Federated Querying of Heterogeneous Spatiotemporal Data over Polyglot Stores

Everett Lewark, Nathan Orwick, Paige Hansen, Ayush Adhikari, Sangmi Lee Pallickara, and Shrideep Pallickara
Computer Science Department, *Colorado State University*
Fort Collins, Colorado, USA
{elewark, nathan.orwick, p.hansen, ayush.adhikari, sangmi.pallickara, shrideep.pallickara}@colostate.edu

***Abstract*—Spatiotemporal datasets hold the key to understanding complex natural and human systems. Diversity in encoding formats, resolution, and storage architectures renders them difficult to use together. In this paper we take soil-related data as a central example, not because soils are unique, but because they exemplify the challenges of integrating and exploring longitudinal data for terrestrial phenomena. Our methodology and associated framework, codenamed HERMES, builds on the principle of polyglot persistence: surveys, vector data, shapefiles, and raster imagery remain in the storage systems best suited to them. Rather than forcing these data into a single repository, HERMES provides a unifying layer that allows queries to reach across them while preserving the performance benefits of each system. Because the order of evaluation matters, we employ heuristics to prioritize predicates most likely to filter strongly or benefit from native indexes, thereby reducing both movement of data and query latency. Finally, by caching results and reweighting heuristics through selective counterfactual evaluation, the system adapts as the data evolve: growing more efficient the more it is used. We present extensive benchmarks profiling multiple facets of our methodology, demonstrating its effectiveness and performance gains; for comparable tasks, HERMES outperforms Apache Sedona and GeoPandas by factors of up to 42x and 52x respectively.**

***Index Terms*—polyglot persistence, big data, spatiotemporal explorations, predicate reordering and heuristics, query planning**

I. INTRODUCTION

Interactive geospatial analytics increasingly require querying heterogeneous spatiotemporal datasets distributed across multiple storage engines, resolutions, and update frequencies. Users expect sub-second visual feedback while combining vector geometries, raster fields, and time-series measurements that reside in polyglot backends such as MongoDB, Postgres, Cassandra, and Redis. However, traditional spatial databases and federated query engines are not designed for this setting: they assume homogeneous storage models, batch-oriented execution, or cost models that ignore spatial resolution mismatch and reprojection overhead.

The core challenge is that query evaluation cost in this setting depends on much more than predicate selectivity. Spatial predicates operate over heterogeneous resolutions, often requiring reprojection or rasterization before evaluation. Intermediate results are spatial footprints (or scopes) rather than scalar tuples, and pruning must occur progressively across heterogeneous backends prior to tile materialization. Data movement costs depend on spatial extent and temporal window, while resolution alignment may dominate execution time even when predicate selectivity is high. These properties undercut classical relational optimization assumptions and necessitate a cost model that jointly reasons about spatial selectivity, resolution alignment, and backend latency.

Consider a representative query identifying regions where `soil moisture < 0.2` (continuous raster), `cropland class == corn` (categorical raster), `slope < 3 %` (derived raster), and `watershed intersects a selected county` (vector), within a specified chronological window. Evaluating this query requires coordinating predicates across heterogeneous storage engines and resolutions while progressively refining a shared spatial scope. Naïvely executing predicates in arbitrary order, or delegating evaluation to a single engine, often results in excessive reprojection, large intermediate masks, and unnecessary data transfer.

A key challenge in integrating such datasets lies in their heterogeneity across spatial, temporal, and structural dimensions. Spatial resolution, geographic coverage, and temporal granularity vary not only across sources, but often within the same dataset. Raster products, for example, may contain pixels at multiple scales, while vector and survey datasets introduce irregular geometries and attribute schemas. As a result, aligning values across datasets is non-trivial: a single pixel in one layer may correspond to multiple, non-overlapping units in another. This complexity is further compounded by variation across depth (e.g., soil horizons) and time, where observations span intervals ranging from hours to decades.

This representational heterogeneity is amplified by storage heterogeneity. Soil surveys, sensor streams, and gridded products are often distributed across relational databases, document stores, and distributed key–value systems, each exposing distinct query models, indexing strategies, and performance characteristics. Consequently, integrated analysis must reconcile both differences in data representation and differences in system behavior. Without such reconciliation, analyses risk producing inconsistent or misleading results. These challenges motivate the need for a unifying framework that treats representational and storage heterogeneity as first-class concerns in the query processing pipeline.

Problem statement. We address the design of an efficient, interoperable, and *polyglot* framework for executing cross-cutting spatiotemporal queries over heterogeneous datasets dis-

tributed across storage architectures and data models. Without such a framework, domain scientists and practitioners must manually stitch together results from separate systems, leading to brittle workflows, redundant data movement, and limited support for interactive, layered analysis.

At its core, the challenge is to support integrated querying and aggregation across datasets that differ in representation, origin, and observational domain. Such queries must operate over both administrative units (e.g., counties) and ecologically or hydrologically meaningful regions (e.g., Hydrologic Unit Codes), while preserving spatiotemporal consistency across layers. Without this users are left with fragmented views that obscure critical interactions across space, time, and depth.

A. Research Questions

The overarching research question that we explore is: *how to expose a coherent, queryable view over polyglot, multi-resolution soil datasets without centralizing the data or sacrificing responsiveness?* Within this, we explore:

RQ-1: *How can we reconcile differences in spatial resolution, coordinate reference systems, depth indexing, and units of measurement across diverse datasets such as rasters, vectors, and shape files?* This would lay the basis for unified querying.

RQ-2: *How can we support expressive, low-latency spatiotemporal queries that span relational databases, document stores, and key-value frameworks while minimizing data movement and ensuring semantic consistency?*

RQ-3: *What are the systems design and caching strategies that support responsive, exploratory interaction with layered datasets; where users can filter, overlay, or compute compound spatial relationships across variable levels of aggregation?* These questions decompose the overall challenge into data harmonization, query processing, and user-facing interaction.

B. Approach Summary

Our approach combines three core ideas: a spatiotemporal scoping layer that spans heterogeneous backends; a cost- and evidence-driven planner for federated predicate evaluation; and a layered visualization engine that renders query results as navigable spatiotemporal composites. HERMES adopts a *polyglot persistence* [1], [2] design, distributing datasets across storage systems based on their fitness to specific data modalities. Structured data reside in relational systems (PostgreSQL), semi-structured records and vector geometries in document stores (MongoDB), and large raster fields are tiled across distributed key–value stores (Cassandra, Redis) for parallel access. This polyglot organization allows each backend to specialize, but also introduces need for a unifying abstraction that spans them. More broadly, HERMES addresses a canonical cloud problem [3]–[5]: how to deliver interactive analytics over heterogeneous data services while minimizing data movement and preserving backend-specific optimizations.

We formalize a *spatiotemporal scope* abstraction in which each predicate progressively refines a shared spatial and temporal footprint. Every data element, irrespective of format or storage backend, is associated with a well-defined scope that serves as a unifying metadata layer across rasters, vectors, categorical attributes, and temporal observations. This abstraction enables alignment, federated querying, and systematic pruning across engines. Temporal reconciliation treats coarse datasets as valid over their coverage intervals and filters observations to query windows, ensuring consistency across resolutions.

Building on this abstraction, we design a federated query evaluation model in which sub-queries are pushed down to their respective data stores and executed natively. Each backend is queried natively (e.g., SQL, MQL, key-range lookups), leveraging its indexing strategy. The architecture also accommodates specialized indexing structures tailored to the capabilities of each store. These include relational range indexes and spherical indexes for geospatial operations. This native execution avoids premature data movement and exploits the optimization strategies of each backend. However, the integration of such results is a challenge. Disparities in query semantics, inconsistencies in indexing, and the absence of cross-store join capabilities complicate the synthesis of results across stores. Spatiotemporal predicates, in particular, require not only logical compatibility but also geometric alignment and temporal coherence which adds substantial coordination overhead. The role of our planner is to orchestrate predicate evaluation across backends while minimizing intermediate footprint expansion and alignment cost. By preserving the autonomy of each store while coordinating through the shared scope abstraction, HERMES achieves efficiency and interoperability without requiring rigid uniformity.

To this end, we implement predicate ordering strategies that explicitly target evaluation cost. For compound queries, predicates are reordered to maximize early pruning. We consider two complementary execution modes: (1) a sequential mode, where predicates are applied in an order that maximizes early pruning; and (2) a parallel mode, where predicates are evaluated independently and their resulting scopes intersected. These strategies allow the system to balance latency, resource utilization, and pruning effectiveness under varying workload conditions. Both strategies produce a bounded spatiotemporal region that drives downstream execution and visualization.

Finally, query results are rendered as visual artifacts, with each result associated with a well-defined spatial and temporal envelope. Geometries reflect complex spatial structures, including concavities, holes, and disjoint regions, while temporal envelopes capture when conditions hold. Temporal envelopes are paired with each spatial output(s) to allow users to understand not just *where* but *when* a pattern holds. These results are organized into layers for overlay and interactive exploration.

We ground and validate our approach in the domain of soils, while explicitly incorporating related drivers such as meteorology, land cover, and topography that shape soil conditions over space and time. This application domain is especially appropriate because it stresses exactly the forms of heterogeneity (*i.e.*, representation, scale, storage, and temporal cadence/frequencies) that HERMES is designed to reconcile.

C. Paper Contributions

Existing systems either operate within a single storage model (e.g., PostGIS [6]), assume batch-oriented distributed processing (e.g., Spark-based spatial systems [7], [8]), or lack cost models that account for spatial resolution mismatch and reprojection overhead. Federated query engines [9] address cross-source joins but do not reason about progressive spatial footprint pruning or interactive tile materialization. HERMES targets this intersection of query planning & spatial execution.

1) We formalize a spatiotemporal scope model that treats predicate evaluation as progressive footprint refinement across diverse data modalities reconciling disparities in coordinate systems, temporal granularity, and depth indexing. This is the core interface through which otherwise incompatible stores participate in joint queries.
2) A predicate-aware federated query engine that dynamically reorders and executes spatial, temporal, and thematic/range filters across polyglot backends, preserving native execution of individual predicates and minimizing data movement to achieve low-latency query evaluation while exploiting storage-specific optimizations.
3) A layered visualization framework that renders query results as spatiotemporal composites and ties them back to their originating predicates, enabling interactive exploration and semantically meaningful overlays even across fragmented or multi-resolution datasets.
4) We design an adaptive cost model for cross-engine predicate ordering that integrates index-based cardinality estimates, historical latency, and resolution alignment.

Translational impacts: While HERMES is motivated by geospatial analytics, its core ideas extend well beyond that. In particular, the spatiotemporal scope abstraction and adaptive planning strategy apply to any federated setting in which spatial masks, heterogeneous storage engines, and progressive visualization pipelines must work together. More broadly, HERMES can support domains such as climate modeling, disaster response, and environmental monitoring, where satellite imagery, sensor streams, and survey products must be integrated across formats and resolutions. These same capabilities also support layered querying and interactive visualization in real-time decision-support settings, including wildfire risk management and public health monitoring.

II. METHODOLOGY

HERMES decomposes query execution into three tightly coupled stages: (1) abstraction and alignment of heterogeneous data into a common spatiotemporal representation; (2) cost-aware planning and execution of predicates across distributed backends; and (3) progressive materialization and visualization of results. Fig. 1 and 2 provide an architectural overview.

A. Spatial data staging and harmonization [RQ-1]

To query datasets drawn from diverse sources without incurring the cost of repeated transformations at runtime, it is necessary first to harmonize the way those datasets are stored. For raster data, consistency in both the tiling scheme and

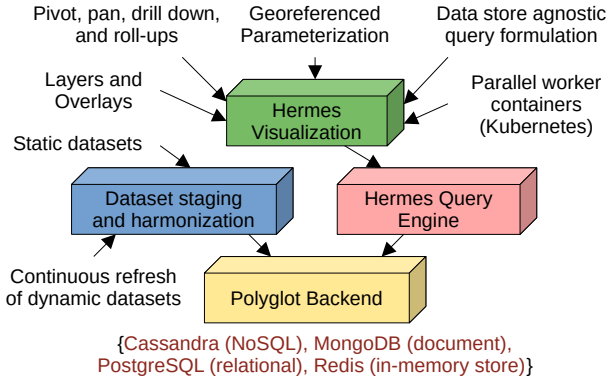


Fig. 1. Overall system architecture of HERMES

the coordinate reference system (CRS) ensures that tiles can later be compared pixel-by-pixel during query evaluation. We leverage the Geospatial Data Abstraction Library (GDAL) [10] through Rasterio [11] to unify the CRS when ingesting data into the Cassandra database. However, raster datasets arrive in a wide variety of spatial resolutions: ranging from meters to kilometers per pixel. To accommodate such variation, we allow datasets to be tiled at different scales, while at the same time imposing clear constraints on which resolutions are permitted. These constraints preserve comparability and keep subsequent queries coherent and efficient.

Storing spatial data within a key-value systems such as Cassandra or Redis requires consistent, discrete spatial identifiers. We utilize the quadkey tiling scheme [12], which divides the globe into a series of increasingly fine grids. Similar to Geohash-based spatial encodings [13], quadkeys produce text strings that uniquely identify spatial regions by descending into a quadtree. Each successive zoom level contains four times the number of tiles as the previous, so while the quadkey 0 represents the upper-left quadrant of the map, 02 refers to the lower-right portion of that quadrant, occupying one sixteenth of the map. These tiles align with the spatial regions in use by common visualization frameworks such as deck.gl or Leaflet.

We generate tiles of our raster datasets to this type of grid, and size each tile so that its resolution is a power of two along each axis. To allow users to zoom out of large datasets without having to retrieve an excessively large number of tiles, we additionally precompute and store multiple overview levels that downsample the original dataset. For example, a dataset that includes tiles at a zoom level of 12 also includes overviews at zoom levels 10 and 6. Quadkeys, and if relevant, dates, form composite keys identifying compressed GeoTIFF, WebP, or run-length encoded (RLE) tiles within Redis and Cassandra.

Datasets update at varying temporal intervals and this diversity must be managed to ensure consistency. Some datasets, such as gridMET [14] (meteorological data) and Soil Moisture Active Passive (SMAP) [15], update daily. Others, like the National Land Cover Database (NLCD) [16] and Cropland Data Layer (CDL) [17], are updated only once a year. To keep frequently updated datasets current, we rely on automatic

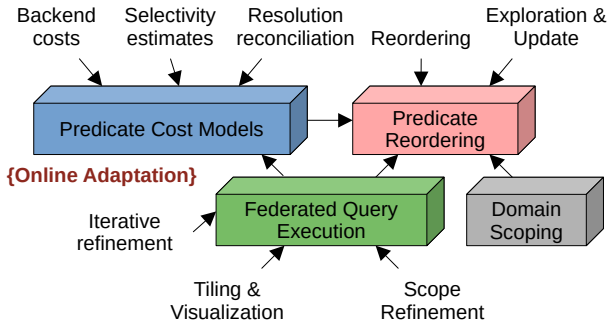


Fig. 2. The query engine, comprising cost modeling, predicate reordering, and federated query execution.

ingestion. Daily datasets may lag behind by a few days due to delays in data retrieval and processing, but we ensure that our copies remain within a one-day delay of the source. In the case of NLCD, the latest version of the dataset typically is delayed by a year, so the latest release (August 2025) covers land cover categories in 2024.

Some satellite data sources, such as NASA's SMAP, update frequently, but their spatial coverage on any given day is incomplete due to the satellite's orbital path. To manage these gaps, we respect the "no data" values in GeoTIFF tiles and render those pixels as transparent; this ensures that the missing data does not distort the visual output.

Finally, vector datasets, while rich in detail, can overwhelm rendering engines when their shapes are too complex. To ensure smooth interactivity, we employ both raster versions of these datasets and vector shape simplification techniques. This dual approach allows HERMES to maintain performance without sacrificing the accuracy of visual representation.

B. Federated query execution across polyglot stores [RQ-2]

Database servers differ in both the formats they support and the tradeoffs they impose. NoSQL systems often excel at distribution and sharding but may require data to be denormalized in order to accommodate diverse query patterns. Relational systems, on the other hand, provide more structured support for joins across normalized tables, but may not scale as readily under high-volume workloads. This need to combine strengths rather than rely on any single system is what motivates HERMES' design choice of polyglot persistence. A geospatial querying system intended for federated analysis must therefore be able to draw upon multiple kinds of data stores, each chosen for the role it serves best. In our methodology, raster datasets are stored in Cassandra whose distributed architecture allows us to manage large, tiled images efficiently. Vector polygon data are stored in MongoDB and PostGIS, with PostGIS providing additional strength where queries rely on relational joins across multiple tables. This combination allows us to harness the strengths of each system while mitigating their individual weaknesses.

Having established where vector data reside, we turn to how they can be queried efficiently. To make vector queries efficient we rely on the indexing features of these systems. Indexing schemes such as 2dsphere and R-tree enable us to retrieve only those shapes that fall within a region of interest. By limiting the candidate set through these indexes, we reduce the need for more expensive geometric comparisons that would otherwise dominate query costs.

Cross-cutting queries that involve both vector and raster data introduce a further challenge: they require both data types to be aligned to a common frame of reference. To achieve this, vector datasets are rasterized onto the same tiling grid used for raster datasets. Here, spatial indexing again plays a role: only polygons that overlap the raster tile being computed are retrieved. This ensures that each tile's query result can be calculated efficiently. Queries involving the same variable from a vector dataset reuse previously-cached rasterized tiles within Redis, reducing the amount of times rasterization is performed. This alignment is what allows cross-cutting queries to be evaluated coherently across raster and vector datasets, while still preserving efficiency. The result is a unified pipeline that aligns heterogeneous data and enables layered queries to execute consistently and efficiently.

C. Query semantics and spatiotemporal scopes [RQ-1, RQ-2]

We model each user request as a declarative query Q that specifies (1) a set of predicates over spatial, temporal, and thematic dimensions, and (2) an output specification that determines the shape of the result (e.g., tiled rasters, polygons, or aggregated summaries). At the core of this model is a collection of predicates $(P_1, P_2, \ldots, P_n)$, each of which expresses a constraint along one or more axes: spatial (e.g., "within this polygon"), temporal (e.g., "during 2018–2020"), or thematic/variable-specific (e.g., "soil moisture $> 30\%$"). In its most basic and commonly used form, a query is interpreted conjunctively: $Q = P_1 \wedge P_2 \wedge \cdots \wedge P_n$. Each predicate P_i is evaluated independently against its relevant dataset(s). This evaluation yields a corresponding *spatiotemporal scope*, denoted $S_i = (\Omega_i, \tau_i)$. Here, $\Omega_i \subseteq \mathbb{R}^2$ represents the spatial footprint over which the predicate holds, and $\tau_i \subseteq \mathbb{T}$ specifies the time interval (or collection of intervals) during which the condition is satisfied. Intuitively, S_i captures "where and when" the predicate is true, independent of how the underlying data are stored or indexed.

Under this conjunctive interpretation, the full query scope S_Q is defined as the intersection of individual predicate scopes:

$$S_Q = \bigcap_{i=1}^{n} S_i = \left(\bigcap_{i=1}^{n} \Omega_i, \bigcap_{i=1}^{n} \tau_i \right).$$

This formulation reflects the intuition behind queries that combine multiple criteria across domains. Consider, for example, the compound condition: "soil moisture $> 30\%$ *and* soil texture = sandy-loam *and* crop rotation is winter wheat/corn." Each predicate narrows the data within its domain; and, their intersection yields the precise spatiotemporal envelope where all conditions *simultaneously* hold. This envelope becomes the common frame of reference that determines which tiles, rows, and geometries must be retrieved from each backend.

Although we focus on conjunctive queries, the model naturally extends to disjunction and negation. A disjunction such as $P_1 \vee P_2$ is represented as the union of scopes $S_1 \cup S_2$, while a negated predicate $\neg P_i$ is interpreted as the complement of its scope within a user-specified or default domain.

The structure of each scope component reflects the nature of the data it represents. The spatial extent Ω_i may take the form of vector geometries (polygons with possible concavities or interior holes) or raster-based masks aligned with gridded surfaces depending on the modality of the source. The temporal component τ_i may span contiguous intervals or consist of multiple disjoint fragments. It may also vary in resolution, ranging from hourly sensor observations to annual survey summaries. Temporal reconciliation ensures that these differences are harmonized consistently with query intent. For example, annual products may be treated as valid over an entire calendar year, whereas event-based observations are clipped to query-specific windows.

The resulting query scope S_Q plays a dual role. First, it constrains evaluation: federated sub-queries are only issued to backends whose data intersect S_Q, and only for the tiles or geometries that fall within this envelope. Second, it informs result construction: depending on the output specification, HERMES can materialize S_Q as a set of raster tiles, as a collection of vector geometries, or as an aggregated summary (e.g., averages, counts) computed over Ω_Q and τ_Q. Representing scope as a structured pair not only enforces logical consistency, but also underpins system-level features such as predicate reordering for cost-based query planning, scope-driven filtering, and the layered visualization of intermediate and final results. In the subsequent subsection, we exploit these spatiotemporal scopes to drive predicate-aware planning over polyglot backends, and later we use the same scopes to generate the visual layers presented to the user.

D. Predicate-aware query planning [RQ-2, RQ-3]

Having defined a query Q as a conjunction of predicates $P_1 \wedge P_2 \wedge \cdots \wedge P_n$, each of which yields a spatiotemporal scope $S_i = (\Omega_i, \tau_i)$, we address the question of how these predicates should be evaluated. Because each predicate may involve different data sources, indexing strategies, and computational costs, the order and manner in which they are executed play a key role in the efficiency of query processing. In our methodology, we explore two different approaches: *pipelined* evaluation and *concurrent* evaluation.

In the pipelined strategy predicates are applied sequentially, one after the other, in a carefully chosen order. The spatiotemporal scope resulting from each predicate is passed forward to constrain the evaluation of subsequent predicates. Let $S_i = P_i(D_i)$ denote the scope produced by evaluating predicate P_i against its corresponding dataset D_i. Then the composite query scope is built iteratively as follows:

$$S_Q^{(1)} = S_1, \quad S_Q^{(i+1)} = S_Q^{(i)} \cap S_{i+1}, \quad \text{for } i = 1, \ldots, n-1.$$

This model supports early pruning of the search space because predicates with high selectivity can eliminate irrelevant spatial and temporal regions before later stages incur additional computational cost. However, the success of pipelining depends heavily on predicate ordering; improperly ordered chains can lead to unnecessary work and delayed convergence.

Our concurrent strategy evaluates all predicates independently and in parallel. Each P_i is resolved on its own, producing a scope S_i without prior constraint from other predicates. The final scope is then derived by intersecting the independently obtained results: $S_Q = \bigcap_{i=1}^{n} S_i$. This approach can leverage parallel compute resources to reduce overall latency, particularly when the predicates operate over disjoint datasets or when query planning time is a limiting factor. However, this may be less efficient in environments with limited resources or when predicates exhibit wide variation in selectivity or evaluation cost.

We rely on heuristics to shape our query evaluation strategies. Some operations though correct in principle can be costly in practice; as such they must be scheduled accordingly. Raster queries that reference external tables in PostgreSQL, for e.g., tend to be slower because they involve complex joins; these we defer to later stages of the query. Predicates that operate on vector datasets pose a similar challenge: before they can be joined with raster results, they must first be rasterized, a step that adds computational expense. By recognizing such patterns, our heuristics allow us to order evaluation so that it minimizes unnecessary overhead while preserving correctness.

When evaluating predicates in a pipelined strategy, the order in which they are evaluated significantly influences both performance and responsiveness. Predicates have variability in their pruning power: some eliminate large portions of the data early on while others contribute little to the pruning of the spatiotemporal scope. We prioritize those predicates that are expected to perform the greatest amount of filtering. We associate with each predicate P_i a *selectivity score* $\sigma(P_i) \in [0, 1]$, where lower values indicate stronger filtering potential. The query engine defines an ordering function π over the predicate set $\{P_1, P_2, \ldots, P_n\}$ such that $\sigma(P_{\pi(1)}) \leq \sigma(P_{\pi(2)}) \leq \cdots \leq \sigma(P_{\pi(n)})$. This allows the most selective predicates to be evaluated first, thereby reducing the amount of data that must be considered by subsequent stages.

To guide this ordering, we compute a scalar priority score $h(P_i)$ for each predicate P_i. This score is defined as a weighted combination of three core attributes:

$$h(P_i) = \alpha \cdot c(P_i) + \beta \cdot (1 - \iota(P_i)) + \gamma \cdot r(P_i)$$

In this formulation, $c(P_i)$ reflects the estimated computational cost associated with evaluating the predicate. The binary indicator $\iota(P_i)$ returns 1 when the predicate can be resolved using an available index structure, and 0 otherwise. The term $r(P_i)$ introduces a penalty for operating over high-resolution or dense data formats. The weights α, β, γ are tunable parameters that allow the system to tailor the scoring function based on prevailing workload conditions and data organization.

From this composite score, we derive a normalized selectivity score: $\sigma(P_i) = \frac{h(P_i)}{\max_j h(P_j)}$. This formulation produces

a dimensionless value between 0 and 1; this allows comparison across predicates that differ in form and function. The denominator, which is the highest priority score across all predicates within the query, serves as a reference for normalization. Lower values of $\sigma(P_i)$ reflect greater expected pruning strength or computational efficiency. The query engine evaluates predicates in ascending order of $\sigma(P_i)$, so that those predicates offering the most beneficial combination of cost and selectivity are executed first. By doing so our methodology strikes a balance between the filtering strength of each predicate and the execution efficiencies made possible by data-store-native indexing. This cost-aware reordering mechanism enables our query planner to structure evaluation paths that are both semantically correct and computationally fast.

Counterfactual exploration of the predicate space Certain predicates are likely to defy estimation. This occurs in cases where: (1) no meaningful statistics are available, (2) the predicates are constructed over dynamically generated attributes, and/or (3) the distributions involved are particularly skewed. For these predicates, we conservatively assign $\sigma(P_i) = 1$ and $h(P_i) = \infty$, which guarantees that they will be scheduled toward the end of the evaluation pipeline. This conservative placement prevents insufficiently-characterized predicates from dominating the plan to the detriment of those with well-understood performance profiles.

Although this strategy provides a robust basis for evaluating and ordering predicates, it is still an approximation. The scope that each predicate sees depends directly on those that precede it. In order to empirically assess and refine our heuristics, we introduce a hybrid counterfactual evaluation framework.

For a small and configurable subset of queries, we evaluate each predicate independently against the full dataset. This process yields predicate-local scopes that reveal a predicate's intrinsic filtering power, independent of any specific execution sequence. These scopes are cached and reused to simulate alternative execution orders. By intersecting these scopes in various permutations, we estimate the impact that reordering would have had on the resulting spatiotemporal extent, the volume of data processed, and the associated response time.

For the remainder of the workload, we interpolate from the small number of fully evaluated queries. This allows us to refine predicate weights and improve future execution strategies without requiring exhaustive recomputation. As new data is introduced, existing data is modified, or obsolete data is removed the system automatically revisits and refreshes only the affected predicate-local scopes. This selective updating allows our methodology to accommodate the continual evolution of the data space without requiring manual tuning or full reanalysis. Over time, the system moves from relying solely on heuristics to integrating direct empirical evidence.

E. Multi-resolution data layering [RQ-1, RQ-3]

Users often wish to compare datasets visually in order to perceive broader spatial patterns and correlations at a glance. For this reason we allow raster and vector datasets to be placed in layers above one another. Users can drag layers to alter their order or adjust their opacity to let underlying patterns show through. Unlike the side-by-side slider view, which isolates datasets for comparison, this layered interface emphasizes *composition*. To render raster datasets within a viewport, requests are divided into tiles. When a tile is requested the server queries the relevant datasets and retrieves the raster data corresponding to the region of interest. Partitioning requests in this fashion allows the system to distribute traffic among multiple workers leveraging parallelism and reducing latency.

Responsiveness is further improved through server-side caching. Tiles that have been previously requested are stored in a distributed Redis cluster, while intermediate zoom levels are synthesized by combining and downscaling finer-grained tiles. A trie data structure maintains tile relationships in the cache, allowing us to minimize redundant queries to the Cassandra database. On the client side, rendering performance is enhanced by GPU acceleration through deck.gl [18]. Raster tiles are loaded from the center outward so that the most relevant regions appear first. The client also computes likely movements of the user by prefetching tiles along their projected trajectory. Together these mechanisms create an interactive environment in which users can filter, overlay, and explore spatiotemporal data with low latency and visual clarity.

F. Interactive visualization of layered query results [RQ-3]

The power of layered visualization reveals itself most fully in queries that cut across multiple raster or vector datasets. Such queries allow users to identify areas of interest; for example, by examining weather conditions as they intersect with agricultural regions within a specified time range. To achieve this, queries augment the ordinary loading process: it retrieves multiple datasets simultaneously and processes the resulting tiles according to the query predicates. We improve parallelism during query evaluation by splitting the task across workers using round-robin task assignment. Server containers running in a Kubernetes [19] cluster each host multiple worker processes to handle incoming requests.

Once the tiles have been retrieved on the backend, HERMES evaluates them pixel by pixel against the user's criteria. A query seeking regions where precipitation exceeds a threshold, for instance, is executed as a greater-or-equal comparison applied to each pixel of the relevant tiles. These operations produce Boolean raster masks, which are then combined using logical operators such as AND/OR to express intersections or unions of conditions. When tiles differ in resolution, the system reconciles them automatically by upscaling the lower-resolution tiles with nearest-neighbor interpolation.

Before the results are sent to the client, the system colorizes each query-result tile. Pixels that satisfy the user's condition are marked in bright green, while those that do not are rendered fully transparent. The resulting images are compressed using the lossless WebP [20] format and then transmitted to the browser for display. By choosing WebP over PNG, the system reduces the volume of data that must be transferred, thereby improving responsiveness while maintaining visual fidelity.

III. Performance Benchmarks & Discussion

Our performance benchmarks are conducted over a large number of spatiotemporal datasets (see section III-A) stored in distributed cluster of machines using storage frameworks such as PostgreSQL, MongoDB, Cassandra, and Redis. Our evaluation is organized around five distinct questions: planner quality, interactive latency, cache reuse under exploratory navigation, external baseline comparison, and bandwidth efficiency of rendered query masks. We separate these because HERMES is not intended as a single-purpose batch engine; rather, it is designed for interactive, federated exploration over heterogeneous stores. Accordingly, some benchmarks report end-to-end viewport latency, others report per-tile latency, and others isolate specific subsystems such as caching or compression. Throughout, our aim is not merely to maximize throughput in the abstract, but to characterize the latency profile that governs whether a layered exploratory workflow remains responsive.

A. Datasets, area of interest, and storage framework selection

We focus on the contiguous United States (CONUS) using multiple publicly available datasets. SSURGO (Soil Survey Geographic Database) is a dataset published by the USDA that describes soil properties, including soil texture at multiple depths. The dataset holds over 320,000 map unit regions containing a total of 36 million polygons and 3.6 billion points. Since GeoPackages are interoperable with the SQLite format, SSURGO data is suited for storage in PostGIS.

For vector data cases where relational table support is not necessary, we use MongoDB, which provides document-based storage and a well-rounded set of geospatial operations. We store many datasets such as EnviroAtlas and the associated Watershed Boundary Dataset in this database. This allows us to leverage MongoDB's built-in support for data replication to ensure redundancy in case of node failure.

Finally, we use Cassandra as our primary store for raster datasets such as the National Land Cover Database (NLCD), Cropland Data Layer (CDL), and Soil Moisture Active Passive (MAP) because this database provides functionality for key-value replication and data dispersion at scale, making it well-suited for the large quantities of images involved. Quadkey indexing for tiles integrates smoothly with key-value storage in this manner because complex join operations are not required.

TABLE I
DATASETS USED FOR HERMES BENCHMARKS

Dataset	Storage	Type	Resolution	Size
gridMET [14]	Cassandra	Raster	4 km	312 GB
SMAP [15], [21]	Cassandra	Raster	9 & 36 km	482 MB
CDL [17]	Cassandra	Raster	30 m	77.5 GB
NLCD [16]	Cassandra	Raster	30 m	87 GB
SSURGO [22]	PostGIS, Cass	Both	30 m raster	150 GB
EnviroAtlas [23]	MongoDB	Vector		2.9 GB

B. Experimental setup

These benchmarks were run on a cluster of 25 HP DL60 g9 servers using AlmaLinux 8.10. These servers ran Intel Xeon E5-2620 v4 2.10GHz processors and had 64 GB of RAM. Across these machines, we hosted database nodes including Cassandra 4.1.5, MongoDB 8.0.5, PostgreSQL 17.0, and Redis 7.2.5. Apache Sedona benchmarks used Sedona version 1.7.2, Apache Spark version 3.5.1, and Apache Hadoop version 3.4.0. For visualization-related benchmarks, we ran the client frontend on a workstation with a 24-core 13th Gen Intel i9-13900K processor, 64 GB of RAM, and a 3840x2160 display. We ran three server containers, each with 16 worker processes.

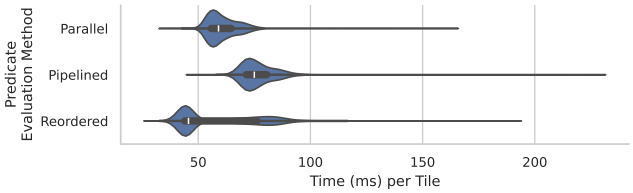


Fig. 3. Leveraging the HERMES heuristic pipeline reordering approach leads to the quickest response time per tile with complex queries when compared to running predicates in parallel or sequentially without optimization.

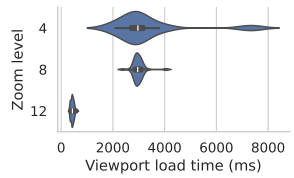
C. Predicate reordering and pipelining [RQ-2, RQ-3]

Fig. 3 shows the performance of impact of predicate-aware reordering to query performance. Three predicate planning strategies were profiled: running each predicate in parallel, running predicates as a pipeline, and leveraging the HERMES query planner to reorder the predicates. Parallel may seem at first like a preferential approach, but reordering the predicates can more effectively prune the search space with each step. These queries may be user specified, which also makes predicate reordering important. This allows a response to the user to remain accurate but be more interactive.

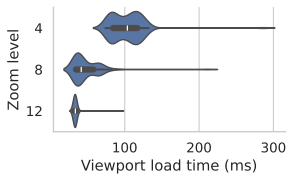
To benchmark this reordering, the following query was used: select areas with specific soil textures (e.g. loamy sand, sandy loam, silty clay loam); where corn is grown, the land is grassland, or the land is idle cropland; and HUC 12 watershed locations have a mean manure application value greater than 0.5 kg/ha/year. This complex query involves vector and raster data from three data stores (PostGIS, MongoDB, & Cassandra). The response times per tile for each reordering method are shown in Fig. 3. The lowest average response time is achieved by the HERMES heuristic reordering approach (mean 58.7, median 45.7, SD 17.9 ms). This approach seems to be bimodal, which is likely caused by tiles that could not leverage early exiting or extra pruning from reordering. The next fastest method was the parallel approach (mean 60.6, median 59.1, SD 7.3 ms), followed by the basic pipelining strategy (mean 76.9, median 75.0, SD 10.1).

D. Layering & aligning multi-resolution data [RQ-1,RQ–3]

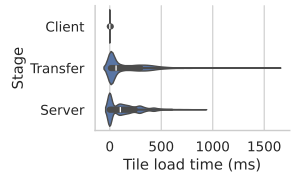
We measured the speed of client caching (Fig. 4) by repeatedly showing and hiding three raster layers 300 times while keeping the displayed map region fixed. We used three datasets: 36km SMAP, 4km gridMET, and 30m CDL. As shown in Fig. 4b, mean load times were fastest with caching enabled at zoom level 12 (34.39 ms, SD 6.94), followed by



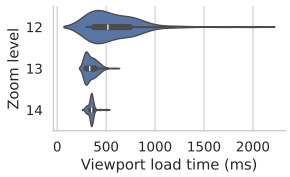
(a) Raster layer loading w/o caching



(b) Raster layer loading w/ caching

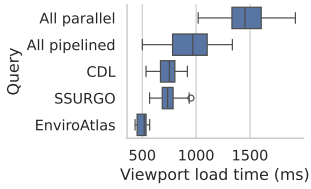


(c) Non-cached tile loading by stage

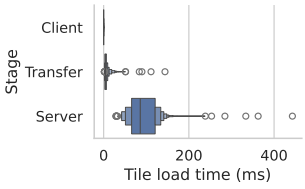


(d) SSURGO load time

Fig. 4. (a), (b) Showing and hiding raster layers on the client is accelerated by client- and server-side caching. (c) Most non-cached tile load time is spent during server and network transfer stages. (d) Loading varied vector regions for SSURGO leveraged less caching, and time varied more by zoom level.



(a) Time to load queries in viewport



(b) Pipelined tile load time by stage

Fig. 5. (a) Pipelined query evaluation improved response time when evaluating a federated query across multiple datasets, such that its response time was lower than the three constituent datasets combined naively. (b) The bulk (roughly 90ms) of pipelined load time is due to server-side processing, while latency from client-server I/O and client decoding is considerably shorter.

level 8 (48.13 ms, SD 21.77), and slowest at level 4 (103.22 ms, SD 25.17). This is the result of different tile counts being loaded, since at high zoom levels, lower-resolution datasets require fewer tiles to cover the viewport. **Caching provided over an order-of-magnitude speedup:** without it, zooms 12, 8, and 4 took 440.38 (SD 66.91), 2988.85 (SD 248.79), and 3339.06 (SD 1427.14) ms respectively (Fig. 4a).

We also separately measured the performance of loading vector data in the viewport (Fig. 4d). We loaded SSURGO vectors at three different zoom levels and 100 different locations. The resulting load time for each viewport directly related to the zoom level used, since lower zoom levels loaded larger areas. This demonstrates the utility of raster proxy data for vector datasets, since using rasterized SSURGO for zoom $<$ 12 allows us to maintain interactivity and leverage tile caching.

E. Rendering layered spatiotemporal query results [RQ-2/3]

Next, we measured the speed of evaluating a multi-part query and rendering it on the frontend. This provides us more information on what latency an end user would perceive. Again, we searched for areas matching the query in section III-C because it leveraged the federated nature of HERMES, including a mixture of data formats across three databases.

We evaluated five versions of the query: one using a parallel evaluation, one using pipelined, and three that only access one dataset from the original query. For each variant, we loaded 100 different viewports and 5,178 tiles. Fig. 5a shows the results. Overall, the pipelined approach performed best for evaluating the full query, loading a viewport in a mean 928.90 ms (SD 205.10). Conversely, the parallel approach took 1471.72 ms (SD 203.33). This result showcases the benefits of the short-circuiting possible in pipelined evaluation, since fewer datasets need to be queried in the event a tile is found empty. Most per-tile latency for pipelined queries occurred server-side (Fig. 5b), the average being 91.90 ms (SD 33.66).

F. Apache Sedona [RQ-2, RQ-3]

To compare our approach with Apache Sedona, we will use the same query and data: select areas from the 2023 NLCD and CDL datasets that were cultivated cropland *and* where corn was grown. This is more difficult than only filtering one dataset because it involves a geospatial union of the results from each. To match the concurrency level of a web client, we used six requestor threads for Hermes, and six Spark workers.

TABLE II
COMPARISON OF HERMES QUERY SPEED VS. APACHE SEDONA

Zoom	Tile count	HERMES		Apache Sedona	
		Mean	SD	Mean	SD
6	169	**0.63s**	0.08	26.73s	1.70
10	31,088	135.37s	22.73	401.84s	34.50
12	486,810	2353.78s	562.92	**OOM**	

At the coarsest zoom level, our approach is 42x faster than Apache Sedona with the same raster tiles. The startup and overall overhead for Apache Spark/Sedona are higher than that of the HERMES approach, not lending well to immediate results. Faster results are preferred so that user queries are interactive, especially since there is no difference in accuracy between these methods. This benchmark was averaged over ten runs, and the startup time for Apache Spark was not included.

Apache Sedona is able to scale well as the zoom level increases, due to Apache Spark's ability to leverage data locality through the Hadoop File System. Even with Sedona leveraging data locality for Spark task executor placement, the HERMES approach is nearly 3x faster at zoom level 10. At zoom level 12, the Spark executors failed due to out of memory errors. HERMES completed the query at zoom level 12 for the contiguous United States without errors. We manually verified the two level 6 results matched using QGIS.

G. GeoPandas & Rasterio [RQ-2, RQ-3]

Table III compares the HERMES approach for a query between raster and vector datasets with using two Python packages allowing for this capability. GeoPandas [24] is a powerful library for working with vector datasets in ways that are intuitive for users comfortable with Pandas, and Rasterio allows for a Pythonic way to interact with capabilities provided

through GDAL. Rasterio supports shape masking, which may be leveraged to compute intersections for an *and* query.

TABLE III
COMPARISON OF HERMES QUERY SPEED VS. GEOPANDAS & RASTERIO

Zoom level	HERMES		GeoPandas & Rasterio	
	Mean	SD	Mean	SD
6	**1.08s**	0.48	57.05s	1.61
10	110.16s	14.94	728.69s	1.15

The query benchmarked is the following: select areas that are low-density developed (NLCD/raster data) *and* within a county where the median age is greater than 40 years (county shapes/vector data). To evaluate this query using GeoPandas and Rasterio, shapes that geospatially intersect with a tile's boundary are loaded into a GeoDataFrame and filtered to those matching the criteria. These shapes mask the raster tile loaded using Rasterio, and then filter to pixels with the specified value, yielding the intersection between datasets. Mean time and standard deviation are computed across ten runs, with six threads for both approaches, and visual verification as in III-F.

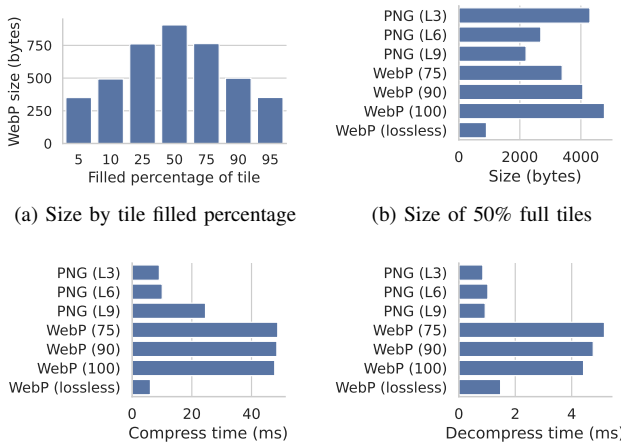
At zoom level 6, a coarse zoom level near the default view of the visualizer application, the HERMES approach is 52x faster than using GeoPandas and Rasterio. This allows for interactive queries covering a large area. At zoom level 10, tiles are much smaller, and the HERMES approach is 6x faster than using GeoPandas and Rasterio, on average. Note that the user would not wait this entire time with HERMES, as the viewport would be restricted at that zoom level and tile results would stream as they complete.

H. Compressing raster query results [RQ-3]

Since HERMES also returns raster query results as images, this means compression formats can have a substantial impact on bandwidth requirements between client and server. To benchmark the efficiency of various image formats on Boolean query result masks, we generated 1,000 fractal noise images. Each image combined eight octaves of white noise to produce a 256x256 tile, with noise frequency halved and amplitude doubled at each octave. This noise approximates the continuous spatial variation observed in geospatial datasets.

We thresholded these resulting images at several percentiles (5, 10, 25, 50, 75, 90, and 95) to simulate queries covering different proportions of the map. After thresholding, we applied a fixed color scheme to these masks (green for matching pixels, transparent for non-matching). Finally, we compressed them using 3 different compression levels for PNG (3, 6, 9) and 4 levels for WebP (75%, 90%, 100%, and lossless).

Under these conditions, we found that **lossless WebP produced images drastically smaller in size than those from PNG and lossy WebP** (Fig. 6b). Additionally, tiles composed of 50% matching pixels produced the largest file sizes (Fig. 6a), since they had fewer large, contiguous areas of consistent color. These images averaged 906.73 bytes (SD 216.81) in size for lossless WebP, while other formats were above 2 kB, such as 2204.51 bytes (SD 424.04) for PNG level 9. SSIM for



(a) Size by tile filled percentage

(b) Size of 50% full tiles

(c) Compression time (50% full)

(d) Decompression time (50% full)

Fig. 6. (a) Tile images were the least compressible when 50% full, since this introduced more spatial variation. (b) For these images, lossless WebP produced smaller mean file size than PNG and lossy WebP. (c), (d) Lossless WebP was faster than lossy to compress, but slower than PNG to decompress.

lossy-compressed tiles was consistent for lossy WebP, with a mean of 0.282 (SD 0.041) across 75%, 90%, and 100% quality settings, and an MS-SSIM value of 0.591 (SD 0.075).

Finally, the benchmarks in this section are intended to establish feasibility and interactive advantage, not to exhaust every axis of systems comparison. The paper emphasizes representative federated workloads, steady-state exploratory interaction, and cross-store execution behavior. A broader scaling study targeting more extreme workloads is an important next step; however, current results already show that predicate-aware planning, cache reuse, and native pushdown across polyglot stores materially improves the latency regime most relevant to interactive layered analytics.

IV. RELATED WORK

Querying and analyzing spatiotemporal data have been explored from several directions: (1) spatial data processing; (2) indexing-centric storage systems; (3) scientific platforms for voluminous arrays; (4) ontology-based integration across heterogeneous repositories; and (5) streaming and summarization using sketches and wavelets. Each addresses part of the broader challenge of integrating across storage backends, evaluating cross-cutting queries, and adapting as data evolve.

Distributed spatial data processing has been extensively studied in systems built on Hadoop and Spark. Efforts such as SpatialHadoop [25] and Hadoop-based spatial indexing and analytics [26] extended MapReduce with spatial partitioning, indexing, and query primitives to support scalable batch processing over large geospatial datasets. GeoSpark, later evolved into Apache Sedona [7], [27], advances this line of work by leveraging Spark's in-memory execution model to provide a richer set of spatial operators with improved scalability and expressiveness. Systems such as PostGIS [6] provide rich and query support within a single engine, but assume centralized data. However, these systems are designed

either for single-engine execution or for batch-oriented distributed analytics within a largely homogeneous execution environment. General-purpose federated query engines (e.g., Presto [3], BigDAWG [28]) support cross-source querying but do not account for spatial scope refinement or resolution-aware cost modeling. In contrast, HERMES targets interactive, federated query processing across polyglot storage systems, where raster, vector, and temporal data reside in heterogeneous backends with distinct query languages, indexing strategies, and resolution characteristics. Unlike PostGIS, HERMES does not assume that all data can or should be housed within a single engine; unlike Hadoop- and Spark-based systems such as SpatialHadoop and Sedona, it does not optimize primarily for throughput-oriented batch analytics. Instead, our work explicitly models spatiotemporal scope refinement, resolution alignment costs, and cross-engine data movement, and introduces adaptive predicate planning for low-latency, cross-system query execution over heterogeneous geospatial data.

Data storage systems have long relied on indexing as the central mechanism for accelerating retrieval and reducing the overhead of join operations. However, indexes are not without cost: each consumes disk space, and depending on the attribute indexed, some can grow nearly as large as the data itself. For spatial data, R-Trees [29], geohashes [30], and quadtiles [31] have provided effective strategies for partitioning and organizing data across large extents. These structures are most often used to support efficient dispersion and location of records rather than to evaluate complex predicates that span both space and time, such as queries sliding across temporal windows. HERMES takes indexing as one starting point but builds upon it further to enable cross-layer, spatiotemporal queries that go beyond simple dispersion and lookup.

Managing data and metadata [32] for large scientific collections remains a persistent challenge across web services, large-scale peer-to-peer grids [33]–[35], and collaborative environments [36], [37]. Prior systems for spatiotemporal data management, including Galileo [38], Trident [39], and Atlas [40], [41], move beyond storage to support ad hoc [42], geometry-constrained [43], and expressive querying [44], [45]. Complementary work has developed compact summaries or sketches of large datasets, including Granules [46], [47], Synopsis [48], Pebbles [49], and Gossamer [50]. HERMES is complementary to these efforts: rather than replacing such systems, it can interoperate with them as data sources while focusing on federated, scope-driven querying across heterogeneous backends.

Scientific data management systems have explored other avenues. SciDB [51], [52] specifically targets multidimensional arrays within a shared-nothing architecture to meet the needs of large-scale science. The Data Capacitor project [53], built on the Lustre file system [54] and wide-area networks, provides access to voluminous datasets spread across distributed infrastructures. However, such systems have tended to focus on raw access and storage rather than on real-time discovery, transformation, or cross-layer analytics. OpenFLAME [55] proposes a federated system for mapping and spatial search. Conclave [56] and Hu-Fu [57] implement federated query engines built around secure multi-party computation, with the latter refining spatial operations. HERMES introduces functionality assisting spatiotemporal queries. Through our scope abstraction, both space and time dimensions of these queries inform caching and data dispersion within compute clusters.

Ontology-based data access (OBDA) systems provide a conceptual layer over heterogeneous data sources [58], and have been applied to address data variety in distributed and heterogeneous environments [59], [60]. One of the necessary components of using ontologies for data exploration involves mapping concepts to feature space. However, maintaining and evolving mappings introduces significant overhead [61]. Our methodology does not preclude the use of ontology, while focusing on efficient cross-backend query execution.

Popivanov and Miller [62] study similarity search over large time-series datasets using wavelet-based summarization. Cormode et al. [63] extend this line of work by employing multiple wavelet variants to generate synopses or approximations of streaming data, and exploring their efficacy in summarizing voluminous datasets. Yousefi et al. [64] further show their utility for prediction. However, such synopsis-based methods are typically tailored to specific workloads, limiting support for arbitrary, cross-cutting queries.

Prior work on tiled rendering has primarily focused on visualization performance rather than analytical querying. Netek et al. [65] study latency and bandwidth trade-offs across raster and vector map formats, including WebP compression. Tile-based caching has likewise been used to improve interactive map responsiveness and reduce server load: Liu and Nie [66], Lewark et al. [67], and Wu et al. [68] demonstrate how caching can reduce server load while maintaining responsiveness. HERMES extends their utility: it combines tiling and caching with layered querying and predicate reordering.

V. CONCLUSIONS & FUTURE WORK

HERMES unifies storage-level harmonization, scope-based querying, and interactive rendering in a single pipeline.

RQ-1: Reconciling heterogeneous datasets requires harmonization at the level of storage rather than transformation at runtime. Our query model represents each predicate as producing a scope $S_i = (\Omega_i, \tau_i)$ and composes queries by intersecting (or unioning) these scopes, which directly constrains pushdown, data retrieval, and result materialization (tiles, geometries, or aggregates). By enforcing a consistent tiling scheme and CRS, raster datasets can be compared pixel-to-pixel while still accommodating multiple scales of resolution. Our use of the quadkey tiling scheme provides a globally consistent framework enabling datasets to interoperate across rasters, vectors, and shapefiles with predictable alignment guarantees. Precomputed overview levels support efficient zooming across scales.

RQ-2: Low-latency spatiotemporal queries across relational, document, and key-value stores are effectively supported by treating queries as compositions of predicates that each yield a spatiotemporal scope. By exploring both pipelined

and concurrent evaluation, we demonstrate that early pruning and parallelism can each reduce query cost under different conditions. Heuristic-based reordering that is grounded in selectivity estimates and index availability further ensures that high-impact predicates are executed first, while minimizing unnecessary data movement. This maintains semantic consistency across heterogeneous backends while reducing join and raster–vector overhead. We outperform Apache Sedona by a factor of up to **42x** (see subsection III-F).

RQ-3 Responsive exploration of layered spatiotemporal datasets requires both systems design and caching strategies. By dividing queries into tiles and distributing them across workers, HERMES ensures responsiveness even when multiple datasets must be overlaid. Server-side caching in a distributed Redis cluster allows us to minimize redundant datastore accesses and accelerates repeated predicate evaluation over common viewports. When coupled with client-side GPU acceleration and predictive prefetching, these strategies allow HERMES to sustain interactive, low-latency exploration of complex queries across multiple levels of aggregation. We outperform GeoPandas by a factor of up to **52x** (see III-G).

In future work, we plan to explore two complementary tracks. We will explore reinforcement learning for predicate selectivity and cost estimation. This will allow the framework to adapt autonomously to evolving data distributions and workloads. We will also test HERMES in domains such as public health, transportation, and disaster response to assess how harmonization, predicate reordering, and caching strategies scale across data with different structural/semantic characteristics.

VI. ACKNOWLEDGMENTS

This research was supported by the National Science Foundation (1931363, 2312319), the National Institute of Food Agriculture (2025-77039-45531, COL014021223), an NSF/NIFA Artificial Intelligence Institutes AI-LEAF Award [2023-03616], and Luce Professorship.

REFERENCES

[1] P. P. Khine and Z. Wang, "A review of polyglot persistence in the big data world," *Information*, vol. 10, no. 4, 2019.

[2] P. J. Sadalage and M. Fowler, *NoSQL distilled: a brief guide to the emerging world of polyglot persistence*. Pearson Education, 2013.

[3] R. Sethi, M. Traverso, D. Sundstrom, D. Phillips, W. Xie, Y. Sun, N. Yegitbasi, H. Jin, E. Hwang, N. Shingte *et al.*, "Presto: SQL on everything," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1802–1813.

[4] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, and T. Vassilakis, "Dremel: interactive analysis of web-scale datasets," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 330–339, 2010.

[5] L. Xu, R. L. Cole, and D. Ting, "Learning to optimize federated queries," in *Proceedings of the Second International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*, 2019, pp. 1–7.

[6] R. Obe and L. Hsu, *PostGIS in action*. Simon and Schuster, 2021.

[7] J. Yu, J. Wu, and M. Sarwat, "GeoSpark: A cluster computing framework for processing large-scale spatial data," in *Proceedings of the 23rd SIGSPATIAL international conference on advances in geographic information systems*, 2015, pp. 1–4.

[8] F. Baig, H. Vo, T. Kurc, J. Saltz, and F. Wang, "SparkGIS: Resource aware efficient in-memory spatial query processing," in *Proceedings of the 25th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2017, pp. 1–10.

[9] M. Muniswamaiah, T. Agerwala, and C. C. Tappert, "Federated query processing for big data in data science," in *2019 IEEE International Conference on Big Data (BigData)*. IEEE, 2019, pp. 6145–6147.

[10] GDAL/OGR contributors, *GDAL/OGR Geospatial Data Abstraction software Library*, Open Source Geospatial Foundation, 2025. [Online]. Available: https://gdal.org

[11] S. Gillies *et al.*, "Rasterio: geospatial raster i/o for Python programmers," Mapbox, 2013–. [Online]. Available: https://github.com/rasterio/rasterio

[12] J. Schwartz. (2024) Bing Maps Tile System. [Online]. Available: https://docs.microsoft.com/en-us/bingmaps/articles/bing-maps-tile-system

[13] C. Zhou, H. Lu, Y. Xiang, J. Wu, and F. Wang, "GeohashTile: Vector geographic data display method based on Geohash," *ISPRS International Journal of Geo-Information*, vol. 9, no. 7, 2020. [Online]. Available: https://www.mdpi.com/2220-9964/9/7/418

[14] J. T. Abatzoglou, "Development of gridded surface meteorological data for ecological applications and modelling," *International Journal of Climatology*, vol. 33, no. 1, pp. 121–131, 2013.

[15] P. O'Neill, S. Chan, E. Njoku, T. Jackson, R. Bindlish, J. Chaubell, and A. Colliander, "SMAP enhanced L3 radiometer global and polar grid daily 9 km EASE-Grid soil moisture, version 6," 2023.

[16] U.S. Geological Survey (USGS), "Annual NLCD collection 1 science products: U.S. Geological Survey data release," 2024.

[17] United States Department of Agriculture (USDA) National Agricultural Statistics Service (NASS), "Cropland Data Layer," 2024.

[18] OpenJS Foundation, "deck.gl: GPU-powered, highly performant large-scale data visualization," 2025.

[19] C. Carrión, "Kubernetes scheduling: Taxonomy, ongoing issues and challenges," *ACM Computing Surveys*, vol. 55, no. 7, pp. 1–37, 2022.

[20] G. Ginesu, M. Pintus, and D. D. Giusto, "Objective assessment of the WebP image coding algorithm," *Signal Processing: Image Communication*, vol. 27, no. 8, pp. 867–874, 2012.

[21] P. O'Neill, S. Chan, E. Njoku, T. Jackson, R. Bindlish, and J. Chaubell, "SMAP L3 radiometer global daily 36 km EASE-Grid soil moisture, version 9," 2023.

[22] Soil Survey Staff, Natural Resources Conservation Service, United States Department of Agriculture, "Soil Survey Geographic (SSURGO) Database," 2024, accessed 4/18/2024.

[23] B. R. Pickard, J. Daniel, M. Mehaffey, L. E. Jackson, and A. Neale, "EnviroAtlas: A new geospatial tool to foster ecosystem services science and resource management," *Ecosystem Services*, vol. 14, pp. 45–55, 2015.

[24] K. Jordahl, J. Van den Bossche, J. Wasserman, J. McBride, M. Fleischmann, J. Gerard, J. Tratner, M. Perry, C. Farmer, G. A. Hjelle *et al.*, "geopandas/geopandas: v0.7.0," *Zenodo*, 2021.

[25] A. Eldawy, "SpatialHadoop: Towards flexible and scalable spatial processing using MapReduce," in *Proceedings of the 2014 SIGMOD PhD symposium*, 2014, pp. 46–50.

[26] R. T. Whitman, M. B. Park, S. M. Ambrose, and E. G. Hoel, "Spatial indexing and analytics on Hadoop," in *Proceedings of the 22nd ACM SIGSPATIAL international conference on advances in geographic information systems*, 2014, pp. 73–82.

[27] J. Yu, Z. Zhang, and M. Sarwat, "Spatial data management in apache spark: the GeoSpark perspective and beyond," *GeoInformatica*, vol. 23, no. 1, pp. 37–78, 2019.

[28] J. Duggan, A. J. Elmore, M. Stonebraker, M. Balazinska, B. Howe, J. Kepner, S. Madden, D. Maier, T. Mattson, and S. Zdonik, "The BigDAWG polystore system," *ACM Sigmod Record*, vol. 44, no. 2, pp. 11–16, 2015.

[29] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.

[30] G. Niemeyer, "Geohash," 2008, https://en.wikipedia.org/wiki/Geohash.

[31] OpenStreetMap, "QuadTiles: Geodata storage and indexing," 2022. [Online]. Available: https://wiki.openstreetmap.org/wiki/QuadTiles

[32] S. L. Pallickara, S. Pallickara, M. Zupanski, and S. Sullivan, "Efficient metadata generation to enable interactive data discovery over large-scale scientific data collections," in *2010 IEEE Second International Conference on Cloud Computing Technology and Science*. IEEE, 2010, pp. 573–580.

[33] G. Fox, S. Pallickara, M. Pierce, and H. Gadgil, "Building messaging substrates for web and grid applications," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 363, no. 1833, pp. 1757–1773, 2005.

[34] G. Fox, H. Bulut, K. Kim, S.-H. Ko, S. Lee, S. Oh, S. Pallickara, X. Qiu, A. Uyar, M. Wang *et al.*, "Collaborative web services and peer-to-peer grids," *SIMULATION SERIES*, vol. 35, no. 1, pp. 3–12, 2003.

[35] Y. L. Simmhan, S. L. Pallickara, N. N. Vijayakumar, and B. Plale, "Data management in dynamic environment-driven computational science," in *Grid-Based Problem Solving Environments: IFIP TC2/WG 2.5 Working Conference on Grid-Based Problem Solving Environments: Implications for Development and Deployment of Numerical Software July 17–21, 2006, Prescott, Arizona, USA*. Springer, 2007, pp. 317–333.

[36] S. Lee, S. H. Ko, and G. C. Fox, "Adapting content for mobile devices in heterogeneous collaboration environments." in *International Conference on Wireless Networks*, 2003, pp. 211–217.

[37] G. C. Fox, S. H. Ko, K.-S. Kim, S. Oh, and S. Lee, "Integration of hand-held devices into collaborative environments." in *International Conference on Internet Computing*, 2002, pp. 231–250.

[38] M. Malensek, S. L. Pallickara, and S. Pallickara, "Galileo: A framework for distributed storage of high-throughput data streams," in *2011 Fourth IEEE International Conference on Utility and Cloud Computing*. IEEE, 2011, pp. 17–24.

[39] M. Malensek, W. Budgaga, R. Stern, S. Pallickara, and S. L. Pallickara, "Trident: Distributed storage, analysis, and exploration of multidimensional phenomena," *IEEE Transactions on Big Data*, vol. 5, no. 2, pp. 252–265, 2018.

[40] D. Rammer, S. Lee Pallickara, and S. Pallickara, "Atlas: A distributed file system for spatiotemporal data," in *Proceedings of the 12th IEEE/ACM International Conference on Utility and Cloud Computing*, 2019, pp. 11–20.

[41] D. Rammer, T. Buddhika, M. Malensek, S. Pallickara, and S. L. Pallickara, "Enabling fast exploratory analyses over voluminous spatiotemporal data using analytical engines," *IEEE Transactions on Big Data*, vol. 8, no. 1, pp. 213–228, 2019.

[42] M. Malensek, S. Pallickara, and S. Pallickara, "Fast, ad hoc query evaluations over multidimensional geospatial datasets," *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 28–42, 2015.

[43] ——, "Evaluating geospatial geometry and proximity queries using distributed hash tables," *Computing in Science & Engineering*, vol. 16, no. 4, pp. 53–61, 2014.

[44] ——, "Analytic queries over geospatial time-series data using distributed hash tables," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 6, pp. 1408–1422, 2016.

[45] M. Malensek, S. L. Pallickara, and S. Pallickara, "Expressive query support for multidimensional data in distributed hash tables," in *2012 IEEE Fifth International Conference on Utility and Cloud Computing*. IEEE, 2012, pp. 31–38.

[46] S. Pallickara, J. Ekanayake, and G. Fox, "Granules: A lightweight, streaming runtime for cloud computing with support, for map-reduce," in *2009 IEEE International Conference on Cluster Computing and Workshops*. IEEE, 2009, pp. 1–10.

[47] ——, "An overview of the granules runtime for cloud computing," in *2008 IEEE Fourth International Conference on eScience*. IEEE, 2008, pp. 412–413.

[48] T. Buddhika, M. Malensek, S. L. Pallickara, and S. Pallickara, "Synopsis: A distributed sketch over voluminous spatiotemporal observational streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 11, pp. 2552–2566, 2017.

[49] T. Buddhika, S. L. Pallickara, and S. Pallickara, "Pebbles: Leveraging sketches for processing voluminous, high velocity data streams," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 8, pp. 2005–2020, 2021.

[50] T. Buddhika, M. Malensek, S. Pallickara, and S. L. Pallickara, "Living on the edge: Data transmission, storage, and analytics in continuous sensing environments," *ACM Transactions on Internet of Things*, vol. 2, no. 3, pp. 1–31, 2021.

[51] P. Cudré-Mauroux, H. Kimura, K.-T. Lim, J. Rogers, R. Simakov, E. Soroush, P. Velikhov, D. L. Wang, M. Balazinska, J. Becla *et al.*, "A demonstration of SciDB: a science-oriented DBMS," *Proceedings of the VLDB*, vol. 2, no. 2, pp. 1534–1537, 2009.

[52] M. Stonebraker, P. Brown, A. Poliakov, and S. Raman, "The architecture of SciDB," in *International Conference on Scientific and Statistical Database Management*. Springer, 2011, pp. 1–16.

[53] S. C. Simms, G. G. Pike, and D. Balog, "Wide area filesystem performance using Lustre on the TeraGrid," Tech. Rep., 2007.

[54] W. Yu, R. Noronha, S. Liang, and D. K. Panda, "Benefits of high speed interconnects to cluster file systems: a case study with Lustre," in *IEEE International Parallel & Distributed Processing Symposium*, 2006.

[55] S. Bharadwaj, A. Rowe, and S. Seshan, "Uniting the world by dividing it: Federated maps to enable spatial applications," in *Proceedings of the 2025 Workshop on Hot Topics in Operating Systems*, ser. HotOS '25. New York, NY, USA: Association for Computing Machinery, 2025, p. 74–79.

[56] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *Proceedings of the Fourteenth EuroSys Conference 2019*, ser. EuroSys '19. New York, NY, USA: ACM, 2019.

[57] Y. Tong, Y. Zeng, Y. Song, X. Pan, Z. Fan, C. Xue, Z. Zhou, X. Zhang, L. Chen, Y. Xu, K. Xu, and W. Lv, "Hu-Fu: efficient and secure spatial queries over data federation," *The VLDB Journal*, vol. 34, no. 2, p. 19, Jan. 2025.

[58] M. Gagnon, "Ontology-based integration of data sources," in *2007 10th International Conference on Information Fusion*. IEEE, 2007, pp. 1–8.

[59] I. Horrocks, M. Giese, E. Kharlamov, and A. Waaler, "Using semantic technology to tame the data variety challenge," *IEEE Internet Computing*, vol. 20, no. 6, pp. 62–66, 2016.

[60] D. Calvanese, I. Horrocks, E. Jiménez-Ruiz, E. Kharlamov, M. Meier, M. Rodriguez-Muro, and D. Zheleznyakov, "On rewriting and answering queries in OBDA systems for big data," ser. CEUR Workshop Proceedings, vol. 1080. Aachen: RWTH, 2013.

[61] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo, "The MASTRO system for ontology-based data access," *Semantic Web*, vol. 2, pp. 43–53, 01 2011.

[62] I. Popivanov and R. J. Miller, "Similarity search over time-series data using wavelets," in *Proceedings 18th international conference on data engineering*. IEEE, 2002, pp. 212–221.

[63] G. Cormode, M. Garofalakis, P. J. Haas, C. Jermaine *et al.*, "Synopses for massive data: Samples, histograms, wavelets, sketches," *Foundations and Trends® in Databases*, vol. 4, no. 1–3, pp. 1–294, 2011.

[64] S. Yousefi, I. Weinreich, and D. Reinarz, "Wavelet-based prediction of oil prices," *Chaos, Solitons & Fractals*, vol. 25, no. 2, pp. 265–275, 2005.

[65] R. Netek, J. Masopust, F. Pavlicek, and V. Pechanec, "Performance testing on vector vs. raster map tiles—comparative study on load metrics," *ISPRS International Journal of Geo-Information*, vol. 9, no. 2, 2020.

[66] H. Liu and Y. Nie, "Tile-based map service GeoWebCache middleware," in *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, vol. 1, 2010, pp. 692–697.

[67] E. Lewark, M. Young, P. Khandelwal, S. L. Pallickara, and S. Pallickara, "Periscope: A framework for visualizations of multiresolution spatiotemporal data at scale," in *2024 IEEE International Conference on Big Data (BigData)*, 2024, pp. 1373–1380.

[68] H. Wu, X. Guan, T. Liu, L. You, and Z. Li, *A High-Concurrency Web Map Tile Service Built with Open-Source Software*. Boston, MA: Springer US, 2013, pp. 183–195.