UAB
Universitat Autònoma de Barcelona

Dipòsit digital de documents de la UAB

This is the **published version** of the bachelor thesis:

El Messaouri Belmehdi, Mouad. *UCRStar Lite*. Treball de Final de Grau (Universitat Autònoma de Barcelona), 2026 (Grau en Enginyeria de Dades)

This version is available at https://ddd.uab.cat/record/337318

under the terms of the BY-NC-ND license.

UCR StarLite: Exploración Web Interactiva de Datos Geoespaciales

Mouad El Messaouri Belmehdi

Resumen—UCR StarLite es una aplicación web para explorar de forma interactiva conjuntos de datos geoespaciales del repositorio UCR-STAR. La plataforma combina visualización mediante teselas vectoriales, estilado dinámico por atributos, geocodificación, exportación de subconjuntos y un asistente local basado en Ollama para sugerir estilos de mapa. Su arquitectura cliente-servidor, implementada con MapLibre GL JS, Flask y formatos eficientes como MVT y Parquet, permite trabajar con grandes volúmenes de datos sin cargar el dataset completo en el navegador. Como resultado, el sistema ofrece una interfaz ligera, flexible y accesible para usuarios técnicos y no técnicos.

Palabras clave—datos geoespaciales; UCR-STAR; MapLibre GL JS; teselas vectoriales; estilado cartográfico; geocodificación; exportación de datos; asistente LLM local; Ollama; visualización web.

Resum—UCR StarLite és una aplicació web per explorar de manera interactiva conjunts de dades geoespacials del repositori UCR-STAR. La plataforma combina visualització amb tessel·les vectorials, estilat dinàmic per atributs, geocodificació, exportació de subconjunts i un assistent local basat en Ollama per suggerir estils de mapa. La seva arquitectura client-servidor, implementada amb MapLibre GL JS, Flask i formats eficients com MVT i Parquet, permet treballar amb grans volums de dades sense carregar el conjunt complet al navegador. Com a resultat, el sistema ofereix una interfície lleugera, flexible i accessible per a usuaris tècnics i no tècnics.

Paraules clau—dades geoespacials; UCR-STAR; MapLibre GL JS; tessel·les vectorials; estilat cartogràfic; geocodificació; exportació de dades; assistent LLM local; Ollama; visualització web.

Abstract—UCR StarLite is a web application for the interactive exploration of geospatial datasets from the UCR-STAR repository. The platform combines vector-tile visualization, attribute-based styling, geocoding, subset export, and a local Ollama-based assistant that suggests map styles. Its client-server architecture, built with MapLibre GL JS, Flask, and efficient formats such as MVT and Parquet, enables large datasets to be explored without loading them entirely in the browser. The result is a lightweight, flexible, and accessible interface for both technical and non-technical users.

Index Terms—geospatial data; UCR-STAR; MapLibre GL JS; vector tiles; cartographic styling; geocoding; data export; local LLM assistant; Ollama; web visualization.

1 Introducción y motivación

DURANTE la última década, el volumen de datos espaciales y espacio-temporales disponibles públicamente ha crecido de forma espectacular, impulsado por satélites, redes de sensores, dispositivos móviles e iniciativas de datos abiertos. La mayoría de estos conjuntos de datos se publican a través de portales estáticos que presentan largas listas textuales de recursos: el usuario debe adivinar el contenido de un dataset a partir de una breve descripción, o bien descargarlo, preprocesarlo y cargarlo en sus propias herramientas antes de poder evaluar su cobertura espacial, calidad y utilidad para una pregunta de investigación concreta.

El UCR Spatio-temporal Active Repository (UCR-STAR) [1] fue creado para abordar esta limitación, ofreciendo un catálogo interactivo basado en mapas de grandes conjuntos de datos geoespaciales. En lugar de una tabla estática de entradas, UCR-STAR permite a los usuarios seleccionar un dataset y visualizar instantáneamente su distribución espacial en un mapa web, respaldado por índices de big data que permiten visualizar datasets de cientos de gigabytes en fracciones de segundo, y ya aloja más de un centenar de datasets con un tamaño total superior a un terabyte. A pesar de este progreso, persiste una brecha entre la potente infraestructura de backend de UCR-STAR y las necesidades de los usuarios finales: investigadores y profesionales que desean herramientas ligeras para explorar, dar estilo y descargar subconjuntos de datos para su propio análisis. La interfaz original se centra en visualizar rápidamente un dataset completo con un estilo predefinido, pero ofrece soporte limitado para la experimentación interactiva con estilos de visualización y selección de atributos.

A esta brecha se añade una oportunidad identificada en

investigaciones recientes: los modelos de lenguaje de gran escala (LLMs) pueden asistir a los usuarios recomendando qué atributos visualizar y cómo darles estilo, reduciendo el esfuerzo manual necesario para configurar visualizaciones cartográficas eficaces para conjuntos de datos complejos [2]. El proyecto UCR StarLite propone cerrar ambas brechas desarrollando una herramienta de visualización ligera basada en web, construida en base a UCR-STAR, que no requiere ningún software adicional más allá de un navegador y un servidor local.

Este documento describe el diseño, la implementación y la evaluación del prototipo resultante. El documento se organiza de la siguiente manera: La sección 2 enumera los objetivos específicos del TFG. La sección 3 describe la arquitectura general del sistema y la separación entre sus tres bloques funcionales. La sección 4 detalla el bloque de funcionalidad: el sistema de teselas vectoriales MVT, la gestión de múltiples capas activas con el diccionario activeLayers, la geocodificación y la exportación espacial. La sección 5 describe el motor de estilado dinámico, con sus tres modos de visualización, los esquemas de color y el sistema de leyendas centralizado. La sección 6 explica el asistente de estilo basado en Ollama: la construcción del contexto del mapa, el protocolo de comunicación con el modelo y la aplicación del estilo sugerido. La sección 7 recoge las limitaciones actuales y las líneas de trabajo futuro. La sección 8 concluye el documento.

2 Objetivos

El objetivo general de este TFG es diseñar e implementar UCR StarLite, una aplicación de navegador que proporcione un punto de entrada accesible e interactivo al repositorio UCR-STAR tanto para usuarios técnicos como no técnicos. Este objetivo general se descompone en los siguientes objetivos específicos, organizados por área funcional:

- Visualización interactiva en mapa: conectar el frontend a los datasets de UCR-STAR y renderizarlos con MapLibre GL JS [4], garantizando una interacción fluida incluso para datasets de gran tamaño gracias a la infraestructura de teselas vectoriales.
- Estilos de visualización configurables: permitir elegir el atributo de etiquetado y aplicar esquemas de color categóricos y graduados, con leyendas que faciliten la interpretación del mapa.
- Interactividad avanzada: mostrar el conjunto completo de atributos de un registro al hacer clic, habilitar animaciones de tipo "volar-a" y codificar el estado de la vista en la URL para que pueda compartirse y restaurarse.
- Soporte de búsqueda y geocodificación: permitir localizar nombres de lugares mediante una API de geocodificación pública y desplazar el mapa hacia el resultado elegido.
- Descarga de subconjuntos: exponer una interfaz sencilla para descargar el dataset completo o solo el subconjunto correspondiente a la vista actual, en múltiples formatos.
- Implementar un asistente que sugiera atributos y estilos, ejecutado íntegramente en local mediante Ollama.
- Documentación y evaluación: documentar la arquitectura y las decisiones de implementación, y evaluar el prototipo mediante escenarios de uso representativos.

3 Arquitectura del sistema

El sistema mantiene la separación cliente-servidor clásica. Sobre el navegador, la arquitectura se organiza en tres bloques funcionales: funcionalidad, estilado y asistente de IA. Estos módulos se apoyan en un servidor Flask y en dos servicios externos, Ollama y Nominatim. La Figura 1 resume esta arquitectura, mientras que las secciones 4 a 6 detallan cada bloque.

El frontend está compuesto por tres artefactos principales:

- view_mvt.html, que define la estructura de la interfaz.
- view_mvt.css, que gestiona el diseño mediante variables para modo claro y oscuro.
- view_mvt.js, que contiene toda la lógica del cliente organizada en módulos.

El backend se implementa en server.py con Flask y expone las siguientes funcionalidades:

- Endpoints de teselas MVT.
- API de datasets y estadísticas.
- Servicio de descarga multiformato mediante download_service.py.
- Soporte para filtro MBR.

El estado de la aplicación se distribuye entre cuatros mecanismos complementarios:

- Variables JavaScript globales, que mantienen la sesión activa.
- El hash de la URL, que conserva la posición y el zoom.
- El parámetro ?datasets=, que indica los datasets cargados.
- localStorage, que almacena la preferencia de tema.

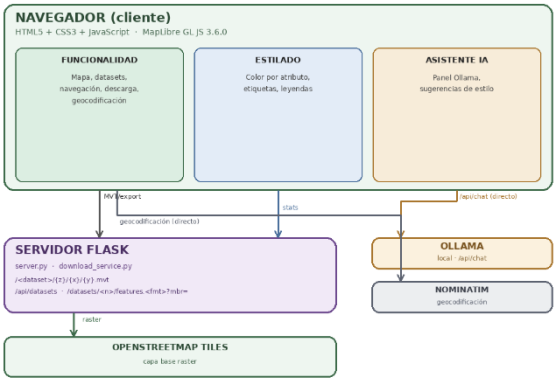


Fig 1. Arquitectura de UCR StarLite: el navegador concentra la funcionalidad, el estilado y el asistente de IA; el servidor Flask expone los datasets y gestiona las descargas; Ollama y Nominatim operan al margen del servidor, directamente desde el cliente.

4 FUNCIONALIDAD EN DETALLE

4.1 Visualización de teselas vectoriales (MVT)

UCR StarLite no envía los datasets completos al navegador: los divide en teselas vectoriales (Vector Tiles, MVT), pequeños fragmentos binarios indexados por nivel de zoom y posición en una cuadrícula (z, x, y). Cuando el usuario navega, MapLibre GL JS calcula qué teselas son visibles y las solicita al servidor mediante una URL del tipo /<dataset>/{z}/{x}/{y}.mvt. El servidor resuelve cada petición en tres niveles: primero comprueba una caché en memoria (TileCache, con política LRU y un límite de 256 entradas por dataset); si no hay acierto, busca la tesela ya generada en disco bajo mvt/{z}/{x}/{y}.mvt, evitando recalcularla si un usuario anterior ya la solicitó; solo si no existe en ninguno de los dos niveles, VectorTiler la genera de cero (generate()): consulta el índice espacial ParquetIndex sobre el mismo directorio parquet_tiles que utiliza el servicio de descarga (sección 4.5) para localizar únicamente los archivos que intersecan el bounding box de la tesela en coordenadas geográficas, reproyecta su contenido a EPSG:3857, recorta las geometrías exactamente al límite de la tesela (clip_to_tile()) y las codifica en formato MVT (MVTEncoder). La tesela resultante se escribe en disco y se añade a la caché en memoria antes de devolverse, de modo que la siguiente petición sobre esa misma celda, se resuelve sin volver a tocar los archivos parquet. Este diseño de generación perezosa con caché persistente es lo que permite que UCR StarLite ofrezca tiempos de respuesta bajos sin necesidad de pre-generar todas las teselas del dataset por adelantado, a costa de una primera petición más lenta sobre cada celda nueva del mosaico.

Esto permite visualizar datasets de cientos de gigabytes en fracciones de segundo, ya que el cliente solo descarga, decodifica y renderiza la porción de datos relevante para la vista actual, en lugar de cargar el dataset entero. Al cambiar el zoom, MapLibre solicita un nuevo conjunto de teselas con mayor o menor nivel de detalle, las geometrías se simplifican en niveles de zoom bajos para evitar sobrecargar el renderizado, las teselas ya descargadas se cachean en el navegador mientras el usuario permanece en la misma región, y se descartan progresivamente al alejarse de ella. Sobre cada tesela, UCR StarLite crea automáticamente las capas de renderizado según el tipo de geometría detectado: fill y outline para polígonos, circle para puntos y line para líneas, asignando a cada una un identificador de capa con el prefijo del dataset para poder gestionarlas individualmente cuando hay varias activas (sección 5.3).



Fig 2. Flujo de carga de un dataset: del clic en el panel a las teselas MVT renderizadas en el mapa.

4.2 Selección de datasets y navegación

l panel lateral obtiene la lista de datasets disponibles mediante una llamada asíncrona a /api/datasets y la renderiza dinámicamente mediante manipulación del DOM, incluyendo un cuadro de búsqueda que filtra la lista por nombre a medida que el usuario escribe. Un clic normal sobre un dataset ejecuta selectDataset(), que elimina las capas anteriores y carga la nueva en modo de sustitución; el botón "+" que aparece al pasar el cursor sobre un dataset no cargado ejecuta addDatasetLayer() en modo de superposición, acumulando la nueva capa sobre las existentes (sección 4.3).

El estado de la vista, dataset, centro y zoom, se codifica en la URL de la página mediante window.history.replaceState con un formato del tipo ?dataset=<nombre>#center=<lat>,<lng>&zoom=<z>, actualizado con un debounce de 150 ms en cada movimiento del mapa para evitar saturar el historial del navegador. Esto permite restaurar y compartir cualquier vista del mapa mediante un enlace directo, y constituye la base sobre la que se apoyan tanto la descarga de subconjuntos (sección 4.5) como la restauración de sesiones con múltiples capas.

4.3 Gestión de múltiples capes activas

Esta es una de las extensiones más relevantes incorporadas durante el proyecto, no contemplada en el plan de trabajo original. El sistema soporta hasta seis capas vectoriales activas simultáneamente sobre el mismo mapa base, con colores diferenciados asignados automáticamente a partir de una paleta predefinida de seis tonos. El estado de todas las capas activas se mantiene en el diccionario activeLayers, que almacena por cada dataset su fuente MapLibre, el prefijo de sus capas, el tipo de geometría detectado, el índice de color asignado y los estados de visibilidad y opacidad; este diccionario es la única fuente de verdad tanto para el renderizado del panel Active Layers como para la sincronización con la URL de la página.

El panel Active Layers lista todas las capas activas con su color identificador, un control de visibilidad con icono diferenciado según el estado, un slider de opacidad que actualiza la capa en tiempo real mediante setPaintProperty, y un botón de eliminación individual.

Al hacer clic sobre una fila, esa capa pasa a ser la capa activa para estilado y los selectores de atributos se actualizan para reflejar sus campos específicos. La URL se actualiza con el parámetro ?datasets= que codifica todos los datasets activos separados por comas, permitiendo restaurar sesiones completas de múltiples capas mediante enlace directo (Figuras 3 y 4).

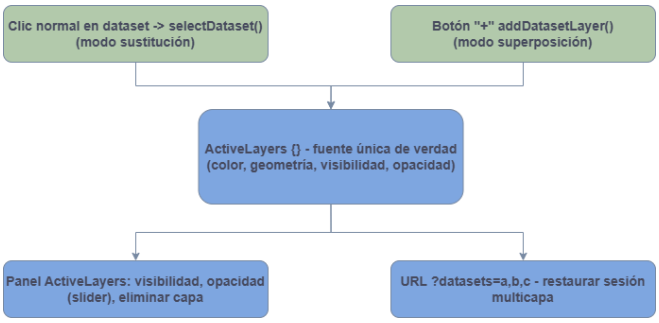


Fig 3. Flujo de gestión de múltiples capas: sustitución (clic) vs. superposición (botón "+") sobre el diccionario activeLayers.

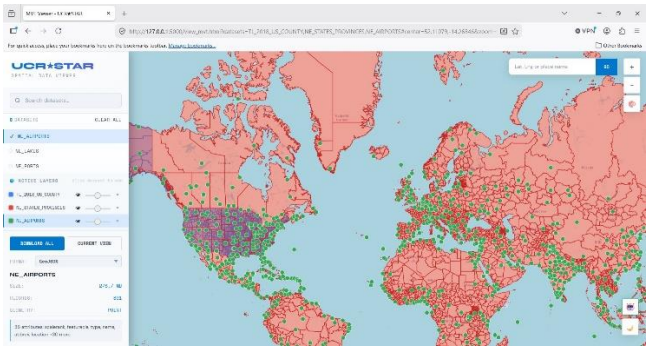


Fig 4. Panel Active Layers con tres datasets superpuestos (condados, fronteras de provincia y aeropuertos), cada uno con su color

4.4 Búsqueda geocodificada

El cuadro de búsqueda acepta coordenadas directas (lat, lng) o nombres de lugar. En el segundo caso, se consulta la API de Nominatim [5] de OpenStreetMap con Accept-Language. Se muestra una lista desplegable de hasta seis resultados mediante la función _showGeoResults() (Figuras 5 y 6). Al seleccionar uno, el mapa ejecuta map.flyTo() con una transición animada (speed: 1.4, curve: 1.42) hacia las coordenadas del resultado; si la búsqueda devuelve un único resultado, el desplazamiento se produce directamente sin mostrar la lista. La lista de resultados se cierra automáticamente al hacer clic fuera del área correspondiente, y se muestran mensajes de error o ausencia de resultados cuando procede.

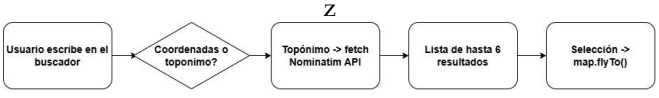


Fig 5. Flujo de búsqueda geocodificada: distinción entre coordenadas directas y topónimos, consulta a Nominatim y desplazamiento con flyTo().

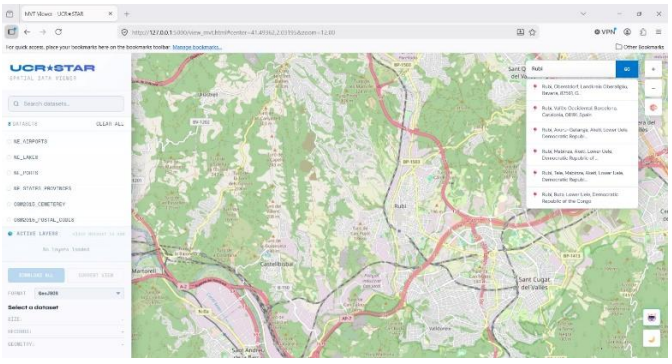


Fig 6. Búsqueda geocodificada de "Rubí" mediante la API de Nominatim, con lista de resultados desplegable sobre el mapa.

4.5 Descarga y exportación de datos

La función downloadDataset(mode) gestiona dos modos de operación. En el modo 'all' construye la URL /datasets/{dataset}/features.{fmt}. En el modo 'viewport' calcula el bounding box visible mediante map.getBounds() y genera el parámetro mbr con los valores oeste, sur, este y norte redondeados a seis decimales, construyendo la URL /datasets/{dataset}/features.{fmt}?mbr=W,S,E,N.

En el servidor, download_service.py (clases BoundingBox, TileManager y FeatureStreamer) filtra los parquet_tiles del dataset que intersecan ese MBR y transmite las features en streaming, línea a línea en CSV, objeto a objeto en GeoJSON, evitando cargar el dataset completo en memoria incluso para los datasets más grandes.

Se ha añadido un selector de formato de salida en el panel lateral con opciones GeoJSON, CSV y Shapefile comprimido en ZIP (Figuras 7 y 8); la función adapta la extensión del archivo y el nombre sugerido para la descarga en función del formato elegido. Los botones muestran un indicador de estado ("Preparing…") durante el proceso y se reactivan automáticamente tras 2,5 segundos.

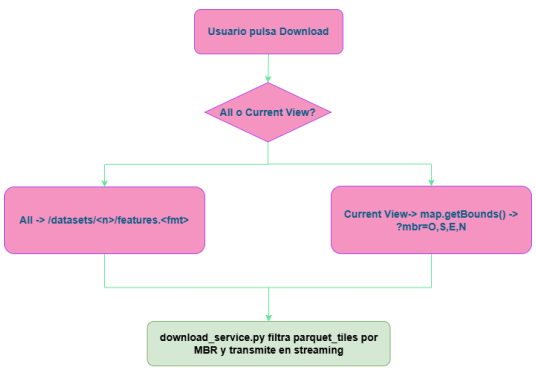


Fig 7. Flujo de descarga: All vs. Current View, cálculo del MBR y filtrado de parquet_tiles en streaming.

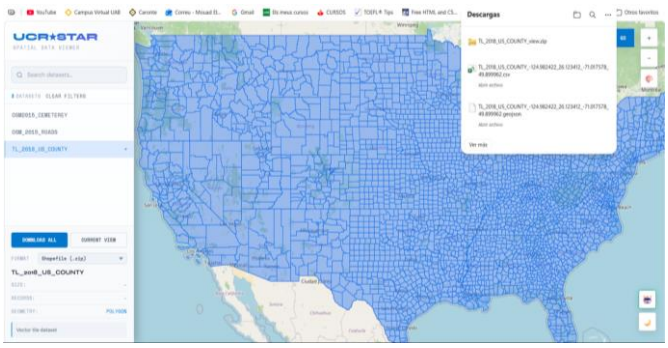


Fig 8. Descarga de un subconjunto del dataset TL_2018_US_COUNTY en diferentes formatos: el navegador recibe el archivo .zip Shapefile junto con sus equivalentes CSV y GeoJSON solicitados previamente.

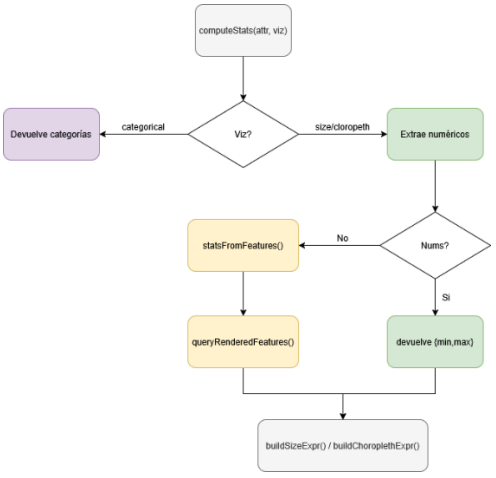


Fig 9. Proceso de obtención de estadísticas para construir las expresiones de estilo. ComputeStats()

5 ESTILADO EN DETALLE

5.1 Estilado por atributos

El endpoint /api/datasets/<n>/stats devuelve, para cada atributo, información como los valores más frecuentes, valores mínimos, máximos, media y el recuento de valores no nulos. A partir de esta información, el usuario elige un atributo y un esquema de color entre diecisiete disponibles, organizados en familias secuenciales (Blues, Reds, Greens…), divergentes (Red→Blue, Spectral) y perceptuales (Viridis, Magma, Rainbow). Para atributos numéricos se construye una expresión interpolate que mapea el rango de valores sobre una rampa de cinco colores (modo continuo); para atributos textuales se usa una expresión match que asigna un color por categoría, con un color gris de reserva (#d3d3d3) para valores no contemplados (modo categórico). Un tercer modo, size by value, varía el radio de los puntos en lugar del color, útil para datasets de tipo punto donde la magnitud importa más que el color.

El cálculo de las estadísticas necesarias para construir estas expresiones (computeStats()) usa dos fuentes de forma ordenada: primero el endpoint del servidor, y en su defecto un cálculo sintético sobre las features visibles mediante map.queryRenderedFeatures() (statsFromFeatures()) (Figura 9), lo que permite que el estilado funcione incluso sin conexión al servidor Flask. Todos los estilos se aplican en tiempo real mediante setPaintProperty, sin eliminar ni recrear las capas vectoriales, lo que evita parpadeos visuales al cambiar de esquema o de atributo. La Figura 11 muestra un caso de uso donde evaluamos el estilado y el resultado: el dataset TL_2018_US_COUNTY coloreado por el atributo ALAND con el esquema Viridis en modo continuo, junto con el popup de atributos de un condado y el panel Styling Options.

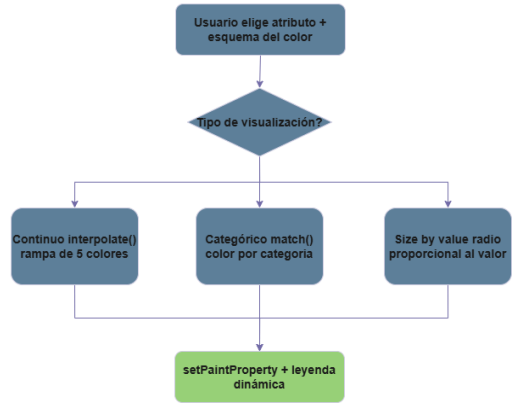


Fig 10. Flujo de estilado por atributos: continuo (interpolate), categórico (match) o size by value, hasta setPaintProperty() y la leyenda.

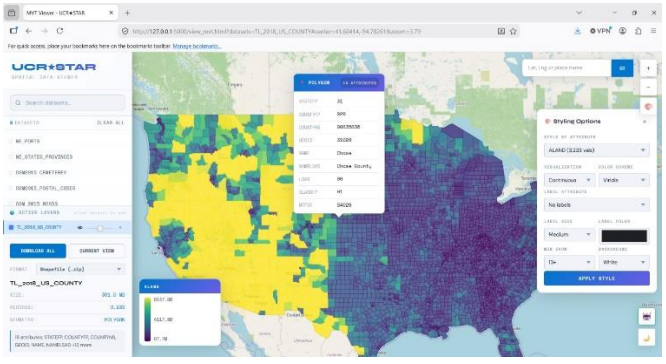


Fig 11. Estilado por atributos (ALAND, esquema Viridis), popup de atributos al hacer clic sobre un condado y panel Styling Options.

5.2 Etiquetas y leyendas dinámicas

Un menú desplegable permite elegir qué atributo se muestra como etiqueta textual sobre las geometrías, renderizada en un canvas superpuesto (#label-canvas) en lugar de una capa de símbolos de MapLibre. Esta decisión evita las limitaciones de la capa de símbolos nativa a la hora de combinar múltiples datasets superpuestos (sección 4.3) con fondos personalizados por capa, ya que cada dataset activo necesita su propio color de fondo y los símbolos de MapLibre comparten una única configuración de colisión global entre capas. Por cada fotograma visible, el sistema calcula primero un punto de anclaje por feature según su tipo de geometría, las coordenadas directas para puntos, el centroide del anillo exterior para polígonos (o del anillo más grande en el caso de multipolígonos) y el punto medio de la polilínea para líneas y proyecta ese punto a coordenadas de pantalla mediante map.project(). A partir de las posiciones proyectadas se aplica un algoritmo de detección de colisiones por cajas delimitadoras (bounding boxes) con un margen de 4 píxeles: cada candidata se compara contra las etiquetas ya colocadas en el fotograma actual, y solo se dibuja si su caja no se superpone con ninguna de ellas, descartándose en caso contrario. Esto evita el solapamiento visual entre etiquetas cuando hay muchas geometrías visibles a un mismo nivel de zoom, a costa de que algunas features queden sin etiqueta cuando la densidad es alta. Las etiquetas solo se dibujan a partir de un zoom mínimo configurable (5, 7, 9, 11 o 13+), y admiten distintos fondos (ninguno, blanco, oscuro o color a juego con la capa), dibujados como rectángulos redondeados sobre el canvas; cuando no hay fondo, el texto se traza con un contorno blanco para mantener la legibilidad sobre cualquier color de mapa base. (Figura 12)

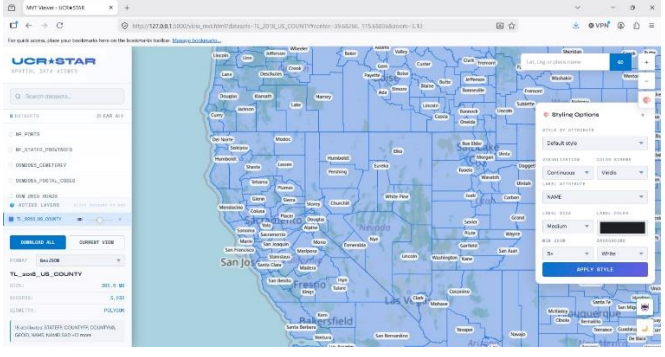


Fig 12. Label del atributo NAME de tamaño medio sobre el mapa con zoom mínimo de 5+.

La leyenda se gestiona de forma centralizada en la función _dispatchStyle(), que invoca hideLegend() siempre al inicio y actualiza la leyenda una única vez al final, independientemente de qué tipos de geometría estén presentes en el dataset, este diseño elimina una clase de errores de persistencia en la que la leyenda de un modo permanecía visible al cambiar a otro. Se distinguen tres tipos de leyenda: una barra de gradiente con valores mínimo, medio y máximo para el modo continuo; chips de color por categoría para el modo categórico; y tres círculos proporcionales (máximo, medio y mínimo) en el color original de la capa para el modo size by value, donde una barra de color sería engañosa porque el color no varía. (Figura 13)

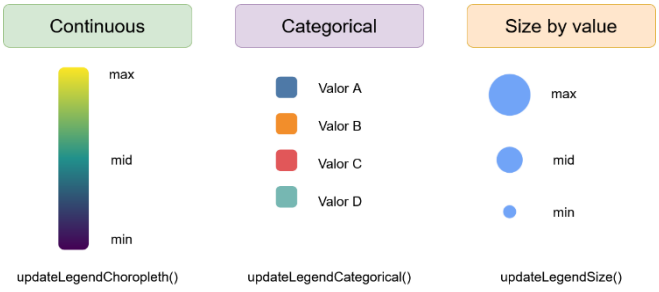


Fig 13. Diagrama comparativo en tres columnas mostrando los tres tipos de leyenda

6 Asistente de estilo con Ollama

UCR StarLite incorpora un panel de IA que sugiere atributos y estilos de visualización a partir de un modelo de lenguaje ejecutado íntegramente en local mediante Ollama [6], sin enviar ningún dato a servicios externos. El panel incorpora un botón flotante de activación, un selector de modelo que se puebla dinámicamente desde el endpoint /api/tags de Ollama al abrir el panel, mostrando solo los modelos realmente instalados en la máquina del usuario, un indicador de estado de conexión en tiempo real, un historial de mensajes, y cinco acciones rápidas: describir el dataset, sugerir una visualización, identificar el mejor atributo, analizar la vista visible y detectar anomalías.

6.1 Construcción del contexto y el prompt

La función getMapContext() construye un prompt con el nombre del dataset activo, el tipo de geometría, la lista completa de atributos con sus estadísticas de valores no nulos y top-k, y cuando es posible, las propiedades de una feature representativa de las visibles en el mapa mediante map.queryRenderedFeatures(). El sistema usa dos fuentes de estadísticas de forma ordenada: primero el endpoint /api/datasets/<n>/stats del servidor, y en su defecto el cálculo sintético statsFromFeatures(), lo que permite que el asistente funcione incluso sin conexión al servidor Flask, igual que el motor de estilado de la sección 5.1.

Este contexto se envía junto con el historial de conversación en cada llamada al endpoint /api/chat de Ollama, permitiendo respuestas coherentes con el estado actual del mapa. El prompt de sistema instruye al modelo para que responda únicamente con una estructura JSON con los campos attribute, viz, scheme, labelAttr, labelBg y reason, incluyendo instrucciones explícitas sobre qué esquemas de color usar según el tipo de dato (por ejemplo, categóricos para variables de tipo texto con pocas categorías).

6.2 Parseo y aplicación del estilo sugerido

El bloque JSON se extrae mediante una expresión regular (/```json\s*([\s\S]*?)```/) aplicada sobre el texto crudo de la respuesta, antes de cualquier escape HTML; cuando esta no encuentra el bloque, un mecanismo de fallback busca directamente una estructura {…} que contenga el campo "attribute", dado que distintos modelos de Ollama varían en el formato de salida, algunos añaden texto libre antes o después del bloque JSON, o no respetan exactamente la valla de código solicitada. Cuando el parseo tiene éxito se activa el botón Apply suggested style.

La función aiApplyStyle() valida el atributo sugerido contra la lista real de atributos del dataset, con búsqueda insensible a mayúsculas y por prefijo para tolerar pequeñas variaciones en la respuesta del modelo, calcula las estadísticas necesarias mediante computeStats() y statsFromFeatures(), y aplica el estilo directamente sobre las capas MapLibre mediante _dispatchStyle(). Los selectores del panel se sincronizan a posteriori para reflejar el estilo aplicado, manteniendo la coherencia visual entre lo sugerido por la IA y lo mostrado en los controles manuales (Figuras 14 y 15).

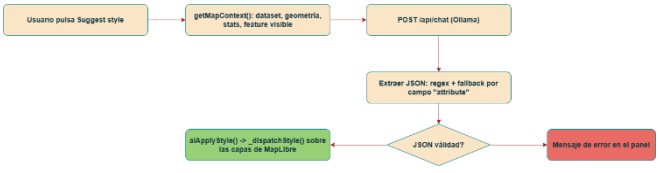


Fig 14. Flujo del asistente IA: contexto del mapa, llamada a Ollama, extracción del JSON y aplicación del estilo o mensaje de error.

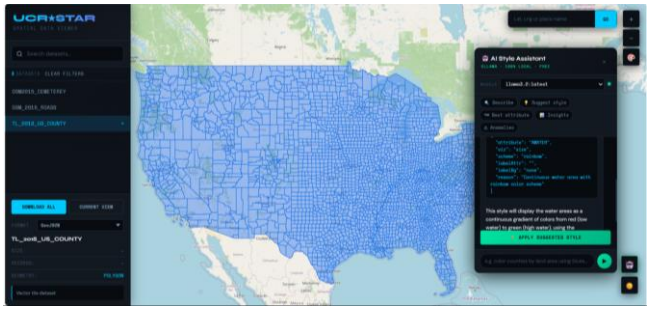


Fig 15. Panel AI Style Assistant: sugerencia estructurada en JSON para el atributo AWATER (modo size, esquema rainbow) y botón para aplicar el estilo sobre las capas vectoriales.

7 Limitaciones y trabajo futuro

El asistente LLM requiere Ollama en ejecución local con un modelo compatible; la variabilidad en el formato de salida de distintos modelos puede hacer fallar el parseo del JSON en casos extremos, mitigado por el mecanismo de fallback de la sección 6.2 pero no garantizado para todos los modelos. La descarga en Shapefile depende del soporte efectivo del backend de UCR-STAR para ese formato. El número máximo de seis capas simultáneas está limitado por la paleta de colores predefinida y la visibilidad del visor, y las etiquetas canvas pueden presentar retrasos de refresco al cambiar de dataset rápidamente, debido a la latencia del ciclo de renderizado de MapLibre GL JS frente al redibujado del canvas superpuesto.

Como trabajo futuro queda: validar la descarga en todos los formatos contra un backend de UCR-STAR en producción; ampliar el límite de capas simultáneas mediante generación dinámica de color en lugar de una paleta fija, lo que eliminaría la restricción actual de seis datasets; mejorar la robustez del parseo de la respuesta del LLM probando distintos modelos de Ollama y refinando aún más el prompt de sistema; y revisar la sincronización del canvas de etiquetas con el ciclo de renderizado de MapLibre al cambiar de dataset rápidamente. A más largo plazo, sería interesante explorar la generación automática de múltiples sugerencias de estilo alternativas por parte del asistente de IA, permitiendo al usuario comparar varias propuestas antes de aplicar una.

8 Conclusión

UCR StarLite alcanza un grado de madurez superior al planteado en el informe inicial. Las ocho fases del plan de trabajo se completaron, incorporando cuatro extensiones no previstas: gestión de múltiples capas activas, un motor de leyendas centralizado, una landing page rediseñada y mejoras de robustez en el sistema de estilado. El resultado es una herramienta web ligera, interactiva y extensible que cumple el objetivo principal: ofrecer un punto de entrada accesible a UCR-STAR con visualización vectorial de alto rendimiento, estilado cartográfico interactivo, búsqueda geocodificada, exportación espacial y un asistente de estilo basado en un LLM que se ejecuta íntegramente en local, sin comprometer la privacidad de los datos del usuario.

Más allá del resultado funcional, el proyecto demuestra que es posible construir una herramienta de exploración geoespacial competitiva sin recurrir a frameworks de frontend, apoyándose únicamente en los mecanismos nativos del DOM y la API de MapLibre GL JS. Esta decisión de diseño, junto con la arquitectura centrada en el cliente y la elección de Ollama como motor de asistencia local, hacen de UCR StarLite una base sólida, reproducible y fácilmente extensible para futuros trabajos dentro del UCR Big Data Lab.

Agradecimientos

Quiero agradecer a Eduardo Cesar Galobardes (UAB) por el seguimiento académico de este TFG.
Agradezco también al UCR Big Data Lab, y en particular a Ahmed Eldawy, por el acceso al repositorio UCR-STAR y por el apoyo durante mi estancia en UC Riverside.

BIBLIOGRAFÍA

[1] S. Ghosh, T. Vu, M. A. Eskandari y A. Eldawy, "UCR-STAR: The UCR Spatio-temporal Active Repository," SIGSPATIAL Special, vol. 11, n.º 2, pp. 34–40, 2019. DOI: 10.1145/3377000.3377005.

[2] T. Bahadori, S. S. Sarvepalli y A. Eldawy, "LASEK: LLM-Assisted Style Exploration Kit for Geospatial Data," PVLDB, vol. 18, n.º 12, pp. 5435–5438, 2025. DOI: 10.14778/3750601.3750690.

[3] S. Ghosh y A. Eldawy, "AID*: A Spatial Index for Visual Exploration of Geo-Spatial Data," IEEE TKDE, vol. 34, n.º 8, pp. 3569–3582, 2022. DOI: 10.1109/TKDE.2020.3025568.

[4] MapLibre Community, MapLibre GL JS – Documentación oficial. Disponible en: maplibre.org/maplibre-gl-js/docs/

[5] OpenStreetMap Foundation, Nominatim – Servicio de geocodificación. Disponible en: nominatim.org

[6] Proyecto Ollama – Plataforma para ejecución local de modelos de lenguaje. Disponible en: ollama.com

[7] Repositorio del proyecto. Disponible en: github.com/Mouad-El-Messaouri/TFG-UCRLite

- *E-mail de contacte: mouad.elmessaouri@gmail.com*
- *Treball tutoritzat per: Eduardo Cesar Galobardes (Àrea d'Arquitectura i de Tecnologia de Computadors)*
- *Curs 2025/26*

APÉNDICE

A1. ENDPOINTS PRINCIPALES DEL SERVIDOR FLASK

GET /<dataset>/<z>/<x>/<y>.mvt – teselas vectoriales servidas por VectorTiler.
GET /api/datasets, /datasets.json?q=, /datasets/<n>.json – descubrimiento y metadatos de datasets.
GET /api/datasets/<n>/stats – estadísticas de atributos (valores más frecuentes y recuento de no nulos).
GET /datasets/<n>/features.<fmt>?mbr= y POST /datasets/<n>/features.<fmt> – exportación de subconjuntos en GeoJSON, CSV o Shapefile, filtrados por MBR o por una geometría GeoJSON personalizada.
GET .../features/sample.json|.geojson?mbr= – registro representativo de una región, usado por el asistente IA cuando no hay features visibles en el cliente.

A2. LANDING PAGE



Fig 16. Landing page de UCR StarLite, con estadísticas del repositorio y acceso directo al visor y a la documentación.