# HiFIVE: High-Fidelity Vector-Tile Reduction for Map Exploration*

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

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

## Abstract

Interactive visualization is a common tool for exploring large open-data repositories, where users examine datasets across diverse domains. For large-scale spatial data, many existing tools rely on server-side rendering to produce small image tiles that can be viewed on the client side. However, exploratory analysis often requires client-side rendering, where users can interactively restyle the same data without regenerating tiles on the server. This paper presents HiFIVE, a data-management framework for scalable, high-fidelity client-side geospatial visualization. We formalize the visualization-aware tile reduction problem, which captures the trade-off between tile size and visualization distortion, and prove its NP-hardness. HiFIVE introduces a two-stage solution that combines triage and sparsification to selectively prune records, attributes, and values using information-theoretic and spatial criteria. Experiments demonstrate substantial tile-size reductions while preserving visual fidelity and interactive performance at terabyte scale.

## CCS Concepts

• **Information systems** → **Geographic information systems**; *Data compression*; • **Human-centered computing** → *Geographic visualization.*

## Keywords

Vector Tiles, Visualization Fidelity, Interactive Map Exploration

**ACM Reference Format:**
Tarlan Bahadori and Ahmed Eldawy. 2026. HiFIVE: High-Fidelity Vector-Tile Reduction for Map Exploration. In *The 34th ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '26), November 03–06, 2026, Riverside, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3841645.3842946

## 1 Introduction

Interactive visualization is one of the quickest, and sometimes the only, means to determine what a dataset actually contains and assess its fitness to answer certain scientific questions. With hundreds of thousands of publicly available datasets [30, 32, 33], users must be able to scan, compare, and eliminate candidates quickly in order to select the right data for the task at hand. Moreover, multiple reports estimate that roughly 60% of publicly available data carries a geospatial component, highlighting the central role of maps as a primary interface for exploration across a wide range of domains, including urban planning [7, 18], public health [5], environmental monitoring [21], and transportation [35].

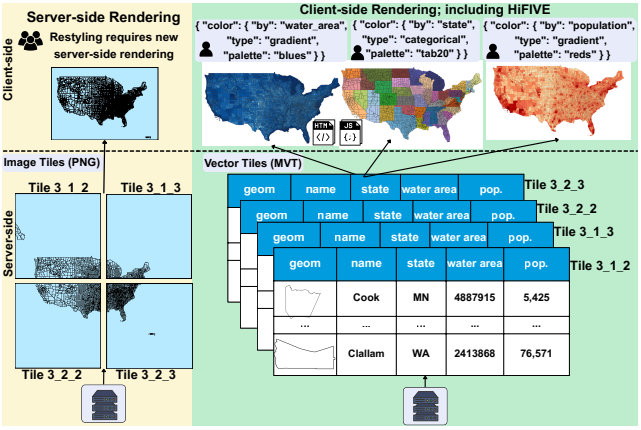



**Figure 1: Server-side rendering vs. client-side rendering. In client-side rendering, the same MVT tiles produced on the server can be reused for different styled visualizations.**

Scaling geospatial exploration to today's data volumes remains difficult. Although web maps use multi-resolution tile pyramids for interactive pan-and-zoom [2, 12, 15], existing systems still face a core trade-off between scalability and visualization fidelity. Existing approaches can be largely classified as *server-side rendering* or *client-side rendering*. In *server-side rendering*, the server generates and styles *raster tiles*, which scales well because computation happens on the backend and tile size is bounded by pixel resolution [15, 20, 36]. However, the visualization is effectively fixed, and the client cannot restyle or recover details that were not encoded in the pixels. In *client-side rendering*, the server delivers *vector tiles* [3, 18, 27, 31], i.e., geometries and attributes, and the client performs styling, enabling high-fidelity, fully customizable maps. Yet vector tiles are not naturally size-bounded. Payload can grow arbitrarily with feature density, geometry complexity, and attribute cardinality, often exceeding practical network and browser rendering budgets. Some work has been done to use dictionary-style encoding in Mapbox Vector Tiles (MVT) [13] or column compression in Maplibre Tiles (MLT) [31], but they do not address the unbounded tile size problem. Some applications use carefully handcrafted rules that are tailored for a specific dataset to reduce vector tile size, e.g., base maps in Google Maps, but they do not generalize to other datasets.

Figure 1 contrasts server-side raster rendering and client-side vector-tile rendering. In server-side rendering, the server produces

*This work is supported in part by the National Science Foundation (NSF) under grants IIS-2046236 and TI-2520163.


*SIGSPATIAL '26, Riverside, CA, USA*

ACM ISBN 979-8-4007-2950-8/2026/11
https://doi.org/10.1145/3841645.3842946

styled image tiles. This bounds tile size, but restyling requires server-side regeneration or precomputation of additional tile sets, which limits interactive exploration across many styles and users. In contrast, client-side rendering sends vector tiles that contain geometries and attributes, allowing each user to apply a different style locally, such as coloring counties by water area, state, or population. The challenge with this approach is that vector tiles can grow arbitrarily large with millions of records and dozens of attributes per tile, making them very slow to transfer and render.

This paper presents HiFIVE, a high-fidelity vector tile reduction system for map exploration that addresses two fundamental bottlenecks in interactive vector-tile visualization, *scalability* and *visualization fidelity*. As data size grows, the resulting vector tiles can be massive, and, unlike raster tiles, vector tiles are unbounded, a challenge that is particularly imminent when dealing with tens of thousands of datasets. Aggressively shrinking tiles is only valuable if the resulting maps remain visually and semantically faithful. HiFIVE reframes tile generation as a data-management optimization problem. The key idea is that not all records and not all attributes are equally important to produce a high-fidelity visualization. For example, a spatially large polygon is more prominent than a small one in the visualization. Additionally, some attributes, e.g., categorical attributes, tend to take less space and are more useful in styling compared to, say, a generated UUID. HiFIVE exploits these asymmetries by deciding which records and attributes to retain at each tile such that tiles remain within a certain size while supporting rich client-side styling. Since the decision is made independently per tile, tiles become more detailed as the user zooms in and the tile size decreases, eventually showing full details.

This paper introduces and formalizes the *visualization-aware tile reduction* problem. It builds on concepts from information theory to measure the information loss in reduced tiles and combines them with spatial data metrics to capture visualization loss. To build on that, we propose a two-stage solution, *sparsification* and *triage*. The sparsification step reduces the tile size by setting some values to null while minimizing information and visual loss. The triage step applies a set of heuristics to make quick and significant reductions to the tile, making it ready for the fine-grained sparsification step. Together, these techniques produce compact, standard-compliant vector tiles that support interactive, user-defined styling entirely on the client, enabling exploration over terabyte-scale datasets without incurring per-style recomputation on the server. We validate our approach through extensive experiments, demonstrating substantial reductions in tile size while preserving visual fidelity and maintaining interactive performance at scale.

Below, we summarize the contributions of this paper:

- We propose the tile distortion metric that quantifies visual information loss in map tiles (§4).
- We formalize the visualization-aware tile reduction problem and prove its NP-hardness (§4).
- We introduce HiFIVE with a two-stage solution to the problem based on integer linear programming (§5,6).
- We carry out an extensive experimental evaluation to study the scalability of HiFIVE (§7).

## 2 Related Work

Geospatial visualization systems face dual challenges of managing exponentially growing datasets while maintaining interactive performance. In recent years, two dominant approaches have emerged to tackle these challenges:

First, **cluster-based rendering** systems such as GeoSparkViz [36], HadoopViz [12], and AID* [15] perform **server-side rendering** using parallel frameworks like Spark or Hadoop MapReduce. Liu et al. [23] instead use adaptive tile drawing on a single machine to precompute and serve vector data as WMTS image tiles. While effective for high-resolution static maps, these approaches limit client-side interactivity because tiles are partially or fully pregenerated. Moreover, specialized methods are needed for different visualizations, such as kernel density maps [5] and heat maps [22]. Although scalable to large datasets, the server can quickly become overloaded when users request different styling options.

The second approach is **client-side rendering** which focuses on delivering the raw data to the client which can style and render it based on user needs without incurring any additional overhead on the server. Some tools first aggregate the data at the server side and deliver a summarized version that can be visualized efficiently but it still requires the server to be aware of the visualization type and style [20]. Another line of work selects or samples representative spatial objects for map visualization [18, 26]. These methods improve visual clarity for point-based exploration, but they do not address general vector-tile reduction across geometry types or preserve arbitrary attributes for later client-side styling. Other approaches build vector tiles, e.g., MVT [13] or MLT [31], which contain all records with all their attributes. Unlike raster tiles, vector tiles are not size-bounded and can grow arbitrarily large, posing an additional overhead on transferring and rendering these tiles. Planetiler [24] proposes a parallel algorithm for generating map tiles but it uses application-specific rules that are specific to OpenStreetMap data. Similarly, [27] introduces the *thinning* problem to cap the number of records per tile in Google Maps while maintaining geographic coherence across tiles but it does not consider the importance of individual attributes. Tippecanoe [1] provides a general purpose MVT tile generator that employs a greedy, density-based feature dropping technique to ensure map legibility across zoom levels. However, its heuristic-driven simplification disregards the semantic utility of non-spatial attributes, leading to visual fidelity loss when styling is applied to data.

This paper adopts the client-side rendering model and treats vector-tile reduction as an optimization problem rather than a heuristic post-processing step. Given a target tile-size budget, HiFIVE decides which records, attributes, and cell values to retain in each tile so as to reduce storage cost while minimizing visual and semantic information loss. The result is a compact set of vector tiles that remains compatible with standard client-side styling and supports interactive exploration at a tunable level of detail.

## 3 Overview

This work enables scalable, high-fidelity geospatial visualization via a two-stage data processing pipeline that produces compact, stylable vector tiles as shown in Figure 2. Our goal is to support

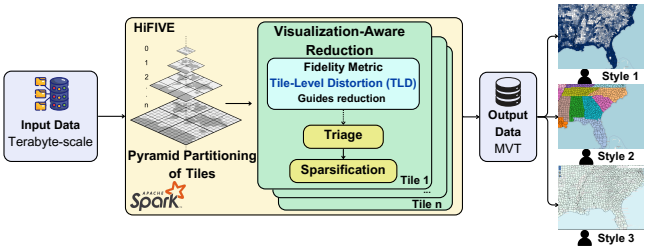



**Figure 2: HiFIVE distributed Spark pipeline for producing MVTs through tile construction, triage, and sparsification.**

terabyte-scale datasets while enforcing a per-tile size budget that keeps network transfer and client-side rendering interactive.

*Usage scenario and front-end interface.* HiFIVE supports the interactive web-map exploration session in Figure 2. To interact with the data, an analyst can pan, zoom, and restyle a large dataset across a sequence of actions, not a single request. The dataset publisher prepares the data through an offline process that creates a tile pyramid while setting the per-tile size budget $B$ (default 256 KB) and pyramid depth $z_{\max}$. At runtime, a standard web-map client, e.g., OpenLayers [25] or MapLibre [31], requests tiles from HiFIVE's HTTP endpoint `GET /tiles/{z}/{x}/{y}.mvt` for the tiles intersecting the current viewport, and the server returns existing pre-built MVT [13] tiles or generates them on-the-fly. The client provides map styles as a JSON object that contains styling attributes color/size, line width, feature draw order, etc. (e.g., `{"color_by":"population","palette":"reds"}`). HiFIVE 's reduced MVTs serve as an interactive visualization layer for initial dataset discovery; they are not used as the substrate for analytical queries. After an analyst identifies a dataset of interest, feature inspection, filtering, and spatial queries are evaluated against the indexed full-resolution backend, where all original records and attributes remain available. Users can use any standard JavaScript map library or GIS tools that supports MVT tiles so they do not need to change their existing web applications to use HiFIVE backend and they can customize the style to match their needs. Since all styling happens on the same set of tiles, applications can benefit from caching both at the server and client, which greatly improves the performance.

The pipeline in Figure 2 begins with a *data preparation stage* that (i) reprojects input geometries to the web Mercator projection used in web maps and (ii) partitions the data into a multi-zoom tile pyramid structure compatible with standard web visualization libraries [17, 25, 31]. This stage follows well-established practice and we refer the readers to [12, 15] for full details. Unlike raster tiles [12, 15, 36], vector tiles store individual geometries and their attributes. This results in tile sizes that are highly variable and can exceed browser and network budgets in dense regions. Therefore, this paper focuses on visualization-aware tile reduction: given a tile and a target byte budget $B$, the goal is to reduce the tile toward that budget while minimizing loss in visualization fidelity. Further details on the problem definition are given in section 4.

To solve the visualization-aware tile reduction problem, we run two algorithms individually on each tile, *triage* and *sparsification*,

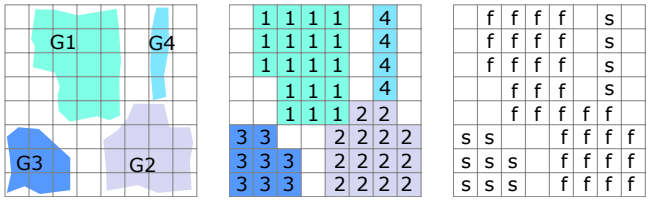
**Table 1: Running example: $N$=4 lakes, $d$=3 attributes. Salinity is (f)resh or (s)alt. Output $T_{\text{out}}$ nullifies the `name` of the two smallest lakes and additionally Dune's salinity.**

**(a) Input $T_{\text{in}}$**

| Geom. | name | salin. |
|---|---|---|
| G1 | Azul | f |
| G2 | Birch | f |
| G3 | Cobalt | s |
| G4 | Dune | s |

**(b) Output $T_{\text{out}}$**

| Geom. | name | salin. |
|---|---|---|
| G1 | Azul | f |
| G2 | Birch | f |
| G3 | ⊥ | s |
| G4 | ⊥ | ⊥ |




**(a) Input tile $T_{in}$** **(b) Rasterized tile** **(c) Rasterized salinity**

**Figure 3: Rasterization of $T_{\text{in}}$ at $R$=64 pixels. (a) Input geometries; (b) their pixel footprints; (c) the `salinity` attribute image; empty cells denote ⊥. Formal definitions of the rendering function and the attribute image are in Appendix A.**

where the triage algorithm makes big cuts to bring the tile size down while the sparsification algorithm further reduces the tile using a size model that approximates the serialized MVT size.

This sparsification algorithm formulates the problem as an integer linear program (ILP) that nulls selected attribute values to reduce tile size while preserving visualization fidelity. Its constraints and objective balance (i) visual prominence by pixel footprint, (ii) storage size, and (iii) information loss via Jensen-Shannon Divergence (JSD).

Due to the limitations of ILP solvers, they cannot work on extremely large tiles. Thus, we introduce a lightweight *visualization-aware triage* algorithm which uses various empirical techniques to reduce the tile size before running the more costly sparsification step. The triage step involves removing selected rows and columns and quantizing numeric columns. The order of operations is determined by *attribute saliency*, measured via the drop in entropy caused by reduction. This ensures that triage targets attributes least likely to be used for styling the output visualizations later on.

Both algorithms run in parallel using a distributed Spark job which allows horizontal scaling with data size. They output standard MVT tiles [13] that can be rendered using any existing client-side visualization library such as OpenLayers [25] or MapLibre [31]. Since our reduction algorithms do not alter the tile schema or format, existing styling tools can be used without modification.

## 4 Visualization-Aware Tile Reduction

This section formally defines the visualization-aware tile reduction problem. The input to this problem is a tile $T$, which is represented as a relation with $N$ rows and $d$ columns. Each row represents a geometric feature, e.g., a lake, where the first attribute is the

geometry and the additional attributes can be of any type. Table 1 shows an example of an input relation with four features and their geometries are displayed in Figure 3a. Tiles are serialized in column-oriented or dictionary-based formats (MLT [31], MVT [13]), so the serialized size is the sum of per-column encoded lengths. That is, the size of the tile is $\text{Size}(T) = \sum_{j=1}^{d} \text{Size}_j(T)$, where $\text{Size}_j$ is a function that measures the size of column $j$ in bytes. A tile size can be reduced by setting some attribute values to null ($\perp$). For example, the output tile $T_{\text{out}}$ in Table 1 is smaller than the input tile $T_{\text{in}}$. Our problem is to find a reduced tile with minimum visual distortion. Appendix A gives the formal definitions of the rendering function and pixel-weighted distributions; here we summarize only the quantities needed for the optimization.

*Tile-Level Distortion (TLD).* TLD measures visual distortion between two tiles. To capture visual saliency, we compare tiles through their rasterized footprints at the rendering resolution. Figure 3 illustrates this for the running example at $R = 64$ pixels: larger geometries, such as $G_1$ with 18 pixels, receive more weight than smaller ones, such as $G_4$ with four pixels.

Using this rasterization, we compute a pixel-weighted empirical distribution $\tilde{p}_j^T$ for each attribute $j$ by counting how many pixels are covered by each attribute value. For two tiles $T_1$ and $T_2$, let $m_j = \frac{1}{2}(\tilde{p}_j^{T_1} + \tilde{p}_j^{T_2})$. We define the visualization-aware divergence of attribute $j$ as the Jensen–Shannon divergence between the two pixel-weighted distributions:

$$\text{VAD}_j(T_1, T_2) = \tfrac{1}{2}\text{KLD}(\tilde{p}_j^{T_1} \| m_j) + \tfrac{1}{2}\text{KLD}(\tilde{p}_j^{T_2} \| m_j), \tag{1}$$

where $\text{KLD}(p\|q) = \sum_{x \in \text{Dom}_j} p(x) \log_2 \frac{p(x)}{q(x)}$. With base-2 logarithms, $\text{VAD}_j(T_1, T_2) \in [0, 1]$.

We then aggregate attribute-level divergences into tile-level distortion (TLD). To account for the relative importance of attributes in styling, we use inverse-entropy weights ($w_j \propto H_j^{-1}$), since low-entropy attributes, such as categorical fields, are often useful for styling.

$$\text{TLD}(T_1, T_2) = \sum_{j=2}^{d} w_j(T_1)\, \text{VAD}_j(T_1, T_2). \tag{2}$$

TLD therefore penalizes changes that (a) affect geometrically prominent features or (b) distort attributes likely to drive styling.

*Problem statement.* Given an input tile $T_{\text{in}}$ with schema $\boldsymbol{S}$ and a size budget $B$, the *visualization-aware tile reduction* problem is

$$\begin{aligned} \min_{T_{\text{out}} \in \text{Rel}(\boldsymbol{S})} \quad & \text{TLD}(T_{\text{in}}, T_{\text{out}}) \\ \text{s.t.} \quad & \text{Size}(T_{\text{out}}) \le B, \qquad |T_{\text{out}}| = |T_{\text{in}}|, \end{aligned} \tag{3}$$

where $T_{\text{out}}$ may differ from $T_{\text{in}}$ only by replacing non-null cell values with $\perp$. The $T_{\text{out}}$ in Table 1 is a feasible solution: nullifying `name` for Cobalt and both non-geometry cells of Dune shrinks the `name` dictionary from four entries to two, achieving $\text{TLD} \approx 0.031$ because the nullified features are the two smallest lakes ($G_3$: 8 px, $G_4$: 4 px).

Theorem 4.1. *The visualization-aware tile reduction problem of Equation 3 is NP-hard.*

The reduction is from 0-1 Knapsack where a feature $i$ encodes a Knapsack item whose value is its pixel count and whose size is the byte cost of retaining its attribute. The full proof is in subsection A.5.

**Table 2: Spearman correlation among priority metrics.**

| Metric | Vertices | Pixels | Attributes |
|---|---|---|---|
| **Provinces** | | | |
| Pixels | 0.743 | – | – |
| Attributes | 0.195 | 0.089 | – |
| Byte Size | 0.979 | 0.731 | 0.226 |
| **Roads** | | | |
| Pixels | 0.901 | – | – |
| Attributes | 0.061 | 0.070 | – |
| Byte Size | 0.879 | 0.811 | 0.460 |

## 5 Visualization-Aware Triage

HiFIVE first applies a lightweight visualization-aware triage stage to quickly reduce large tiles before fine-grained optimization. This is particularly important for low-zoom tiles, which may contain a large fraction of the input dataset and therefore produce prohibitively large optimization problems. Triage uses inexpensive heuristics to reduce the number of records, attributes, and distinct values while limiting the resulting loss in visualization fidelity. The heuristics are grouped into two approaches: record-level triage, which makes horizontal cuts by retaining high-priority records, and column-level triage, which reduces storage by lowering attribute cardinality and shrinking encoded dictionaries. The resulting tile provides a smaller, more manageable input for the downstream sparsification stage.

### 5.1 Record-Level Triage

Record-level triage reduces tile size by keeping only a subset of features before the more expensive sparsification step. A simple approach is fixed-size random reservoir sampling; however, random sampling does not account for visual importance, and existing visualization-aware sampling methods are not applicable for non-point geometries or styling based on non-geometry attributes [26]. We therefore use a prioritized reservoir that favors records expected to contribute more to the rendered map.

The reservoir design has two components: its capacity and its priority metric. Capacity can be defined by the number of records, total byte size, number of vertices, or number of cells. Since the goal of triage is to bound the cost of the downstream ILP, we use the number of cells as the default capacity, as it directly controls the number of ILP variables. The priority metric determines which records are admitted first when the capacity is exceeded. We consider record byte size, vertex count, attribute tuple size, pixel footprint, and random order as a baseline. These choices are orthogonal: for example, a reservoir may use byte-size capacity while admitting records in decreasing pixel footprint.

To compare candidate priority metrics, we computed Spearman correlations on representative point, line, and polygon datasets: eBird, Roads , and Counties. As shown in Table 2, vertex count, pixel footprint, and byte size are strongly correlated, since geometrically complex features often occupy more screen space and require more bytes to encode. We therefore use vertex count as the default priority because it is a simple and inexpensive proxy for both visual prominence and storage cost.

Record-level triage is inexpensive and highly parallel, making it effective for quickly reducing dense tiles. However, because it

removes entire records, it can discard visually or semantically important features. For this reason, HiFIVE uses record-level triage only as a coarse filtering step before fine-grained sparsification.

## 5.2 Column-Level Triage

The second set of heuristics work at the column level and aim to reduce the number of columns and the amount of information stored in them. This step is performed before sparsification, if the tile exceeds size constraint after record-level triage.

*5.2.1 Numeric Quantization.* For numeric attributes, we reduce cardinality by grouping values into a small number of bins, 10 by default. This is effective for visualization because numeric attributes are often used for graduated color scales, where small value changes typically cause only minor visual differences. Quantization also improves dictionary-based encoding, such as MVT, by reducing the number of distinct values.

*5.2.2 String Trimming.* For text attributes, we reduce size by trimming long strings to a fixed length. This is useful for label-like attributes, where a shortened value can remain recognizable; for example, "Massachusetts" can be shortened to "Mass. . . ". We estimate the information loss using JSD between the original and trimmed value distributions. If trimming merges distinct labels, e.g. "Santa Cruz" and "Santa Clara" into the same prefix, the divergence increases and the attribute becomes less favorable for reduction.

*5.2.3 Prioritized Column Reduction.* Numeric quantization and string trimming are applied selectively based on their estimated information loss. For each candidate attribute, we compare the original and reduced value distributions using JSD, and prioritize reductions with the smallest divergence first. After each reduction, the tile size is recomputed, and the process continues until the tile satisfies the size threshold or no candidate reductions remain.

# 6 ILP-based Sparsification

After triage, HiFIVE applies fine-grained sparsification to tiles that require further reduction toward the target size budget (B). Given a post-triage tile containing $N$ records and $d$ attributes, the goal is to reduce its estimated serialized size while preserving visual salience and attribute utility. We simplify and reformulate this problem as a 0–1 integer linear program (ILP) that jointly decides (i) which records to retain and (ii) which attribute values (cells) to retain for the kept records. By operating on the reduced tile produced by triage, the ILP can make more precise reduction decisions while keeping the optimization problem tractable.

*Decision variables.* Without loss of generality, we assume that the first column contains the geometry attribute. We introduce the following binary variables:

$$y_i \in \{0,1\} \quad \text{if geometry of record } i \in [1,N] \text{ is kept} \tag{4}$$

$$u_j \in \{0,1\} \quad \text{if non-geometric attribute } j \in [2,d] \text{ is kept} \tag{5}$$

$$x_{i,j} \in \{0,1\} \quad \text{if attribute cell } (i \in [1,N], j \in [2,d]) \text{ is kept} \tag{6}$$

Variables $x_{i,j}$ are instantiated only for non-null cells and $u_j$ are instantiated if the column is not empty. Since all decision variables are binary and both the objective and constraints are linear, the resulting formulation is a 0–1 ILP

**Table 3: Example ILP variables for input $T_{in}$ (table 1a).**

| | **name ($u_2$)** | **salinity ($u_3$)** |
|---|---|---|
| $y_1$ | $x_{1,2}$ | $x_{1,3}$ |
| $y_2$ | $x_{2,2}$ | $x_{2,3}$ |
| $y_3$ | $x_{3,2}$ | $x_{3,3}$ |
| $y_4$ | $x_{4,2}$ | $x_{4,3}$ |

**Table 4: Example ILP solution; results in $T_{\text{out}}$ (table 1b).**

| | **1** | **1** |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 0 | 0 |

*Structural constraints.* A cell can be kept only if both its geometry and its column are kept:

$$x_{i,j} \le y_i \qquad \forall i \in [1,N], j \in [2,d], \tag{7}$$

$$x_{i,j} \le u_j \qquad \forall i \in [1,N], j \in [2,d]. \tag{8}$$

*Linear size model.* To approximate tile size while keeping the model linear, we use two dominant components: geometry bytes and attribute-cell bytes, where attribute-cell cost includes an amortized share of the column dictionary. Let $\widehat{b}_i^{geom}$ denote the estimated bytes contributed by the geometry of record $i$, derived from its vertex count. In vector-tile encodings, attribute values are not stored independently; instead, each layer maintains a shared dictionary of distinct values. Consequently, the storage cost of an attribute column $j$ consists of a dictionary cost $\widehat{b}_j^{dict}$, representing the bytes required for unique non-null values, and a per-cell pointer cost $b^{ptr}$ used to reference the dictionary.

To maintain a linear model, we distribute the dictionary cost across the $n_j$ non-null cells in the column. Thus, when a cell $(i,j)$ is retained, it contributes an estimated cost of $b^{ptr} + \widehat{b}_j^{dict}/n_j$. While removing a single cell only reduces the dictionary size if a unique value is entirely eliminated, this approximation remains highly effective. Columns with high cardinality naturally result in higher amortized costs, making them primary candidates for reduction under a fixed size budget. Conversely, columns with few distinct values contribute minimally to the total size, ensuring that the approximation has negligible impact on the overall constraint. The resulting linear size constraint is:

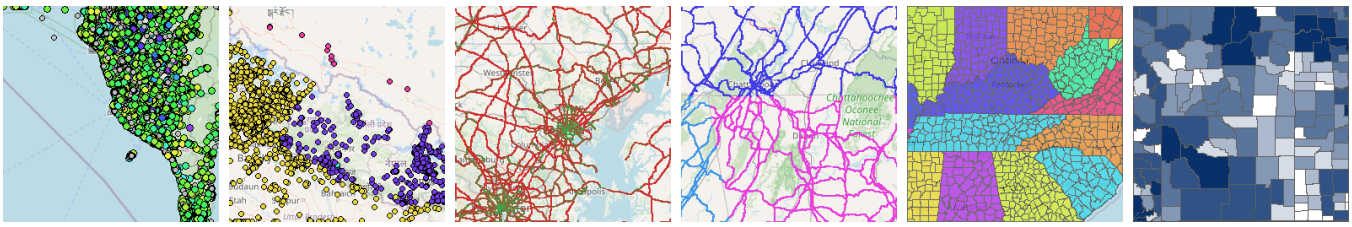
$$\sum_{i=1}^{N} \widehat{b}_i^{geom}\, y_i + \sum_{(i,j)} \left( b^{ptr} + \frac{\widehat{b}_j^{dict}}{n_j} \right) x_{i,j} \;\le\; B, \tag{9}$$

where $B$ is the target tile size. Since MVT encoding uses discrete dictionaries, this constraint is an approximation of the final serialized size; Appendix B evaluates its empirical adherence.

*Objective function.* The ILP maximizes a weighted utility that balances visual salience of records and fidelity of retained attributes:

$$\max \;\; \alpha \sum_{i=1}^{N} U_i^{rec}\, y_i + (1-\alpha) \sum_{(i,j)} U_{i,j}^{cell}\, x_{i,j}. \tag{10}$$

where $U_i^{\text{rec}}$ represents the utility of record $i$ based on its visual salience, $U_{i,j}^{\text{cell}}$ represents the cell utility which measures the information retention, and $\alpha \in [0,1]$ is a parameter that balances the importance of these two parts. Both utility functions are further described below.



(a) eBird (Grad) (b) eBird (Cat) (c) Roads (Grad) (d) Roads (Cat) (e) Counties (Cat) (f) Counties (Grad)

**Figure 4: Examples of visualization stylings used in our experiments: for *eBird*, we use observation time as a gradient and country code as categorical; for *Roads*, length as a gradient and state as categorical; and for *Counties*, water area as a gradient and state as categorical. The same MVTs are reused across different stylings.**

*Record utility.* Visual salience is quantified using the accurate pixel footprint $pc_i$ of record $i$ in the rendered tile. We normalize it by the maximum pixel footprint in the tile,

$$\widetilde{pc}_i = \frac{pc_i}{\max_j pc_j}, \tag{11}$$

and define the record utility as

$$U_i^{rec} = \lambda_{rec}\, \widetilde{pc}_i^{\,p}, \tag{12}$$

where $\lambda_{rec} > 0$ controls the strength of the salience preference and $p \geq 1$ controls its nonlinearity. The exponent $p$ amplifies differences in normalized pixel footprint, increasing the contrast between visually large and small records. This encourages the ILP solver to retain large geometries, which would otherwise be disproportionately attractive to drop due to their higher byte cost when satisfying the size constraint.

*Cell utility.* For each attribute cell $(i, j)$, we measure the importance of retaining its value using a normalized divergence score $\widetilde{D}_{i,j} \in [0, 1]$ built from the same visualization-aware divergence (VAD, Equation 1) that defines TLD in Equation 2. Specifically, $D_{i,j}$ is the per-attribute Jensen-Shannon divergence between the current column distribution and the distribution that would result from nullifying cell $(i, j)$ alone:

$$D_{i,j} \;=\; \mathrm{VAD}_j\Big(T,\, T^{(i,j\leftarrow\perp)}\Big), \tag{13}$$

where $T^{(i,j\leftarrow\perp)}$ denotes the table $T$ with the value of cell $(i, j)$ replaced by $\perp$ (null) and all other cells unchanged, and $\mathrm{VAD}_j$ is the Jensen–Shannon divergence of the pixel-weighted column-$j$ distributions of its two arguments (Equation 1).

$$\widetilde{D}_{i,j} \;=\; \frac{D_{i,j}}{D_j^{\max}}, \qquad D_j^{\max} \;=\; \max_{r\in\{1,\ldots,N\}} D_{r,j}. \tag{14}$$

If $D_j^{max} = 0$, we set $\widetilde{D}_{i,j} = 0$ for all cells in column $j$. We define a *keep-bonus* $\mathrm{KB}_{i,j} = 1 - \widetilde{D}_{i,j}$ and assign the cell utility

$$U_{i,j}^{cell} = \mathrm{KB}_{i,j}. \tag{15}$$

This formulation cleanly separates responsibilities where record utilities prioritize visually salient geometries, while cell utilities preserve the most informative attribute values within the remaining size budget. The ILP therefore removes small or visually insignificant records first and spends the remaining budget on attribute values that minimize distributional distortion.

Once the problem is formulated, we pass it to an ILP solver to assign values to the decision variables. To create the output tile $T_{\text{out}}$, we make a copy of the input tile $T_{\text{in}}$ and set each cell $f_i[j]$ to null if the corresponding variable $x_{i,j} = 0$. If $y_i = 0$, all cells of record $i$ are set to $\perp$; if $u_j = 0$, all cells in attribute $j$ are set to $\perp$. Empty rows or columns can then be omitted during serialization.

*Solver cost.* The ILP exposes $N + d + |(i, j) : f_i[j] \neq \perp|$ binary variables and $O(N, d)$ structural constraints, so the problem size grows linearly in both the number of records $N$ and the number of non-geometric attributes $d$. The remaining quantities affect only *preprocessing*, not the ILP itself:

- **Attribute domain cardinality** $|\mathrm{Dom}_j|$**.** Enters the column-wise JSD (Eq. 13) and the dictionary-cost estimate $\widehat{b}_j^{dict}$. Both are computed once per column in $O(N + |\mathrm{Dom}_j|)$ time and produce a single scalar per cell; not changing the ILP size.
- **Vertex count** $|v_i|$ **(geometric "number of points").** Enters $\widehat{b}_i^{geom}$ and the pixel-footprint computation $pc_i$. Rasterization is $O(|v_i| + pc_i)$ per record, $O(\sum_i |v_i| + \sum_i pc_i)$ for the whole tile, and is independent of $d$.
- **Geometry type.** Polygons, lines, and points all flatten to the same record-level utility $U_i^{rec}$ once $pc_i$ is computed; the ILP is agnostic to type.

Algorithm 1 outlines the ILP formulation for sparsifying a vector tile.

Lines 0–4 compute the visual salience of each record. Line 2 measures the pixel footprint $pc_i$ of each geometry under the rendering function $\mathcal{R}$ (Eq. A.4) in Appendix A. These values are normalized and converted into record utilities $U_i^{rec}$ according to Eq. 12, favoring geometries that occupy a larger visible area.

Lines 4–8 compute statistics required by the linear size model. For each attribute column $j$, Line 6 estimates the dictionary size $\widehat{b}_j^{dict}$ and the number of non-null cells $n_j$. The dictionary size corresponds to the bytes needed to encode the distinct non-null values of column $j$ in the shared vector-tile dictionary. Line 8 estimates the geometry cost $\widehat{b}_i^{geom}$ of each record using its vertex count. Lines 9–11 assign utilities to individual attribute cells. For each non-null cell $(i, j)$, Line 10 computes the normalized divergence $\widetilde{D}_{i,j}$ using the column-wise JSD defined in Eqs. 13–14. Line 11 converts this score into a keep-bonus $\mathrm{KB}_{i,j} = 1 - \widetilde{D}_{i,j}$. As a result, cells whose removal causes *minimal* distributional change receive *higher* utility and are more likely to be retained. Lines 12–17 formulate and solve the ILP.

**Algorithm 1** CellSparsifyILP($T_{\text{in}}, B$)

**Require:** Input tile $T_{\text{in}} = \{f_i\}_{i=1}^{N}$ with $d$ attributes (column 1 is geometry)
**Require:** Size threshold $B$
**Ensure:** Reduced tile $T_{\text{out}}$
**Compute record salience**
1: **for** $i \leftarrow 1$ to $N$ **do**
2: Compute pixel footprint $pc_i$ of record $i$
3: **for** $i \leftarrow 1$ to $N$ **do**
4: Compute record utility $U_i^{rec}$ from $pc_i$
**Compute column statistics for size estimation**
5: **for** $j \leftarrow 2$ to $d$ **do**
6: Compute dictionary bytes $\widehat{b}_j^{dict}$ and non-null count $n_j$
7: **for** $i \leftarrow 1$ to $N$ **do**
8: Estimate geometry bytes $\widehat{b}_i^{geom}$
**Compute cell utilities**
9: **for** each non-null cell $(i, j)$ **do**
10: Compute normalized divergence $\widetilde{D}_{i,j}$
11: $U_{i,j}^{cell} \leftarrow 1 - \widetilde{D}_{i,j}$
**ILP formulation**
12: Create ILP model $\mathcal{M}$
13: Add decision variables $y_i$, $u_j$, and $x_{i,j}$
14: Add objective function (Eq. 10)
15: Add structural constraints linking $x_{i,j}$ to $y_i$ and $u_j$
16: Add linear size constraint (Eq. 9)
17: Solve $\mathcal{M}$
18: Construct $T_{\text{out}}$ from $(y, u, x)$
19: **return** $T_{\text{out}}$

**Table 5: Main datasets used in our experiments. "Size" shows raw GeoJSON file size.**

| Dataset | Size | #Records | #Vertices | Geometry |
|---|---|---|---|---|
| eBird [29] | 935.3 GB | 801M | 801M | Points |
| OSM_Buildings [11] | 187 GB | 455M | 2.4B | Polygons |
| OSM_Roads [11] | 31 GB | 70M | 789M | Linestrings |

The objective function balances record and cell utilities using $\alpha$, while the linear size constraint keeps the estimated tile size within the target budget $B$. Finally, Line 18 constructs the output tile by nullifying records with $y_i = 0$, nullifying columns with $u_j = 0$, and setting individual cells to null when $x_{i,j} = 0$.

## 7 Experiments

In this section, we run a comprehensive evaluation on HiFIVE to provide experimental evidence of its scalability. We also evaluate the resulting stylized vector tiles using visual-quality metrics.

### 7.1 Experimental Setup

We ran experiments on a twelve-node Spark cluster. The master has two 8-core Intel Xeon E5-2609 v4 CPUs (1.70 GHz) and 128 GB RAM, while each worker node has two 6-core Intel Xeon E5-2603 v4 CPUs (1.70 GHz) and 64 GB RAM. All machines run CentOS 7.5.1804, using local SSDs for the OS and HDDs for data storage. The source code is publicly available on GitHub.[1]

[1]https://github.com/tarlaun/hifive

**Table 6: Experimental settings.**

| Setting | Values (default) |
|---|---|
| Tile size constraint ($B$) | 32 KB – 4 MB (256 KB) |
| Triage priority | Highest number of vertices |
| Tile capacity | 50-100k (100K cells) |
| Zoom levels | 1 – 15 (8) |
| $\alpha$ | 0.0 – 1.0 (0.5) |
| Input sample size | 0.001% – 100% (100%) |

Table 5 lists the datasets used in experiments. The datasets include points, lines, and polygons with up to 935GB. For large-scale datasets such as eBird, OSM Roads, and OSM Buildings, we additionally generate multiple subsets using uniform random sampling at different ratios (e.g., 10%, 1%, 0.1%, and smaller). These subsets enable scalability experiments and allow us to study the impact of tile-size constraints on both performance and visual fidelity across a wide range of data volumes, while preserving the spatial and attribute characteristics of the original datasets. In addition, we use smaller regional datasets, including Counties [4] and NA_Roads [9], for controlled quality and ablation experiments.

**Baselines:** We consider two baselines: AID* [15], a Spark-based visualization index that combines pre-generated tiles with a data index to support scalable visualization of raster tiles. Tippecanoe [1] is the tool used by Mapbox [13] to create reduced-size vector tiles.

**Parameters:** Table 6 summarizes the experimental parameters that we vary and their corresponding default values.

### 7.2 Quality Metrics

To evaluate visual fidelity, we render each tile (baseline and reduced) into an RGB image of size $H \times W$ (in our experiments $H = W = 256$) using the same client-side styling. Figure 4 summarizes the styles we use in the evaluation. Let $A, B \in \{0, \ldots, 255\}^{H \times W \times 3}$ denote the resulting baseline and reduced images, and let $A_{u,v,c}$ be the intensity of channel $c \in \{1, 2, 3\}$ at pixel $(u, v)$.

*RMSE.* We compute the per-pixel mean squared error (MSE) over all pixels and channels,

$$\text{MSE}(A, B) = \frac{1}{3HW} \sum_{u=1}^{H} \sum_{v=1}^{W} \sum_{c=1}^{3} \left(A_{u,v,c} - B_{u,v,c}\right)^2, \tag{16}$$

and report the root mean squared error (RMSE),

$$\text{RMSE}(A, B) = \sqrt{\text{MSE}(A, B)}. \tag{17}$$

Here $c$=1, 2, 3 denote the red, green, and blue channels of the rendered sRGB image.

*Structural Similarity Index (SSIM).* The SSIM index [34] is a perceptual metric designed to align with human visual judgment by comparing local patterns of luminance, contrast, and structure rather than global pixel-wise errors. The metric operates by sliding a fixed-size window across the reference image $A$ and compared image $B$, extracting local patches $x$ and $y$ at each location. The similarity between these patches is defined as:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{18}$$

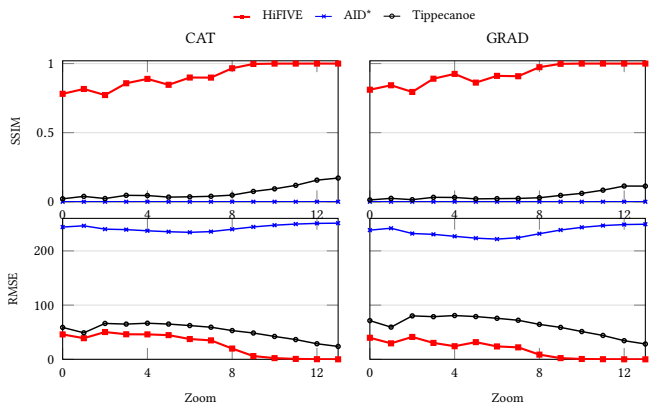



**Figure 5: Per-zoom mean SSIM and RMSE for the categorical (CAT) and gradient (GRAD) render styles on the same set of MVTs generated by HiFIVE for OSM_Roads dataset, comparing HiFIVE against Tippecanoe and AID* at a 256 KB per-tile budget.**

**Table 7: Spearman rank correlation between TLD and visualization quality metrics across datasets. Higher TLD indicates larger deviation; thus correlations are positive for RMSE and negative for SSIM.**

| Metric | Counties | NE_Roads | ebird 0.01 |
|---|---|---|---|
| Spearman(TLD, RMSE) | 0.999823 | 0.813129 | 0.882601 |
| Spearman(TLD, SSIM) | -0.999867 | -0.797708 | -0.856090 |

where $\mu$ and $\sigma$ denote the local mean and variance, $\sigma_{xy}$ is the covariance, and $C_1, C_2$ are stabilizing constants.

The final score is the mean SSIM aggregated over all window locations and color channels. In our evaluation, we use $7 \times 7$ windows on an intensity range of $[0, 255]$. SSIM values range from $[-1, 1]$, with higher values indicating superior structural fidelity. We also computed peak signal-to-noise ratio (PSNR), but omit it due to space; its trends were consistent with RMSE and SSIM.

## 7.3 Quality of Tiles

This section evaluates the visual fidelity of the generated tiles. Since the quality metrics require a rendered image, we define six styles, shown in Figure 4, as two for each of the three datasets, eBird, roads, and counties. Each dataset has one gradient style and one categorical style.

Figure 5 compares HiFIVE with Tippecanoe and AID* across zoom levels on OSM_Roads under the same per-tile budget as we vary the zoom level of the tile from 0 to 13. HiFIVE consistently achieves higher SSIM and lower RMSE, especially at mid-level zooms where tile density most often stresses the byte budget. AID* is included as a raster baseline but does not support client-side restyling. The other two datasets show similar results but we omit them due to space.

To validate the proposed TLD metric in Equation 2, Table 7 reports the Spearman correlation between TLD and both RMSE and SSIM across three datasets. Higher TLD consistently corresponds to higher RMSE and lower SSIM, indicating that TLD captures distortions visible in styled renderings while remaining computable directly from the MVT/data table, without requiring image rendering.

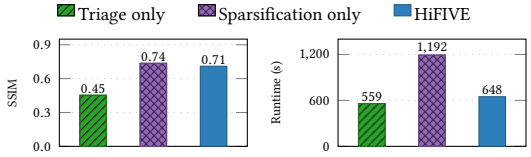



**Figure 6: Ablation study on the Buildings dataset at a 256 KB budget. Triage+sparsification nearly matches sparsification-only fidelity while reducing runtime by 1.84×.**

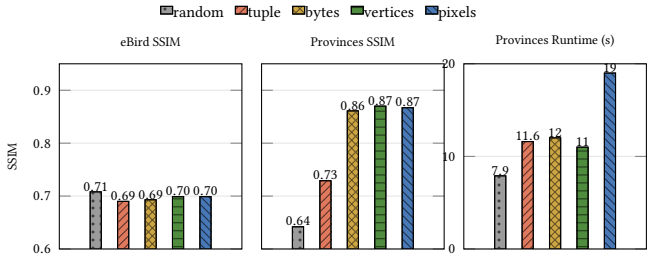



**Figure 7: Runtime and mean SSIM against the unreduced baseline at a 256 KB per-tile budget. Triage priorities have little effect on point data, while geometry-aware priorities substantially improve fidelity on polygons.**

## 7.4 Visualization-Aware Triage

This section verifies that the triage step is effective in quickly reducing the tile size without significant distortion to the visualization. To test that, we test the effect of disabling the triage step (whole or in parts) on the performance and quality of tile reduction.

Figure 6 shows the effect of the entire triage step on quality and performance on the building dataset. First, we notice that triage only has the lowest fidelity (SSIM 0.453), since record-level filtering can remove visually important features under a tight budget. While sparsification only can achieve the highest fidelity (SSIM 0.736), it requires the longest runtime (1,192 s). The full triage+sparsification pipeline provides the best trade-off, reaching SSIM 0.710 in 648 s, a 1.84× speedup over sparsification alone. This experiment verifies that triage can cheaply reduce dense tiles before the ILP solver performs finer-grained sparsification.

Figure 7 breaks down the effect of the record-level triage on two datasets, eBird and Provinces. We try five priorities for the selection process and show their effect on quality and running time. For the eBird point dataset, we see no significant difference between the selection priorities since they are all equal for point data. However, for the polygon dataset, we clearly see that the geometry-aware priorities are clearly better with pixel coverage and vertex count achieving the highest SSIM and substantially outperform random sampling. When comparing their performance

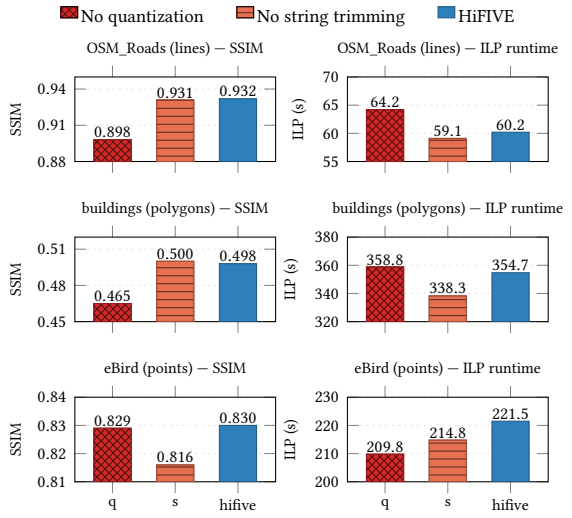



**Figure 8: Triage ablation at $B$=32 KB. Each bar reports SSIM when one triage step is disabled while sparsification runs at the same budget: −q removes numeric quantization, −s removes string trimming.**

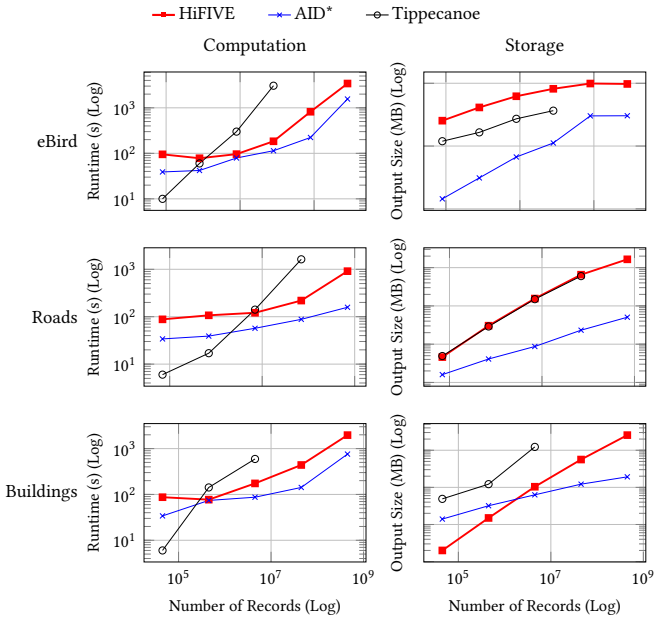



**Figure 9: Scaling comparison across eBird subsets (8k to 801M records), roads subsets (7k to 70M records), and buildings subsets (45k to 455M records). Both axes are logarithmic.**

on the third subfigure, we notice that the vertex count is the fastest among them due to its simplicity, therefore, we use it as the default record-level triage technique.

We observe that vertex count does not necessarily represent the most visually prominent feature. For example, a local winding road might have more vertices than a long highway. However, we found in most real datasets that number of vertices is strongly correlated with the feature size in bytes and its pixel footprint, explaining the similar results in terms of quality. HiFIVE still supports all these priorities if this is not the case.

Figure 8 reports the effect of the column-level triage techniques. In this experiment, we turn off each technique separately, i.e., no quantization and no string trimming, and compare them to the full triage step (reported as HiFIVE). For each case, we measure both the quality and the performance of tile reduction.

In general, we observe that the importance of each of the two column-level triage techniques varies by dataset. However, the full triage step consistently provides the best quality at very good running time. For OSM_Roads and Buildings, numeric quantization has the largest effect: removing it reduces SSIM by 0.034 on roads and 0.033 on buildings, while disabling string trimming has little effect. For eBird, which contains many string attributes, string trimming is the most important step, reducing SSIM by 0.013 when removed, while quantization is neutral. These results show that the triage steps are complementary: different reductions matter for different geometry types and attribute distributions, motivating their combined use in HiFIVE.

## 7.5 End-to-end Evaluation

This section compares HiFIVE with Tippecanoe, a single-node vector-tile baseline, and AID*, a cluster-based raster baseline, on tile-generation runtime, storage footprint, and serving latency.

Figure 9 reports runtime and storage footprint for eBird, OSM_Roads, and OSM_Buildings as input size scales from 0.001% to 100%. For each setting, we generate an eight-level tile pyramid ( 20,000 tiles in total) and report total output size. Spark-based methods, HiFIVE and AID*, scale to the largest inputs, while single-node Tippecanoe fails on larger datasets. HiFIVE processes the largest eBird input, nearly one terabyte, in under an hour and produces reusable vector tiles for client-side styling. AID* produces smaller raster output because it stores pixels rather than geometries and attributes, but does not support restyling. Output size grows with dataset scale as more tiles become populated.

Figure 10 reports per-tile serving latency for a cold viewport trace over the 1.9 GB Buildings dataset. We compare server-side rendering, where each requested style requires server-side raster tile generation, with HiFIVE's client-side rendering pipeline, where the same reduced MVTs can be reused across users and styled locally. As the number of users increases from $N = 1$ to $N = 50$, the server-side curves shift to higher latencies because each user may request a different styled output. In contrast, HiFIVE remains below the 1 s interactivity budget across all tiles, since expensive tiles are sparsified and pre-generated and shared as MVTs while only light tiles are generated on demand.

## 7.6 Sparsification Experiments

All ILP instances are solved with Gurobi Optimizer 11.0.2 [19] using a 1% relative optimality gap. To evaluate sparsification cost, we generate eight zoom levels and record each invocation of Algorithm 1. Figure 11 reports mean runtime as the number of cells ($N \times d$) varies from 50K to 100K, covering tiles that exceed the target size $B$ but

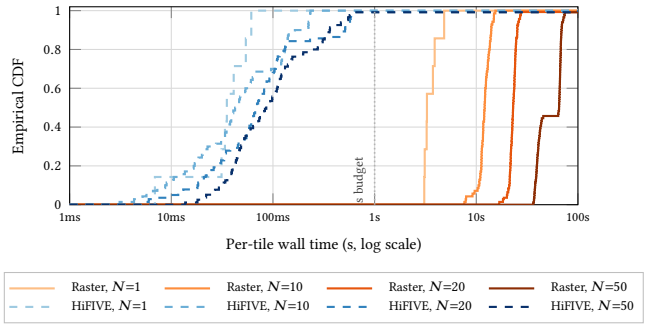



**Figure 10: Per-tile serving-time CDF for Buildings viewport trace. HiFIVE keeps all requests below the 1 s budget using reusable MVTs, while the raster baseline regenerates image tiles for each requested style.**

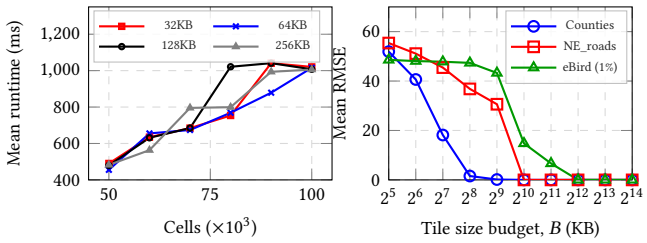



**Figure 11: Left: Runtime vs. number of input cells for ILP solver, for different output size constraints. Right: Mean RMSE after reduction using different size constraints.**

remain within the 100K-cell triage cap. Runtime grows roughly linearly with the number of cells, which determines the number of ILP variables. Thus, reducing either records or attributes lowers the cell count and solver cost; while the target size $B$ has little effect because it does not change the problem size.

The right side of Figure 11 reports the mean error on tiles for three datasets containing points, lines, and polygons. We vary the tile size budget ($B$) from 32 KB to 4 MB and measure its effect on RMSE. As expected, RMSE drops rapidly as the budget $B$ increases since the reduction is triggered for fewer tiles.

We set $\alpha = 0.5$ by default to balance visual prominence and attribute information in Equation 10. A sensitivity analysis in subsection B.1 shows that intermediate values of $\alpha$ consistently outperform the two extremes, indicating that the objective benefits from combining both terms.

## 7.7 Qualitative Evaluation

Figure 12 shows representative tiles rendered with three methods. The first row shows the Counties dataset with categorical styling by the STATE attribute. Under the same target tile size, Tippecanoe drops more complete records because its reduction operates primarily at the feature level. In contrast, HiFIVE can retain more

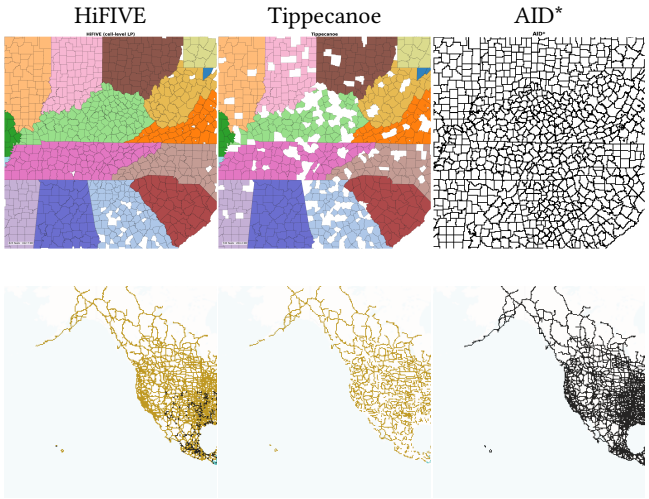



**Figure 12: Qualitative comparison on representative Counties and Roads tiles. Each row shows the same styled tile generated by HiFIVE, Tippecanoe, and AID*.**

geometries by selectively nullifying attribute cells with lower contribution to the objective, preserving a denser and more visually complete map. The second row shows the Roads dataset with categorical styling by the `continent` attribute. In this case, HiFIVE preserves more road geometries but removes the styling attribute for some smaller records, which renders them in the default black color. Tippecanoe preserves attributes for retained records but drops more records entirely, resulting in a sparser output. AID* generates raster tiles that do not support client-side restyling.

## 8 Conclusion and Future Work

This paper introduced HiFIVE, a visualization-aware framework for reducing vector-tile sizes while preserving the information needed for interactive map exploration. We formalize this task as the *visualization-aware tile reduction* problem: given a tile-size budget, select what information to retain while minimizing visualization distortion. To quantify this distortion, we introduce Tile-Level Distortion (TLD), an information-theoretic metric that combines pixel-weighted attribute distributions to capture both visual prominence and semantic information loss. HiFIVE addresses the resulting optimization problem through a two-stage pipeline that combines fast triage with fine-grained sparsification based on geometry complexity, encoding cost, and information loss. The resulting tiles remain compact, standards-compliant, and compatible with client-side styling. Experiments show that HiFIVE scales to large geospatial datasets while maintaining high visual fidelity and interactive serving performance.

A key direction for future work is cross-tile coordination. HiFIVE currently optimizes each tile independently, which enables scalable parallel processing but may lead to inconsistent attribute retention for features clipped across neighboring tiles. Coordinating decisions across adjacent tiles would improve consistency for features spanning tile boundaries. Another possible extension is adapting the cell-level reduction decisions to newer vector-tile encodings, such as MapLibre Tiles (MLT) [31].

## References


[1] [n. d.]. Tippecanoe GitHub Repository. https://github.com/mapbox/tippecanoe/blob/master/README.md. Accessed: 2025-02-15.

[2] [n. d.]. UCR STAR. https://star.cs.ucr.edu. Accessed: 2025-02-15.

[3] Tarlan Bahadori, Sai Sreekar Sarvepalli, and Ahmed Eldawy. 2025. LASEK: LLM-Assisted Style Exploration Kit for Geospatial Data. *Proc. VLDB Endow.* 18, 12 (Aug. 2025), 5435–5438. https://doi.org/10.14778/3750601.3750690

[4] US Census Bureau. 2019. County. https://doi.org/10.6086/N12F7KGT Retrieved from UCR-STAR https://star.cs.ucr.edu/?TIGER2018/COUNTY&d.

[5] Tsz Nam Chan, Pak Lon Ip, Kaiyan Zhao, Leong Hou U, Byron Choi, and Jianliang Xu. 2022. LIBKDV: A Versatile Kernel Density Visualization Library for Geospatial Analytics. *Proc. VLDB Endow.* 15, 12 (2022), 3606–3609. https://doi.org/10.14778/3554821.3554855

[6] Liming Dong, Qiushi Bai, Taewoo Kim, Taiji Chen, Weidong Liu, and Chen Li. 2020. Marviq: Quality-Aware Geospatial Visualization of Range-Selection Queries Using Materialization. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 67–82.

[7] Harish Doraiswamy, Eleni Tzirita Zacharatou, Fabio Miranda, Marcos Lage, Anastasia Ailamaki, Cláudio T. Silva, and Juliana Freire. 2018. Interactive Visual Exploration of Spatio-Temporal Urban Data Sets using Urbane. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, 1693–1696. https://doi.org/10.1145/3183713.3193559

[8] Carsten F. Dormann, Jana M. McPherson, Miguel B. Araújo, Roger Bivand, Janine Bolliger, Gudrun Carl, Richard G. Davies, Alexandre Hirzel, Walter Jetz, W. Daniel Kissling, et al. 2007. Methods to Account for Spatial Autocorrelation in the Analysis of Species Distributional Data: A Review. *Ecography* 30, 5 (2007), 609–628. https://doi.org/10.1111/j.2007.0906-7590.05171.x

[9] Natural Earth. 2019. Main roads in North America. https://doi.org/10.6086/N1JM27N3 Retrieved from UCR-STAR https://star.cs.ucr.edu/?NE/roads_NA&d.

[10] Ahmed Eldawy, Vagelis Hristidis, Saheli Ghosh, Majid Saeedan, Akil Sevim, A.B. Siddique, Samriddhi Singla, Ganesh Sivaram, Tin Vu, and Yaming Zhang. 2021. Beast: Scalable Exploratory Analytics on Spatio-temporal Data. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management (CIKM '21)*. Association for Computing Machinery, New York, NY, USA, 3796–3807. https://doi.org/10.1145/3459637.3481897

[11] Ahmed Eldawy and Mohamed F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 1352–1363. https://doi.org/10.1109/ICDE.2015.7113382

[12] Ahmed Eldawy, Mohamed F. Mokbel, and Christopher Jonathan. 2016. HadoopViz: A MapReduce Framework for Extensible Visualization of Big Spatial Data. (2016), 601–612. https://doi.org/10.1109/ICDE.2016.7498284

[13] GDAL Documentation Team. 2024. *MVT: Mapbox Vector Tiles — GDAL Documentation*. GDAL Project. https://gdal.org/drivers/vector/mvt.html.

[14] Kareem El Gebaly, Guoyao Feng, Lukasz Golab, Flip Korn, and Divesh Srivastava. 2018. Explanation Tables. *Bulletin of the IEEE Technical Committee on Data Engineering* 41, 4 (2018), 43–55.

[15] Saheli Ghosh and Ahmed Eldawy. 2022. AID*: A Spatial Index for Visual Exploration of Geo-Spatial Data. *IEEE Transactions on Knowledge and Data Engineering* 34, 8 (2022), 3569–3582. https://doi.org/10.1109/TKDE.2020.3026657

[16] Saheli Ghosh, Tin Vu, Mehrad Amin Eskandari, and Ahmed Eldawy. 2019. UCR-STAR: the UCR spatio-temporal active repository. *SIGSPATIAL Special* 11, 2 (Dec. 2019), 34–40. https://doi.org/10.1145/3377000.3377005

[17] Google. 2026. Google Maps. https://maps.google.com.

[18] Tao Guo, Kaiyu Feng, Gao Cong, and Zhifeng Bao. 2018. Efficient Selection of Geospatial Data on Maps for Interactive and Visualized Exploration. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. ACM, 567–582. https://doi.org/10.1145/3183713.3183738

[19] Gurobi Optimization, LLC. 2026. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[20] Jianfeng Jia, Chen Li, Xi Zhang, Chen Li, Michael J. Carey, and Simon Su. 2016. Towards interactive analytics and visualization on one billion tweets. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2016, Burlingame, California, USA, October 31 - November 3, 2016*. ACM, 85:1–85:4. https://doi.org/10.1145/2996913.2996923

[21] Yunfan Kang, Ziang Zhao, Amr Magdy, Win Cowger, and Andrew B. Gray. 2019. Scalable Multi-resolution Spatial Visualization for Anthropogenic Litter Data. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2019, Chicago, IL, USA, November 5-8, 2019*. ACM, 560–563. https://doi.org/10.1145/3347146.3359074

[22] Lauro Didier Lins, James T. Klosowski, and Carlos Eduardo Scheidegger. 2013. Nanocubes for Real-Time Exploration of Spatiotemporal Datasets. *IEEE Trans. Vis. Comput. Graph.* 19, 12 (2013), 2456–2465. https://doi.org/10.1109/TVCG.2013.179

[23] Zebang Liu, Luo Chen, Mengyu Ma, Anran Yang, Zhinong Zhong, and Ning Jing. 2024. An Efficient Visual Exploration Approach of Geospatial Vector Big Data on the Web Map. *Information Systems* 121 (2024), 102333. https://doi.org/10.1016/j.is.2023.102333

[24] OnthegoMap. 2024. Planetiler: A fast tool for building planet-scale vector tiles. https://github.com/onthegomap/planetiler.

[25] OpenLayers Contributors. 2025. OpenLayers - Web Mapping Library. https://openlayers.org/

[26] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. 2016. Visualization-Aware Sampling for Very Large Databases. (2016), 755–766. https://doi.org/10.1109/ICDE.2016.7498289

[27] Anish Das Sarma, Hongrae Lee, Hector Gonzalez, Jayant Madhavan, and Alon Y. Halevy. 2013. Consistent thinning of large geographical data for map visualization. *ACM Trans. Database Syst.* 38, 4 (2013), 22. https://doi.org/10.1145/2539032.2539034

[28] Bettina Speckmann and Kevin Verbeek. 2010. Necklace Maps. *IEEE Transactions on Visualization and Computer Graphics* 16, 6 (2010), 881–889. https://doi.org/10.1109/TVCG.2010.197

[29] Brian L Sullivan, Christopher L Wood, Marshall J Iliff, Rick E Bonney, Daniel Fink, and Steve Kelling. 2009. eBird: A citizen-based bird observation network in the biological sciences. *Biological conservation* 142, 10 (2009), 2282–2292.

[30] The World Bank. 2026. World Bank Open Data. https://data.worldbank.org/. Accessed: 2026-02-28.

[31] Markus Tremmel and Roland Zink. 2025. MapLibre Tile: A Next Generation Vector Tile Format. In *Proceedings of the 33rd ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL '25)*. ACM, Minneapolis, MN, USA. https://doi.org/10.1145/3748636.3763208

[32] United Nations Statistics Division. 2026. UNdata: United Nations Data Service. https://data.un.org/. Accessed: 2026-02-28.

[33] U.S. General Services Administration. 2026. Data.gov: The Home of the U.S. Government's Open Data. https://data.gov/. Accessed: 2026-02-28.

[34] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: from error visibility to structural similarity. *IEEE Trans. Image Process.* 13, 4 (2004), 600–612. https://doi.org/10.1109/TIP.2003.819861

[35] Shi Yin, Moyin Li, Nebiyou Tilahun, Angus G. Forbes, and Andrew E. Johnson. 2015. Understanding Transportation Accessibility of Metropolitan Chicago Through Interactive Visualization. In *Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics, UrbanGIS@SIGSPATIAL 2015, Bellevue, WA, USA, November 3-6, 2015*. ACM, 77–84. https://doi.org/10.1145/2835022.2835036

[36] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. 2018. GeoSparkViz: A Scalable Geospatial Data Visualization Framework in the Apache Spark Ecosystem. (2018), 12:1–12:12. https://doi.org/10.1145/3221269.3223040

[37] Zhiguang Zhou, Fengling Zheng, Jin Wen, Yuanyuan Chen, Xinyu Li, Yuhua Liu, Yigang Wang, and Wei Chen. 2023. A User-Driven Sampling Model for Large-Scale Geographical Point Data Visualization via Convolutional Neural Networks. *IEEE Transactions on Human-Machine Systems* 53, 5 (2023), 885–896. https://doi.org/10.1109/THMS.2023.3296692

**Table 8: Pixel-weighted counts $c_j^T(v)$ and Laplace-smoothed distributions $\tilde{p}_j^T(v)$ for the running example ($R$=64, $\varepsilon$=1). Uncovered pixels are assigned to $\perp$. Bold values change between $T_{\text{in}}$ and $T_{\text{out}}$.**

| Attribute $j$ | Value $v$ | $c_j^{T_{\text{in}}}$ | $c_j^{T_{\text{out}}}$ | $\tilde{p}_j^{T_{\text{in}}}$ | $\tilde{p}_j^{T_{\text{out}}}$ |
|---|---|---|---|---|---|
| name | Azul | 18 | 18 | 0.275 | 0.275 |
| | Birch | 14 | 14 | 0.217 | 0.217 |
| | Cobalt | 8 | **0** | 0.130 | **0.014** |
| | Dune | 4 | **0** | 0.072 | **0.014** |
| | $\perp$ | 20 | **32** | 0.304 | **0.478** |
| salinity | fresh | 32 | 32 | 0.493 | 0.493 |
| | salt | 12 | **8** | 0.194 | **0.134** |
| | $\perp$ | 20 | **24** | 0.313 | **0.373** |

**Table 9: Per-attribute entropy $H_j(T_{\text{in}})$, divergence $\text{VAD}_j(T_{\text{in}}, T_{\text{out}})$, inverse-entropy weight $w_j(T_{\text{in}})$ ($\delta$=$10^{-9}$, $\gamma$=1), and final TLD for the running example.**

| Attribute $j$ | $H_j(T_{\text{in}})$ (bits) | $\text{VAD}_j$ | $w_j(T_{\text{in}})$ |
|---|---|---|---|
| name | 2.171 | 0.0678 | 0.406 |
| salinity | 1.487 | 0.0058 | 0.594 |

$\text{TLD}(T_{\text{in}}, T_{\text{out}}) = 0.406 \cdot 0.0678 + 0.594 \cdot 0.0058 \approx 0.031.$

# A Formal Definitions and Proofs

This appendix gives the formal definitions, the running-example intermediate computations, and the NP-hardness proof referenced from section 4.

## A.1 Data Model

*Definition A.1 (Schema).* A schema $\boldsymbol{S} = (\tau_1, \ldots, \tau_d)$ is an ordered list of attribute types. Each attribute $j$ has a finite domain $\text{Dom}_j \subseteq \text{Dom}(\tau_j)$ that includes the null token $\perp$. The first attribute is always the geometry, $\tau_1 = \texttt{Geometry}$.

*Definition A.2 (Feature, Tile).* A feature is a typed $d$-tuple $f \in \prod_{j=1}^{d} \text{Dom}_j$, with $f[j] \in \text{Dom}_j$ its $j$-th value. A tile $T = \{f_i\}_{i=1}^{N}$ is a finite set of $N$ features; we write $T \in \text{Rel}(\boldsymbol{S})$.

In Table 1, the schema is (`Geometry`, `String`, `String`) and the `name` domain is {Azul, Birch, Cobalt, Dune, $\perp$}. Even though `name` and `salinity` share the type `String`, their domains differ.

*Definition A.3 (Column size, tile size).* Column $j$ is the sequence $C_j(T) = \langle f_1[j], \ldots, f_N[j] \rangle \in \text{Dom}(\tau_j)^N$. A type-dependent encoder $\text{Enc}_j : \text{Dom}(\tau_j)^N \to \{0,1\}^*$ jointly encodes the column; its byte length is $\text{Size}_j(T) = |\text{Enc}_j(C_j(T))|$. The tile size is $\text{Size}(T) = \sum_{j=1}^{d} \text{Size}_j(T)$.

This abstracts dictionary/columnar encodings such as MVT [13] and MLT [31], in which nullifying values raises the frequency of $\perp$, shrinks the column dictionary, and reduces the encoded size.

## A.2 Pixel Footprints and Attribute Images

The rasterization of the running example used by the definitions in this subsection is shown in Figure 3 of section 4.

*Definition A.4 (Rendering function).* Given a tile resolution of $R$ pixels and a geometry $g$, the rendering function $\mathcal{R}(g) \subseteq \{1, \ldots, R\}$ returns the set of pixels covered by $g$.

*Definition A.5 (Attribute image).* For non-geometry attribute $j \in \{2, \ldots, d\}$, the attribute image $I_j^T : \{1, \ldots, R\} \to \text{Dom}_j$ paints each pixel with the attribute value of the feature whose geometry covers it: $I_j^T(\mathbf{x}) = f_i[j]$ if $\mathbf{x} \in \mathcal{R}(f_i[1])$ for some $i$, and $\perp$ otherwise.

Figure 3(c) shows the attribute image for `salinity` in $T_{\text{in}}$: pixels covered by $G_1$ and $G_4$ (fresh lakes) are labeled `f`; pixels covered by $G_2$ and $G_3$ (salt lakes) are labeled `s`.

## A.3 Pixel-Weighted Distributions

*Definition A.6 (Pixel-weighted distribution).* The pixel-weighted count of value $v \in \text{Dom}_j$ in tile $T$ is $c_j^T(v) = |\{\mathbf{x} \in \{1, \ldots, R\} : I_j^T(\mathbf{x}) = v\}|$. Counts are Laplace-smoothed with parameter $\varepsilon > 0$:

$$\tilde{p}_j^T(v) = \frac{c_j^T(v) + \varepsilon}{R + \varepsilon|\text{Dom}_j|}, \qquad v \in \text{Dom}_j. \tag{19}$$

Table 8 shows the counts for the running example. Nullifying Cobalt and Dune shifts $c_{\texttt{name}}(\perp)$ from 20 to 32. Even though `f` and `s` each occur in two rows of $T_{\text{in}}$, their pixel counts differ (32 vs 12) because the fresh lakes are physically larger.

## A.4 Per-Attribute Divergence and TLD

The visualization-aware divergence $\text{VAD}_j(T_1, T_2)$ is defined in Equation 1. It applies Jensen–Shannon divergence to the pixel-weighted distributions $\tilde{p}_j^{T_1}$ and $\tilde{p}_j^{T_2}$ defined above, so attribute changes on geometrically larger features contribute more to the divergence.

*Definition A.7 (Tile-level distortion).* The entropy of attribute $j$ in $T$ is $H_j(T) = -\sum_{x \in \text{Dom}_j} \tilde{p}_j^T(x) \log_2 \tilde{p}_j^T(x)$. The inverse-entropy weights are

$$w_j(T) = \frac{(H_j(T) + \delta)^{-\gamma}}{\sum_{k=2}^{d} (H_k(T) + \delta)^{-\gamma}},$$

where $\delta > 0$ avoids division by zero and $\gamma \geq 0$ controls the emphasis on low-entropy columns ($\gamma$=0 recovers uniform weights). The tile-level distortion is $\text{TLD}(T_1, T_2) = \sum_{j=2}^{d} w_j(T_1)\, \text{VAD}_j(T_1, T_2)$ (Equation 2).

Table 9 completes the example. `salinity` has lower entropy (1.487 bits, weight 0.594) than `name` (2.171 bits, weight 0.406), so a unit of salinity divergence costs more. The overall TLD≈0.031 is small because the nullified features are the two smallest lakes and the larger change falls on the higher-entropy, lower-weighted `name` column.

## A.5 Proof of Theorem 4.1

Proof. We reduce 0-1 Knapsack to visualization-aware tile reduction in polynomial time. Consider a Knapsack instance with $n$ items of sizes $s_i$, values $v_i$, and capacity $B$. Construct $T_{\text{in}}$ with $n$ features and two attributes (geometry plus one categorical attribute with a unique value per row, $f_i^{\text{in}}[2] = i$). Choose $\mathcal{R}$ so feature $i$ covers exactly $v_i$ pixels with no overlap and no empty pixels (line geometries of length $v_i$, with $R = \sum_i v_i$). Restrict $T_{\text{out}}$ so $f_i^{\text{out}}[2] \in \{i, \perp\}$, and define $\text{Enc}_2$ so that retaining a non-null cell in row $i$ costs $s_i$ bytes: $\text{Size}(T_{\text{out}}) = \sum_{i:\, f_i^{\text{out}}[2] \neq \perp} s_i$.

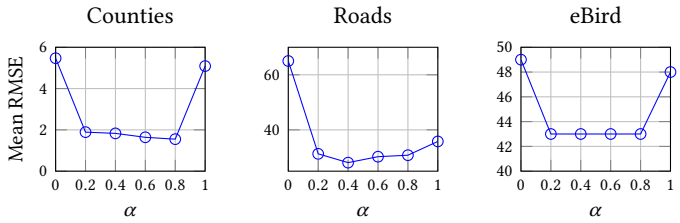



**Figure 14: Mean RMSE vs. $\alpha$ for three datasets.**

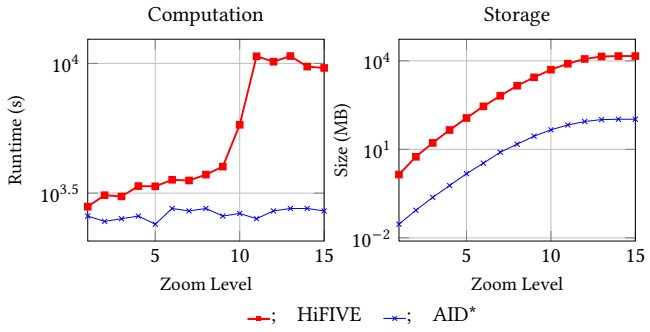



**Figure 15: Performance comparison across zoom levels 1–15 for eBird dataset. Runtime and output size are shown on a logarithmic scale for the ILP-based method (HiFIVE) and AID*.**

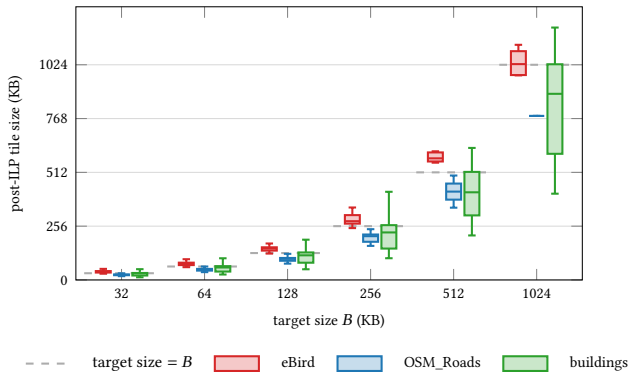



**Figure 13: Sparsification byte-budget adherence based on target size on the three geometry types across six target sizes $B \in \{32, 64, 128, 256, 512, 1024\}$ KB. Each box summarizes the post-sparsification sizes of all tiles the ILP actually reduced for that ($B$, dataset) cell. The dashed line marks size=$B$.**

With $\varepsilon = 0$ in Equation 19, a retained value $i$ contributes zero to $\mathrm{VAD}_2$, because $\tilde{p}_i^{T_{\text{in}}} = \tilde{p}_i^{T_{\text{out}}} = m_i$ in Equation 1. For a nullified value, all its mass moves to $\perp$: $\tilde{p}_i^{T_{\text{out}}} = 0$ and $m_i = \frac{1}{2}\tilde{p}_i^{T_{\text{in}}}$, so its KLD contribution is

$$\tilde{p}_i^{T_{\text{in}}} \log_2 \frac{\tilde{p}_i^{T_{\text{in}}}}{\frac{1}{2}\tilde{p}_i^{T_{\text{in}}}} = \tilde{p}_i^{T_{\text{in}}} \log_2 2 = \tilde{p}_i^{T_{\text{in}}} \propto v_i.$$

Therefore minimizing TLD is equivalent to maximizing the total $v_i$ of retained items subject to $\sum s_i \le B$, which is 0-1 Knapsack. □

The reduction uses $\varepsilon = 0$; positive Laplace smoothing perturbs the objective by $O(\varepsilon)$ but does not change the combinatorial structure of the problem.

## B ILP Solver Target Byte Size Adherence

*ILP byte-budget adherence.* To evaluate how well the linear size model predicts final serialized MVT size, we ran HiFIVE end-to-end at six target budgets and measured the post-sparsification size of every tile reduced by the ILP. Figure 13 contrasts the three geometry types. OSM_Roads sits cleanly below the identity line size=$B$ at every threshold (100% of ILP-binding tiles within budget; median uses 64–82% of $B$), confirming that on linestrings the ILP both fits and gracefully under-shoots the cap when the column-cost model leaves headroom. eBird (points) sits essentially on the identity line —medians within 5–10% of $B$ at every threshold, IQR within ±15% — demonstrating that even on point datasets with 45 string attributes the model tracks the actual MVT encoding accurately. OSM_Buildings shows the highest relative variance at small budgets (IQR≈50% of $B$ at 32 KB) because dense polygon tiles dominate the encoded size with geometry rather than dictionary bytes; this relative spread narrows to ≈20% by $B$=1024 KB as the budget becomes large enough for the linear model to absorb the discrete jumps in geometry cost. Across datasets and budgets, the median post-sparsification size remains close to the target budget, supporting the linear model as a practical proxy for serialized MVT size.

### B.1 Sensitivity to $\alpha$

Figure 14 shows the effect of $\alpha$ in Equation 10, which balances visual prominence from pixel footprint and attribute information from JSD. The two extremes, $\alpha = 0$ and $\alpha = 1$, perform worse because they optimize only one component of the objective. Intermediate values are more stable across datasets, supporting the default choice of $\alpha = 0.5$.

Figure 15 highlights the effect of pyramid depth (varied from 1 to 15) on runtime and storage for eBird (Tippecanoe is omitted because it runs out of memory). As expected, the unstyled raster tiles of AID* exhibit the smallest size and fastest runtime, at the cost of not supporting any client-side styling. HiFIVE's runtime and size initially increase with the number of tiles, then stabilize because the target byte budget limits the growth of each reduced tile.