter is: "Allow me to click on a vegetation area, fade out all other polygons, and display a popup with its vegetation type and area." This requires click-event handling, feature selection, conditional opacity updates, and popup generation, making executable code more appropriate than a fixed declarative style. Figure 3 shows generated code for this prompt.

CATALYST retrieves the Riverside vegetation dataset and uses the enriched metadata to identify the relevant vegetation-class and area attributes. Instead of returning only a structured style object, the code-generation module produces a JavaScript snippet with dataset access, MapLibre styling expressions, and custom rendering logic. The generated code can assign colors by vegetation category, compute opacity from polygon area, group rare classes into an "Other" category, and add labels or hover popups showing the original vegetation type.

This scenario demonstrates why executable styling is useful. Code generation supports richer logic such as multi-attribute encodings, thresholds, zoom-dependent rules, and interaction handlers. The generated code gives users an editable starting point for customized map design while keeping the workflow grounded in the selected dataset schema and aggregate summaries.

5 FUTURE WORK

Future work will improve both the scale and visual quality of CATALYST. Dense vector tiles are currently bounded using sampling, which keeps tile sizes practical but may remove visually important geometries and reduce map quality. We plan to integrate visualization-aware tile reduction methods such as HiFIVE [4], which aim to reduce tile size while preserving visual fidelity and semantic information. Additionally, CATALYST's streaming, mergeable summaries, including HyperLogLog sketches, top-k summaries, numeric moments, and tile histograms, naturally support parallel and distributed preprocessing. Independent workers can process spatial partitions or file shards and merge their outputs into global catalogue summaries and visualization indexes, extending the system toward terabyte-scale geospatial repositories.

REFERENCES

[1] 2025. Interactive Maps with OpenLayers. https://openlayers.org/. OpenLayers Documentation.
[2] 2026. Starlet: MVT generation, and serving. https://pypi.org/project/starlet/.
[3] Tarlan Bahadori et al. 2025. LASEK: LLM-Assisted Style Exploration Kit for Geospatial Data. *Proc. VLDB Endow.* 18, 12 (2025).
[4] Tarlan Bahadori and Ahmed Eldawy. 2026. HiFIVE: High-Fidelity Vector-Tile Reduction for Interactive Map Exploration. arXiv:2603.10270 [cs.DB]
[5] Ahmed Eldawy et al. 2021. Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In *ACM International Conference on Information and Knowledge Management (CIKM '21)*. 3796–3807.
[6] Saheli Ghosh et al. 2019. UCR-STAR: the UCR spatio-temporal active repository. *SIGSPATIAL Special* 11, 2 (2019).
[7] Saheli Ghosh and Ahmed Eldawy. 2022. AID*: A Spatial Index for Visual Exploration of Geo-Spatial Data. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2022).
[8] Kevin Hu et al. 2019. VizML: A Machine Learning Approach to Visualization Recommendation. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*.
[9] Jinfeng Liu et al. 2023. Large language models for information visualization: Survey and opportunities. *Visual Informatics* 7, 4 (2023), 113–124.
[10] MapLibre Community. 2026. MapLibre GL JS (and Native). https://maplibre.org/.
[11] Kanit Wongsuphasawat et al. 2016. Towards a general-purpose query language for visualization recommendation. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA'16@SIGMOD)*.
[12] Yupeng Xie et al. 2024. HAIChart: Human and AI Paired Visualization System. *Proc. VLDB Endow.* 17, 11 (July 2024), 3178–3191.
[13] Jia Yu and Mohamed Sarwat. 2021. GeoSparkViz: a cluster computing system for visualizing massive-scale geospatial data. *The VLDB Journal* 30, 2 (Jan. 2021), 22.