A Generic Machine Learning Model for Spatial Query Optimization based on Spatial Embeddings

ALBERTO BELUSSI, University of Verona, Verona, Italy
SARA MIGLIORINI, University of Verona, Verona, Italy
AHMED ELDAWY, University of California Riverside, Riverside, United States

Machine learning (ML) and deep learning (DL) techniques are increasingly applied to produce efficient query optimizers, in particular in regards to big data systems. The optimization of spatial operations is even more challenging due to the inherent complexity of such kind of operations, like spatial join or range query, and the peculiarities of spatial data. Although a few ML-based spatial query optimizers have been proposed in literature, their design limits their use, since each one is tailored for a specific collection of datasets, a specific operation, or a specific hardware setting. Changes to any of these will require building and training a completely new model which entails collecting a new very large training dataset to obtain a good model.

This article proposes a different approach which exploits the use of the novel notion of *spatial embedding* to overcome these limitations. In particular, a preliminary model is defined which captures the relevant features of spatial datasets, independently from the operation to be optimized and in an unsupervised manner. This model is trained with a large amount of both synthetic and real-world data, with the aim to produce meaningful spatial embeddings. The construction of an embedding model could be intended as a preliminary step for the optimization of many different spatial operations, so the cost of its building can be compensated during the subsequent construction of specific models. Indeed, for each considered spatial operation, a specific tailored model will be trained but by using spatial embeddings as input, so a very little amount of training data points is required for them. Three peculiar operations are considered as proof of concept in this article: range query, self-join, and binary spatial join. Finally, a comparison with an alternative technique, known as transfer learning, is provided and the advantages of the proposed technique over it are highlighted.

CCS Concepts: • **Information systems → Database management system engines**; • **Computing methodologies → Machine learning approaches;**

Additional Key Words and Phrases: Query optimizer, machine learning, big data, range query, spatial join, spatial embedding

ACM Reference Format:
Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2024. A Generic Machine Learning Model for Spatial Query Optimization based on Spatial Embeddings. *ACM Trans. Spatial Algorithms Syst.* 10, 4, Article 36 (October 2024), 33 pages. https://doi.org/10.1145/3657633

Authors' Contact Information: Alberto Belussi, Department of Computer Science, University of Verona, Verona, Italy; e-mail: alberto.belussi@univr.it; Sara Migliorini, Department of Computer Science, University of Verona, Verona, Italy; e-mail: sara.migliorini@univr.it; Ahmed Eldawy, Computer Science and Engineering, University of California Riverside, Riverside, California, United States; e-mail: eldawy@ucr.edu.

ACM 2374-0353/2024/10-ART36
https://doi.org/10.1145/3657633

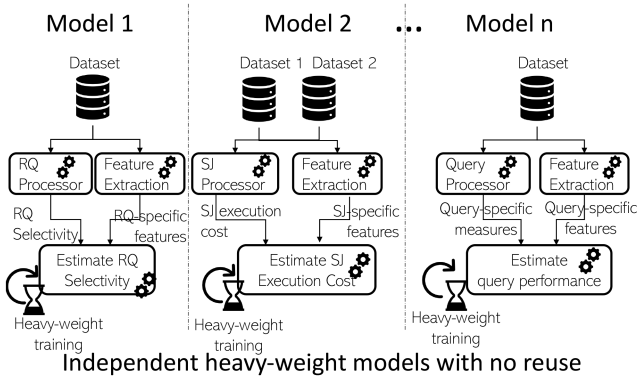
1 INTRODUCTION

In the last years, big data analytics has increased its strategic role in supporting decision systems, thanks to the growing availability of data, sometimes in heterogeneous formats. Very often geo-referenced data and spatial objects represent a significant part of the datasets subject to analysis, leading to the development of many spatial big data systems and libraries [10, 11, 38]. Due to the complexity and richness of some kinds of analysis, in many cases data processing tasks are structured as pipelines of operations [27], and this allows the definition of many alternative ways to produce the requested result. Moreover, the existence of different alternatives could also be originated by the fact that each single operation on spatial data can be implemented in several ways and by applying different algorithms. This could be further exacerbated in distributed and cluster-based systems, since the parallel execution of an algorithm often requires the tuning of several parameters, which depend on both the cluster configuration and the characteristics of datasets at hand. The immediate consequence of such richness of alternatives is the increasing importance covered by the presence of *query optimizers* able to automatically guide towards the choice of the best execution plan in terms of performances.

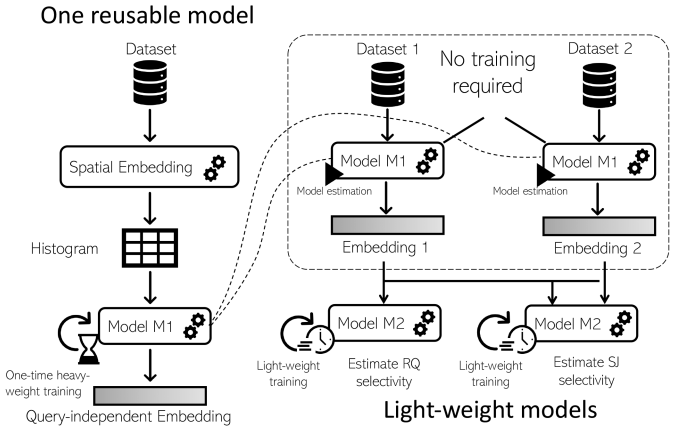
The spatial query optimization problem could be formulated as follows: given an operation o to be performed, the input dataset (or datasets) and the available hardware and software cluster configuration: (i) identify the different alternative implementations of o available on the cluster, (ii) estimate the cost of each of them and choose the best one, denoted as i, based on the characteristics of the input dataset (or datasets), and (iii) estimate the best parameter configurations p for tuning i in the given cluster. The problem becomes even more complicated if the data processing requires a sequence of operations to be applied (like in a pipeline), since the optimization of each single operation could also depend on the operations previously executed or on the operation order. In general, the main objective of each optimization strategy is to produce an estimation of the cost of each available alternative solution.

Recently some techniques have been proposed to address the optimization problem described above, as we will deeply discuss in Section 2. In order to provide an effective solution to this complex task, many of them are based on estimation approaches implemented through **machine learning** (**ML**) or **deep learning** (**DL**) models, with the general architecture summarized in Figure 1(a). As shown in the figure, these models are completely independent from each other although they could share a lot of similarities depending on what they estimate. For example, two models that estimate the range query selectivity and the range query running time are expected to share many similarities in terms of the considered features. However, as a consequence of their independence, each model requires a heavy-weight training step that works on its own large training dataset. In particular, the training dataset needs to be large enough to be able to capture the intricate relationship between input data, query characteristics, and hardware specifications. Building each training dataset is a challenging problem itself, since it has to be diverse enough to capture the effect of all these parameters on the required estimation.

Given these considerations, the main limitations of existing solutions can be summarized as follows: (i) each model is tailored to one specific collection of datasets. Thus, if this set changes significantly, the model must be retrained, consistently reducing the generality and reusability of the solution. In order to overcome this problem, it is necessary to use a collection of datasets even synthetically generated, which is able to capture the relevant characteristics and behaviour of spatial objects in the execution of spatial operations. In this regard, generators of synthetic datasets with different distributions have been proposed [18, 36], so that models can be trained on a variety of input datasets and become independent from a given specific collection of datasets. (ii) Models are usually dedicated to a single specific operation; hence, the extension to other operations requires to collect new data points, namely to execute a great number of experiments to collect the desired



(a) Existing ML-based query optimization models create a separate model from scratch for each problem and cannot reuse models.



(b) Proposed approach trains a model for spatial embedding and reuses it in several models to reduce training time.

Fig. 1. Existing ML-based query optimization models and the proposed spatial-embedding-based approach.

parameters to estimate, and subsequently to retrain the model on them. Moreover, (iii) the results obtained with a specific cluster cannot be easily generalized to different clusters, since also the system configuration has an impact on the cost estimation. Therefore, such expensive activity, performed for building the training set, should be repeated for each cluster configuration. Some attempts to overcome this last limitation have been proposed [33] where few metrics have been identified for determining the best partitioning technique, which are independent from the adopted cluster configurations and depend only on the dataset features. Similarly, some metrics independent from the cluster configurations are used also in [34] where some models for estimating the fastest implementation of an operation on a given dataset or pair of datasets have been proposed. However, this last proposal leads to another limitation: (iv) these solutions are based on the extraction of some features from the input datasets and such features can be different according to the operation we need to estimate the cost of. Thus, they must be recomputed for each operation we consider.

This article proposes a generic ML-based model for spatial query optimization that overcomes the limitations of existing work. It can estimate *different cost parameters* regarding the execution

of *multiple implementations* of some *spatial operations* which are independent from the specific input collection of datasets and the cluster characteristics. The proposed framework, illustrated in Figure 1(b), is characterized by the following components:

(1) A first ML model ($M1$) that is trained on a large dataset D with the goal of extracting a set of significant features that are *independent from the operation and the hardware*. Thanks to its independency and generality, it needs to be trained only once and can be reused for any spatial operation that we want to build an estimation model for. Moreover, since it is trained only once, we can invest more time in training this model to make sure it works with a wide range of datasets including synthetic and real datasets. We call this model *spatial embedding*, since it creates a compact summary of the input spatial data that can then be used in any ML model. To build the spatial embedding, we generate and use a *multifaceted histogram* representing the distribution of some features in the reference space of D (Minimum Bounding Rectangle of D). Furthermore, model $M1$ can be trained in an *unsupervised manner* with a large amount of data, even automatically generated with tools like [18, 36], and whose result does not depend on the specific spatial operation.
(2) Given an operation and a specific implementation for it, a set of *cost parameters* are chosen. Operations can be for example: range query, self-join, or binary spatial join. Different implementations of the same operation can be: index-based and scan-based range query algorithms, or partition-based and index-based spatial join algorithms.
(3) For each chosen combination of an operation o, implementation i, and parameters p, a model $M2(o, i, p)$ is trained starting from the spatial embeddings of the input dataset(s) produced by the first model. Since $M1$ is already trained on large amount of spatial data, the amount of data points required for training $M2$ is reduced, consequently limiting the cost needed to execute expensive spatial operations.

In summary, the proposed framework is composed of a unique unsupervised model $M1$ for producing spatial embeddings, and a supervised model $M2$ for each cost parameter p of a given implementation i of a spatial operation o. More specifically, the training of model $M1$ requires a large collection of synthetic datasets covering the most common distributions of real spatial data. This collection can be automatically generated by using the Spider tool [18], which can produce a large number of synthetic datasets with a small effort. Conversely, each model $M2(o, i, p)$ requires the execution of the chosen implementation i of the operation o where the cost parameter p is measured. Thus, the generation of such data points is costly and can require several hours of processing, depending on the operation we consider. The idea is to use the embedding produced by $M1$ as input for each model $M2$ to reduce the size of the required training set, consequently reducing both the cost for its generation and the training time. In this way the cost of training $M1$ is amortized, since it is trained only once, while models $M2(o, i, p)$ can use a training set with less data points and of reduced size.

This article widely extends our preliminary work [8] where the notion of spatial embedding has been introduced for the first time and some initial experiments on only synthetic data and range query operation have been presented. First of all, in order to prove the effectiveness of the framework, in this article, we focus on: (i) three operations, range query, self-join and binary spatial join; (ii) multiple implementations, the one provided by the library Beast on Spark [10]; (iii) two cost parameters for each operation, selectivity and number of MBR tests. In this regard, selectivity is the ratio between the cardinality of the result of the operation execution and the maximum cardinality the operation can produce, while the number of MBR tests is the number of comparisons between two MBRs that are performed during the execution of the chosen implementation. Notice that the former is independent from the chosen implementation and the cluster configuration,

while the latter only depends on the implementation. Second, experiments are performed by considering both synthetic and real datasets. These experiments show that spatial embeddings can be generated by different models and with different **latent dimensions** (**LD**), but the most effective one, starting from histograms of shape $128 \times 128 \times 6$ is of dimension 3,072 or less. Moreover, we showed that the same embedding can be used for the estimation of different cost parameters of different operations. The estimates produced by models $M2$ outperform some **baseline** (**BL**) values obtained by applying other consolidated estimation techniques. We also demonstrate that the use of spatial embeddings in place of the original histograms can reduce the dimension of the training set and also the amount of computational resources needed to train models $M2$. Finally, a comparison with an alternative approach, known as transfer learning, is proposed which further highlights the advantages of the proposed framework based on spatial embeddings.

The remainder of this article is organized as follows. In Section 2, previous solutions for the estimation of query execution costs are presented. Section 3 introduces the concept of spatial embedding and describes how they can be generated starting from spatial vector datasets. Section 4 presents the three spatial operations considered in this article, range query, self join and two-way spatial join, together with their cost parameters that we want to estimate. Section 5 illustrates the results of the performed experiments about the construction of spatial embeddings, and the estimation of the various operation parameters. In this section, the benefits of using spatial embeddings with respect to histograms during training are discussed and a comparison with the transfer learning technique is also presented. Finally, Section 6 presents conclusions and future work.

2 RELATED WORK

The optimization of query and processing operations has been the subject of many research works starting from the advent of databases and now revamped with the spread of big data systems. In this context, spatial data covers a particular role due to the peculiarity of its operations, like spatial join and range query.

Cost-based optimization – Due to the high cost of such operations and its inherent complexity, the optimization has been studied in terms of both selectivity estimation and join cost estimation. Regarding the first aspect, several works have been proposed in the past to define some formulas or parameters that provide an estimation of the join selectivity, with respect to both uniformly distributed datasets [3] and skewed datasets [5, 12]. These methods have been recently extended to deal with large amounts of data processed by big data systems, like SpatialHadoop [11] and GeoSpark [38]. More specifically, in [31] a cost-based and a rule-based optimizer has been proposed for MapReduce. It has two limitations: firstly it does not consider all possible spatial join algorithms, and secondly, it requires to experimentally collect several parameters in order to properly catch the characteristics of the hardware, algorithms and data at hand. A more detailed model has been proposed in [7], where the cost is subdivided into three main components: CPU, local, and network I/O. It essentially overcomes the limitations of the previous work, but its applicability is limited to uniformly distributed data. The extension of this work to other non-uniform distributions requires the definition of more complex models that can capture the effects of skewness on spatial operations. In [6], the authors extend the use of the correlation fractal dimension for determining the level of skeweness, and consequently the kind of distribution, of a spatial dataset. However, the model has to become very complex to correctly capture all the facets of a dataset distribution and exploit them for correctly partitioning big datasets in the right way. Therefore, the use of ML approaches has begun to be experimented as an alternative way to build sophisticated models [33].

ML/DL optimization – Due to the inherent complexity of many big data operations and the amount of factors that can influence their costs, like the distribution and size of the dataset, or

the used algorithms and clusters, the identification of precise theoretical formulas is not always possible. Therefore, in recent years many different attempts have been made in order to exploit ML and DL techniques for building sophisticated data-driven models for selectivity estimation [14, 20, 37], join cost estimation [21, 23, 25], and join order enumeration [24]. These methods suffer from two limitations: (i) they only work with equi-joins and do not support the complex logic of spatial data, (ii) they are trained with a small number of tables and specific datasets and can produce an estimation only for them. DeepDB [15] attempted to address the second approach by building a pure data-driven model that tries to capture the correlation across attributes and the data distribution of single attributes. This would help in supporting more complex queries and join operations between multiple tables. It achieves this through a probabilistic query compilation procedure that translates a generic query into an evaluation of expectations and probabilities. That approach is a first step towards building a model that applies across many datasets that were not part of the training process. However, it was limited to alphanumeric databases and cannot be applied to spatial data and operations. Indeed, spatial data have some specific characteristics that make them different from any other alphanumerical data type. For instance, spatial data objects are multidimensional, and in presence of more than one dimension, there is no ordering able to preserve proximity, while many of the existing techniques for alphanumeric data (like equi-joins) rely on the fact that neighbouring objects are always adjacent to each other. Moreover, spatial data have an extent which makes other techniques essentially impractical.

A first attempt to extend the use of ML techniques for distributed spatial join selectivity and cost estimation has been done in [34]. It differs from existing ML-based query optimizer as it supports spatial join and can be applied to any input data that was not part of the training set. It also overcomes the limitation of existing theoretical approaches, because it can deal with skewed datasets including real-word ones and it works on well-defined data statistics that can be collected in a simple data scan. However, as emphasized in the introduction, this approach has two main limitations: (i) it requires to collect a big training set which in turn requires to perform a large amount of expensive spatial join operations, namely the construction of the training set is much more costly than the training, (ii) a different training has to be performed for each operation we want to optimize (see Figure 1(a)). This article performs a step forward by introducing the concept of *spatial embedding* with the aim to perform a preliminary unsupervised training on large amount of data, which can be easily automatically generated [18] and then used in the subsequent phase for reducing the amount of training data points needed for each particular operation to be optimized (see Figure 1(b)).

Embedding – Autoencoders are neural networks typically used to learn a compressed representation of a dataset, known as embedding. The notion of embedding has been developed for solving the problem of dimensionality reduction and it comes from the assumption that there may exist a small number of variabilities which can guarantee the "semantics" of the original high-dimensional data [16]. It has been shown that autoencoding is a powerful way to learn the hidden representations of data, since most of them focus on the locality-preserving property of embedding [39].

In some sense autoencoders can be considered a form of image compression algorithms, but they are much more. Their main purpose is to balance two criteria: (i) the compactness of representation, measured by its capability of compressing the input and reconstructing it (i.e., like any compression technique), and (ii) its ability to extract behaviorally relevant variables from the input, so that similar inputs have a similar compact (embedding) representation. More specifically, embeddings try to capture the relevant features for the problem at hand, not necessarily those that allow the best reconstruction. In this way, embeddings allow to provide a similarity measure between different input datasets, and also a more meaningful clustering of them. In other words,

the degree of separation in the embeddings translates into a degree of separation in the original inputs. Embeddings have been successfully used in many DL applications, from natural language processing (see the Word2Vec technique [28]) to image processing [22]). In this article, we specialize the general notion of embedding to the more specific one of spatial embedding. With a spatial embedding, we can cluster all the possible input datasets with respect to their spatial properties, reducing the amount of necessary training data points.

Embedding Space for Regions – In recent years an emerging problem known as **Learning an Embedding Space for Regions** (**LESR**) has been defined in literature [13]. The term "region" refers to an urban region that consists of a location and a neighbour area inside which many **Points-of-Interests** (**POIs**) representing urban functions are located. The main idea is to build a graph where nodes represent POIs and edges are defined based on several viewpoints on neighborhood, for example, distance or connectivity. Then, graph embedding is used on that network to define a similarity between regions which capture both spatial and semantic properties. A follow-up work uses several sources of information when constructing the model, like satellite images, crowd-sourced geo-tagged data, trajectories, and GPS traces, which produces a multimodal embedding [17]. Even if these works represent a first attempt to use the notion of embedding in the spatial domain, they are limited to points and the spatial aspects that are used during graph construction. However, existing work in regional embedding cannot be directly used for our problem due to two factors. First, the constructed graph only captures the relationship among points and not their location in the reference space which makes them inapplicable for range queries and spatial joins where the location is an integral part of the query. Second, the above work only considers points while we consider objects with extents, for example, lines and polygons, for both range query and spatial join. The notion of spatial embedding proposed in the following section starts from a vector representation of a spatial dataset with a variable size and composed of any kind of spatial object, and can condense all the information influencing the cost of a generic spatial operation.

3 SPATIAL EMBEDDING

DL techniques often follow a pattern that tries to organize the architecture of a neural network in two steps. In the first step the goal is to extract the significant information that is contained in the input data so that the second step can be fed with a distillate of the original input where noise has already been purged. This condensed information is usually called *embedding* and it can be orders of magnitude smaller than the original data. The second step focuses on the prediction of the target value, which can be a class of a taxonomy or a forecast of a given parameter. As discussed in Section 2, the distillation of the embedding is very often applied in Natural Language Processing [28] and image processing [22], with the additional effect that the next step is proven to gain accuracy with respect to models in which embeddings are not distilled.

A neural network used for this kind of task is called *autoencorder* and can be implemented as a stack of fully connected layers (*stacked autoencoder*), or of convolutional neural layers (*convolutional autoencoder*). As illustrated in Figure 2, in a stacked autoencoder each layer has the responsibility to reduce the dimensionality of the input with also the aim to detect at each step the most interesting features. Conversely, as illustrated in Figure 3, a convolutional autoencoder typically reduces the spatial dimensionality of the inputs (i.e., height and width) while increasing the depth (i.e., the number of feature maps). A convolutional autoencoder has typically a flatten layer which reshapes the output in order to produce the final embedding.

In its general form, an autoencoder has a symmetric architecture, namely with reference to Figures 2–3, an autoencoder includes also a stack of reverse layers which allow to reconstruct the original input dataset starting from the obtained embedding. In this way, the training of an autoencoder *AE* is unsupervised, since the input set of data points is at the same time the training

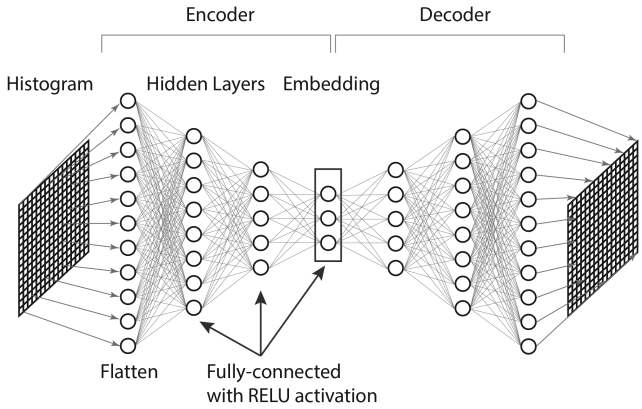


Fig. 2. Architectures of a stacked autoencoder.

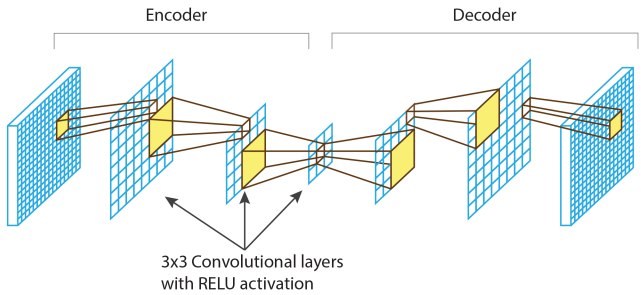


Fig. 3. Architecture of a convolutional autoencoder.

set and the ground truth for the training. Indeed, the metrics used as loss function for the evaluation of the trained model is usually the Mean Squared Error and is computed as the average of the difference between an original data point p_0 and the reconstructed one, that is,:

$$error(p_0) = abs(p_0 - AE.decode(AE.encode(p_0))),$$

where $AE.decode()$ and $AE.encode()$ refer to the application of the encoding and decoding functionalities of the autoencoder, respectively; while the overall loss function becomes:

$$loss = \sum_{i=1}^{n} \frac{error(p_i)^2.}{n}$$

In this article, we propose to apply this approach to the spatial optimization problem; in particular, for generating a condensed representation of the input dataset which correctly synthesizes its peculiar characteristics w.r.t. the cost of spatial operations. As already mentioned in Section 1, this allows us to factorize some large amount of work independently from the given operations. Therefore, we can subsequently produce a model tailored for a specific spatial operation with less training data points. Since this distillation process regards a generic spatial dataset, represented as a collection of vector geometries, instead of an image, it is necessary to perform the following preliminary operations:

(1) Define a format for the model input, indeed while images have a fixed structure (i.e., a grid of pixels), spatial dataset can vary a lot.

Table 1. Datasets used for Training Model $M1$ which Produces the Spatial Embeddings

Distrib.	Count	Cardinality		Size (in bytes)	
		min	max	min	max
Synthetic datasets					
Uniform	1,145	55 K	50 M	19 MB	24 GB
Diagonal	345	17 K	50 M	35 MB	24 GB
Gaussian	343	25 K	46 M	72 MB	22 GB
Parcel	225	24 K	50 M	23 MB	22 GB
Bit	305	23 K	50 M	54 MB	22 GB
Sierpinski	200	52 K	25 M	26 MB	11 GB
Real datasets					
TIGER2018	103	3,191	78,741,390	204 KB	3.76 GB
OSM Lakes	148	9,950	2,933,002	4.5 MB	1.37 GB
OSM Parks	127	10,167	10,445,012	3.3 MB	4.20 GB

Columns **Count** reports the number of datasets for each distribution. Synthetic datasets have been generated by using the SpiderWeb tool [18], while real datasets have been obtained by splitting the global dataset available at https://star.cs.ucr.edu/ by using a fixed grid.

(2) Generate a large number of spatial datasets that cover as much as possible the different real distributions that characterize geographical information on the Earth surface.
(3) Define the structure of the neural network that, after training, can be used for generating the spatial embedding.

Regarding point (1), we choose to compute for each dataset a *multifaceted histogram* of fixed size, as done in many previous work on the field [1, 2, 9, 26, 30, 32–34]. For deciding which features to store in each cell of the histogram, we consider the goal of the successive models: *evaluate the cost of an operation on spatial data*. Usually, such cost is influenced by the **dataset size** and the **complexity of the geometries** contained in the dataset. As representative features for the dataset size we choose the *cardinality of the input* (f_1), namely the number of geometries contained in the input dataset, and the *size in bytes* (f_2) of the input file. Conversely, for measuring the complexity of the geometries we use: the *area of the geometries* (f_3), namely the area of the region enclosed by their boundaries, the *length on x and y axes of their MBR* (f_4 and f_5), namely the width and height of their MBR, and finally the *number vertices* (f_6) of their vector representation, namely the number of points used to describe the boundary of the geometries. Indeed, the vector representation of geometries is a way to describe spatial objects in terms of the list of points/vertices composing its external and/or internal boundaries. These features are computed for each cell of the histogram so that the distribution of the dataset "complexity" in the reference space can be represented by the histogram itself. In particular, in each cell c we compute the sum of features f_1, f_2 and f_6 and the average for features f_3, f_4, and f_5 considering only the geometries that intersect the cell c.

Conversely, for point (2), we generate a collection of about 2,550 synthetic datasets with different distributions and about 380 real datasets. Details concerning such datasets are reported in Table 1. We also choose different sizes from a minimum of about 20 MB to more that 20 GB. Notice that the number of datasets with uniform distribution is higher with respect to the number of other distributions (see column **Count**). This choice is necessary because with highly skewed distributions there are many empty cells inside the reference space. Therefore, to ensure that the computed histograms are balanced between zero and non-zero values, and to avoid creating a model that

over-estimates zeros, we need in this phase a bigger number of uniformly distributed datasets which compensates the overall presence of skewed distributions (independently from their type). The final size of the dataset containing the multifaceted histograms computed on these spatial datasets, which represent the input data points for the first model $M1$ of the architecture in Figure 1(b), is 2.2 GByte.

As final preparation step, we need to normalize the values of the features in each cell of the histograms. Different functions can be applied to normalize data before training. We chose the simplest one, which requires firstly to compute the minimum and the maximum value for each feature among all histograms, producing two arrays of values $min[]$ and $max[]$, and secondly to normalize each original list of feature $v_{orig}[i][]$ as follows:

$$v_{norm}[i][j] = \frac{v_{orig}[i][j] - min[i][j]}{max[j] - min[j]}. \tag{1}$$

Finally, relatively to point (3), we teste both kinds of mentioned autoencoders: the stacked and the convolutional ones. For the stacked autoencoders we choose three fully connected layes (as in Figure 2), while for the convolutional autoencoders we use three convolutional layers (see Figure 3). We try different values for the embedding dimension (also called *LD*), and different numbers of nodes in each layer. As we will see in Section 5, the trained models are able to reconstruct histograms which are very similar to the original one (see for instance Figure 4). However, as we mentioned in Section 2, the aim of spatial embeddings is not necessarily to rebuild correctly the original histograms, but to correctly distill the significant features that are relevant for the following tasks, and, in our case, for the estimation of the cost parameters of spatial operations. Therefore, taking this consideration in mind, we do not choose for the second step only the model that produces the best reconstruction, namely the one with the smallest reconstruction error, but we consider a set of *promising* models that while keeping good behaviour are able to reduce as much as possible the LD. Indeed, the more we reduce the size of the spatial embedding, the more we can obtain a significant distillation of the dataset characteristics for training the next model.

In the next section, we describe how the results obtained by model $M1$ can be used to train a set of models $M2$, each one tailored for a specific spatial operation, considering in particular the range query, the self-join and the binary spatial join.

4 ESTIMATING THE COST OF SPATIAL OPERATIONS

After training the first model $M1$, we need to define the second part of the proposed approach in Figure 1(b). As previously mentioned, while $M1$ can be trained only once starting from a large collection of spatial datasets covering a great variety of distributions, the second model has to be defined for each operation and for each metric that we want to use for measuring its cost.

In this article, we consider three spatial operations: the *Range Query*, the *Self Join* and the *Binary Spatial Join*. For the first one we use *selectivity* as metric for estimating its cost, while for last two we use two metrics: *selectivity* and *number of MBR tests*. As already discussed in literature, these metrics are good indicators of the cost of range query and spatial join operations [3, 34], respectively. In the following the generic term *spatial join* is used to denote both the self-join and the binary spatial join when there is no reason for distinguishing them.

In case of range query, selectivity is the ratio between number of elements returned by the operation and the total number of elements in the input dataset.

$$sel_{RQ}(D, R) = \frac{|\{g \in D : g \cap R \neq \emptyset\}|}{|D|}, \tag{2}$$

where D is the input dataset, R is the range window used by the operation and $g \cap R$ returns the result of the intersection between the geometry g and the query window R.

Similarly, join selectivity is defined as the ratio between the actual number of pairs produced by the join operation and the number of pairs produced by the cross product:

$$sel_{SJ}(D_1, D_2) = \frac{|\{(g_1, g_2) \in D_1 \times D_2 : g_1 \cap g_2 \neq \emptyset\}|}{|D_1 \times D_2|}. \tag{3}$$

Clearly in case of the self join, the two datasets D_1 and D_2 are exactly the same.

Finally, in spatial join the number of MBR tests reports the number of tests performed on MBRs by the operation implementation itself. It is computed as:

$$\#mbr_{SJ}(D_1, D_2) = \left(\sum_{\substack{p_i \in P_i \\ q_j \in P_j}} \sum_{\substack{g_1, g_2 \in \\ p_i \times q_j}} \#MBRtest(g_1 \cap g_2) \right) / |D_1 \times D_2|, \tag{4}$$

where P_* returns the set of partitions in which the dataset D_* has been subdivided in order to perform the join operation, while $g \in p$ returns the geometries contained in the partition p, and $\#MBRtest(g_1 \cap g_2)$ computes the number of MBR comparisons that are performed for testing the not empty intersection between two geometries g_1 and g_2. Notice that, unlike Equation (3), all (g_1, g_2) pairs are considered even if they do not overlap, since some of them still need to be tested to know that they are not in the result. For the self-join, not only the datasets D_1 and D_2 are the same, but also the partitions P_i and P_j.

In the next subsections, we will show how they can be used in our second model $M2$ for estimating the cost of the three spatial operations, while Section 5 will illustrate some experimental results about the ability of spatial embeddings to distill the significant features necessary to estimate the desired cost in the correct way.

4.1 Range Query Estimation

As discussed in the previous section, for each spatial operation we need to instantiate a model $M2$ which will work on the dataset embedding produced by $M1$ and a set of specific information. In case of the range query, we add to the embedding the following eight features: 4 values representing the dataset MBR and 4 values representing the range query window. Therefore, each input data point for model $M2$, which estimates range query selectivity, is the tuple:

$$\begin{aligned} \langle emb(D), min_x(D), min_y(D), max_x(D), max_y(D), \\ min_x(R), min_y(R), max_x(R), max_y(R), \sigma \rangle \end{aligned} \tag{5}$$

where $emb(D)$ is the embedding of D computed by model $M1$, while $min_x(D)$, $min_y(D)$, $max_x(D)$, $max_y(D)$ is the MBR of D and $min_x(R)$, $min_y(R)$, $max_x(R)$, $max_y(R)$ is the MBR of the range query window, finally σ is the selectivity value to be estimated. The main idea is that the MBR of the dataset and the MBR of the range query window are enough to specialize the information distilled by the spatial embedding.

Example 4.1. Let us consider as an illustrative example the case of a dataset D composed of 100K geometries which are enclosed in an MBR described by the minimum coordinate $(0, 0)$ and a maximum coordinate of $(1, 1)$, while the range window R has an area of 10^{-8} starting from a bottom left coordinate of $(0.1, 0.1)$. The tuple in Equation (5) becomes:

$$\langle emb(D), 0, 0, 1, 1, 0.1, 0.1, 0.1001, 0.1001, 0.05 \rangle,$$

where $emb(D)$ is the embedding of D represented as a tensor whose dimension depends on the chosen LD, while $\sigma = 0.05$ means that only 5% of the geometries in D intersects the range query window.

4.2 Spatial Join Estimation

The data points for the binary spatial join operation are built by following the same idea illustrated in the previous section for the range queries. However, in this case we have two datasets at hand that have to be represented; in particular, their spatial embeddings have to be properly combined. We chose to synthesize the spatial embeddings of the two input datasets by overlapping them, so that if they have individually a dimension of for instance $32 \times 32 \times 3$, we obtained a tensor of $32 \times 32 \times 6$. Therefore, similarly to Equation (5), the input of model $M2$ for the estimation of the spatial join metrics is given by the following tuple:

$$\begin{aligned}\langle emb(D_1D_2), min_x(D_1), min_y(D_1), max_x(D_1), max_y(D_1),\\ min_x(D_2), min_y(D_2), max_x(D_2), max_y(D_2), v\rangle\end{aligned} \tag{6}$$

where $emb(D_1D_2)$ is the representation of the combined embedding, the following four values represent the MBR of D_1, the other four values are the MBR of D_2, while v is the value of the metric to be evaluated (i.e., selectivity or number or MBR tests).

For the self join, the representation is essentially the same except that D_1 and D_2 are the exactly same dataset.

Example 4.2. Let us consider as an illustrative example the case of two datasets D_1 and D_2 with the following characteristics. D_1 contains 50 K geometries which are enclosed in an MBR described by the minimum coordinate $(0, 0)$ and a maximum coordinate of $(0.8, 0.8)$, while D_2 contains 25 K geometries which are enclosed in a MBR with minimum coordinate $(0.3, 0.3)$ and a maximum coordinate of $(0.9, 0.9)$. The tuple in Equation (6) becomes:

$$\langle emb(D_1D_2), 0, 0, 0.8, 0.8, 0.3, 0.3, 0.9, 0.9, 0.01\rangle, \tag{7}$$

where $emb(D_1D_2)$ is obtained by combining the embedding of the D_1 with the embedding of D_2. As mentioned above, if for instance each single dataset has an embedding of size $32 \times 32 \times 3$, $emb(D_1D_2)$ is represented by a tensor of dimension $32 \times 32 \times 6$. Finally, if v is the selectivity, a value equal to 0.01 means that only 1% of the cartesian product ($50K \times 25K$) will be included in the result. Conversely, if the element v represents the number of MBR tests and has a value of 10,000, it means that 10,000 intersection tests between MBRs will be necessary to identify the spatial join result.

5 EXPERIMENTS

Considering the architecture of the proposed approach, we divide the experiments in two parts. First, we need to train model $M1$ for obtaining a set of candidate autoencoders which are able to generate meaningful spatial embeddings of a given histogram dataset representation. Second, we need to train a model $M2$ for each considered spatial operation and parameter.

The source code developed for the experiments (together with the training and test sets) are available as a GitHub repository.[1] The experiments have been performed on a server equipped with an Intel i9-10900X processor, with 64 GByte of RAM, a GPU NVIDIA Quadro RTX A5000 GPU with 24 GByte of dedicated RAM, and Ubuntu 22 operating system with Nvidia driver version 535.86.05 and CUDA version 12.2.

[1] https://github.com/smigliorini/spatial-embedding

Table 2. Experiment Exp^1_{M1} Performed by using **Stacked Autoencoders**, which have been **Trained with Only Synthetic Datasets** and Tested on Both Synthetic and Real Datasets

Autoencoder	Latent Dim.	Hyperpar.	Training time(sec)	LOSS	VAL LOSS	WMAPE	WMAPE REAL
AE_{S1}	384	1,024,512	105	9.6E-04	1.5E-03	0.363	57.12
AE_{S2}	1,536	1,024,512	104	1.1E-03	1.6E-03	0.356	54.06

The spatial embeddings have been extracted from *histograms of* $128 \times 128 \times 6$ and the training has been performed with 50 epoches.

5.1 Spatial Embeddings

As previously discussed, the training of model $M1$ requires the preparation of a set of data points by generating the histograms for the datasets listed in Table 1. Every histogram is a grid of 128×128 cells, each one containing six features, as described in Section 3. Given such set of input data points, we perform four set of experiments in order to find the best architecture for $M1$. The four set of experiments will be denoted as Exp^1_{M1}, Exp^2_{M1}, Exp^3_{M1} , and Exp^4_{M1}, respectively. At the end of this phase, we select a collection of 8 good autoencoders which will be tested in the second part of the experiments regarding model $M2$. For not cluttering the presentation, we report in this section only the experimental results obtained with the selected good autoencorders, while the detailed results of each experiment are contained in Section A.1 of Appendix A.

In the first set of experiments, denoted as Exp^1_{M1}, we consider for the training only *synthetic datasets* and the used models are *stacked autoencorders* composed of three dense layers. In this experiment, we evaluate also the capability of a stacked autoencoder trained with only synthetic datasets to perform well also on real datasets. Table 2 reports the results obtained for the selected autoencoders, which are denoted as AE_{S1} and AE_{S2}. In the table, the values of the hyperparameters (column **Hyperpar.**), that is, the number of neurons in each fully connected layer, and the LD (column **Latent Dim.**), that is, the dimension of the produced spatial embedding, are reported together with the final values of the loss and val loss functions (columns **LOSS** and **VAL LOSS**). For the accuracy evaluation we use the standard *WMAPE metric*, namely the Weighted Mean Absolute Percentage Error, since it allows us to correctly treat zeros in the set of actual and predicted values. It is a variant of MAPE in which the mean absolute percentage error is treated as a weighed arithmetic mean. Most commonly the absolute percent errors are weighted by the actual values, which leads to the following formula:

$$WMAPE = \frac{\sum_{i=1}^{n}\left(w_i \cdot \frac{|A_i - P_i|}{|A_i|}\right)}{\sum_{i=1}^{n} w_i} = \frac{\sum_{i=1}^{n}\left(|A_i| \cdot \frac{|A_i - P_i|}{|A_i|}\right)}{\sum_{i=1}^{n} |A_i|} = \frac{\sum_{i=1}^{n} |A_i - P_i|,}{\sum_{i=1}^{n} |A_i|}$$

where A_i are the actual value, while P_i are the predicted value.

More specifically, we compute the WMAPE error on both: (a) a synthetic test set obtained by selecting a 20% of the automatically generated synthetic datasets, that the model has not seen before, and (b) an additional set of real data points produced by considering 194 real datasets, from TIGER and OSM sources. With reference to Table 2 (and Table 10 of Appendix A), the values in column **WMAPE** are those computed by performing the test on only synthetic datasets, while the values in column **WMAPE REAL** are those computed considering the collection of real datasets during the test.

Notice that, the metric WMAPE for fully connected models (stacked autoencoders) is good, in the best case it is around 0.36, but these dense networks are not able to generalize well to real cases. Indeed, when applied to real datasets they produce a WMAPE above 50. More details about the other tested but not selected configurations can be found in Table 10 of Appendix A.

Table 3. Experiment Exp^2_{M1} Performed by using **Convolutional Autoencoders**, which have been **Trained with Only Synthetic Datasets** and Tested on Both Synthetic and Real Datasets

Autoencoder	Latent Dim.	Hyperpar.	Training time(sec)	LOSS	VAL LOSS	WMAPE	WMAPE REAL
AE_{C1}	768	filter(128,64)	121	1.3E-03	1.3E-03	0.352	1.28
AE_{C2}	3,072	filter(64,32)	80	9.8E-04	9.9E-04	0.319	1.48

The spatial embeddings have been extracted from *histograms of* $128 \times 128 \times 6$ and the training has been performed with 50 epoches.

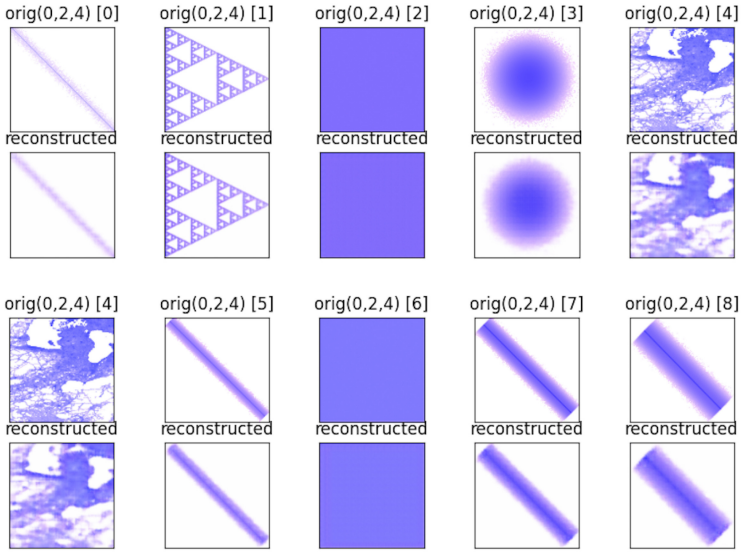


Fig. 4. Application of the encoding-decoding process performed by a **convolutional autoencoder** with a LD of 1,536, 3 layers and filter (64,32,3), that has been **trained with only synthetic data** and tested on both some synthetic and real datasets.

In the second set of experiments, denoted as Exp^2_{M1}, we perform the training considering again only synthetic datasets, but by using convolutional autoencoders, in place of stacked ones. As shown in Table 3 (and Table 11 of Appendix A), in this case the models not only produce a similar quality during the reconstruction of synthetic datasets (i.e., WMAPE is around 0.32 in the best case), but they are also able to generalized better obtaining, with real datasets, a WMAPE around 1.5 on average (see column **WMAPE REAL**). Figure 4 shows an example of application of the encoding-decoding process performed by a convolutional autoencoder trained with only synthetic datasets and tested on both synthetic and real datasets. In particular, for each group the first row shows the original histogram, while the second row reports the decoded one. We can notice that the distribution of the values is maintained in the decoded histograms. To obtain colored images, we convert feature 0, 2, and 4 in RBG values, similar images (but with different colors) can be obtained by considering the other three features.

We also consider a third and fourth set of experiments, denoted as Exp^3_{M1} and Exp^4_{M1}, with the aim to evaluate the impact of including also some real datasets in the training of the autoencoders. Table 4 reports the results obtained with the two autoencorders selected from each set of experiments. In particular, Exp^3_{M1} and autoencoders AE_{S_*} refer to a stacked autoencoder trained with both synthetic and real data, while Exp^4_{M1} and autoencoders AE_{C_*} are related to a convolutional

Table 4. Experiments Exp^3_{M1} and Exp^4_{M1} Performed by using **Stacked** and **Convolutional Autoencoders**, Respectively, which have been **Trained with Both Synthetic and Real Datasets**

Experiment	Autoencoder	Latent Dim.	Hyperpar.	Training time(sec)	LOSS	VAL LOSS	WMAPE
Exp^3_{M1}	AE_{S3}	48	16,32	53	1.7E-03	2.8E-03	2.84
Exp^3_{M1}	AE_{S4}	384	16,32	51	1.6E-03	2.6E-03	2.42
Exp^4_{M1}	AE_{C3}	1,536	filter(128,64)	143	7.1E-04	7.0E-04	0.51
Exp^4_{M1}	AE_{C4}	768	filter(64,32)	98	1.1E-03	1.1E-03	0.54

Models with subscript S_* are stacked autoencoders, while models with subscript C_* are convolutional autoencoders. Spatial embeddings have been extracted from *histograms of* $128 \times 128 \times 6$ and the training has been performed with 50 epochs.

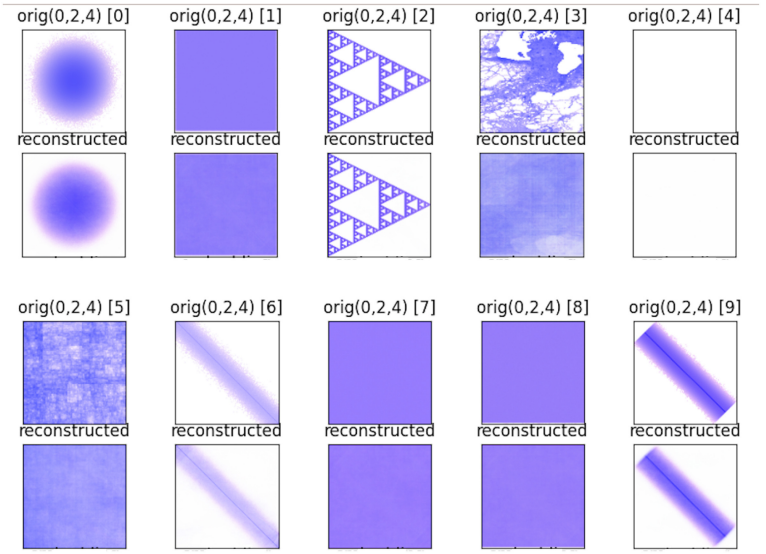


Fig. 5. Application of the encoding-decoding process performed by **stacked autoencoder** with a LD of 384, 2 layers and output (512,256), which have been **trained and tested with both synthetic and real datasets**.

autoencoder trained with both synthetic and real datasets. Detailed results are reported in Tables 12 and 13 in Appendix A for stacked and convolutional autoencoders, respectively. In this case column **WMAPE REAL** is not used, because real datasets are used during both the training and the test phase. Moreover, we can notice that convolutional models are able to produce better results also in this case, with WMAPE values similar to the ones obtained when only synthetic datasets are considered during training and test (see column **WMAPE** in Table 3).

Figure 5 illustrates some example of encodings and decodings performed by a stacked autoencorder trained and tested with both synthetic and real datasets. These cases exemplify the fact that such dense model performs well with syntethic datasets, but their reconstructions are worsen that the one in Figure 4 when real datasets are considered (see cases 4 and 6). Finally, Figure 6 specifically compares the ability of a stacked and a convolutional autoencorder in working with real datasets. In particular, we use them with two real datasets representing roads and municipalities of Italy. Columns (a) and (b) are produced by a dense model, while columns (c) and (d) are obtained with a CNN model. We can notice that the convolutional autoencoder produces a significant improvement in the reconstruction of the original histograms. However, since we are not specifically interested in the reconstruction capabilities of the autoencoders, but in their ability

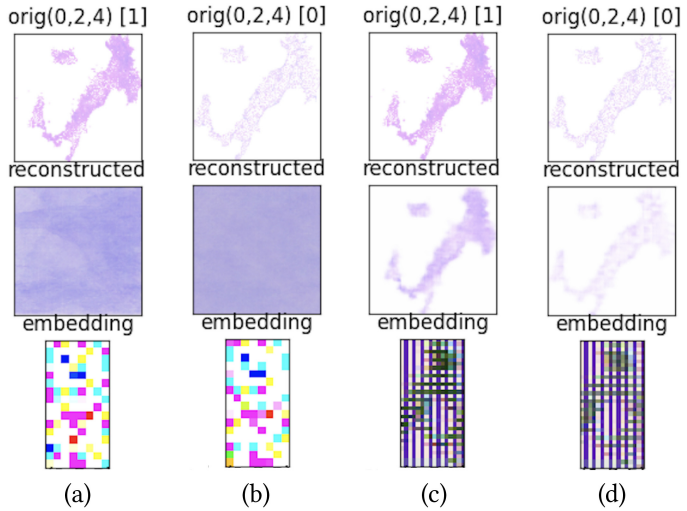


Fig. 6. Application of the encoding-decoding process to 2 real datasets representing roads and municipalities of Italy not used in model training. (a) and (b) are produced by a stacked autoencoder, while (c) and (d) by a convolutional autoencoder.

to properly distill the characteristics of a spatial datasets, we decide to not discard the stacked autoencoders entirely. In particular, given this first session of experiments, we select all the eight models in Tables 2–4 for the generation of the embeddings describing the input datasets:

— Four stacked autoencoders: AE_{S1} and AE_{S2} (trained with only synthetic data and a LD of 384 and 1,536, respectively), AE_{S3} and AE_{S4} (trained with also real data and a LD of 48 and 384, respectively);
— Four convolutional autoencoders: AE_{C1} and AE_{C2} (trained with only synthetic data and a LD of 768 and 3,072, respectively), AE_{C3} and AE_{C4} (trained with also real data and a LD of 1,536 and 768, respectively).

In the second part of the experiments, which are discussed in the following subsections, we focus on the definition of specific models M_2 for estimating the chosen cost parameters for range queries, self-join and binary spatial join.

5.2 Range Query

For the production of the input data points, we consider a set of about 2,950 datasets with various distributions and various sizes, where about 380 are real datasets, as reported in Table 1. As regards to synthetic datasets, in order to simulate their various placements inside the reference space and the fact that they occupy a small area inside it, we randomly place them, with an MBR of at most $(0, 0, 1, 1)$, inside a reference space with MBR $(0, 0, 10, 10)$. Conversely, real datasets were extracted from TIGER and OSM collections by splitting them to generate smaller subsets. For each selected synthetic and real datasets, we randomly produce 50 range queries for synthetic datasets and 100 range queries for real datasets, with different extents and positions. Therefore, we executed about 100,000 range queries Q_i on synthetic datasets, and about 36,000 on real datasets, collecting for each one the selectivity (i.e., $\sigma(Q_i)$). The query windows, generated in the reference space $(0, 0, 10, 10)$, have an area between 1.8×10^{-11} and 7.3×10^{2}. The input data points built in this way are finally normalized by using the $min - max$ formula in Equation (1).

Fig. 7. Comparison among $M2$ models trained with spatial embeddings or directly with histograms. In (a) the WMAPE error for the two cases is shown as the cardinality of the training set is increased, while (b) compares the size in MByte of the input sets needed for the two cases for obtaining the same WMAPE values.

Before using this set of collected data points for training model $M2$, we extract from it a subset of cases with the aim to obtain a *balanced training set*. More specifically, we use an *undersampling* strategy through which we select around 64,000 data points for synthetic datasets from the original 100,000, so that all intervals of possible selectivity values between 0 and 1 are almost equally represented. In the balancing procedure, we subdivide the interval of possible values $[0, 1]$ into ten sub-intervals: $[0, 0.1], \ldots [0.9, 1]$. In the following subsections, the balanced set of synthetic data points for selectivity is denoted as RQ_{σ}^{bal}. We try to produce a balanced set of data points also for real datasets following the same approach. This balanced set of real data points for range query selectivity is denoted as RQR_{σ}^{bal}.

5.2.1 Histograms vs Embeddings. This first experiment has the aim to demonstrate the benefits of using spatial embeddings in terms of required size of the training set. In particular, we show that by using spatial embeddings as input for a prediction model M, instead of the original histograms, we can reach the same accuracy with less training data points and with data of smaller size.

To perform this experiment we prepare two series of data point collections for estimating the range query selectivity by following the approach described in Equation (5). In particular, the first series uses the embeddings produced by autoencoder AE_{C2}, which have the biggest LD, as component $emb(D)$ of the tuple shown in Equation (5). Conversely, the second one substitutes this component by using directly the whole dataset histograms of size $128 \times 128 \times 6$; thus, increasing the size of each data point. Each series of input collections has an increasing number of data points, from 485 to 15,474.

Figure 7(a) compares the WMAPE errors obtained in the two cases (i.e., training with spatial embeddings or with histograms) as the dimension of the data point collection increases. The figure shows that using the spatial embeddings as input always produces an improvement, since the WMAPE decreases of 24% in average. Figure 7(b) shows that in order to obtain a similar accuracy of the model, in terms of WMAPE, the network fed by histograms needs a training set that is an order of magnitude bigger than the training set composed of the corresponding embeddings. This is reflected also in the time needed to execute the training which is for the first configuration in Figure 7(b) about 400 seconds for the training with embeddings and 3,200 seconds for the training with histograms. These results confirm the potentialities of using spatial embeddings in place of the original histograms. Therefore, the following experiments focus only on models fed with spatial embeddings.

Table 5. M2 for **Range Query Selectivity** when it is Trained with Only Synthetic (64,000 Data Points) or Both Real and Synthetic Data (32,000 + 32,000 Data Points) and in Combination with the Spatial Embeddings Presented in Table 4

M2 architecture	M2 Training	Autoencoder	Hyperpar.	Time (sec)	WMAPE	Baseline
$M2_{dnn}$	Synth.	AE_{S1}	dH3	2,007	0.0782	0.691
$M2_{cnn}$	Synth.	AE_{C2}	cH4	1,835	0.0894	
$M2_{dnn}$	Synth. + Real	AE_{S3}	dH2	1,676	0.2143	1.320
$M2_{cnn}$	Synth. + Real	AE_{S4}	cH4	2,700	0.2007	

Hyperparameters for $M2_{dnn}$ are dH2 = 128,64,64,32,32, and dH3 = 256,128,128,64,64 while for $M2_{cnn}$ are cH4 = 512,256,256,128. **Time** is the amount of time needed for training in seconds.

5.2.2 Selectivity of Range Queries. The second experiment regards the prediction of the range query selectivity. As reported in Equation (5), each training data point will contain the information about a given dataset D and the extent of the query window w. More specifically, each dataset D is described in terms of its spatial embedding and the coordinates of its MBR. In Section 5.1, we selected 8 autoencoders (i.e., AE_i) as candidate models for producing spatial embeddings. Starting from them and from the set RQ_{σ}^{bal} previously introduced, we generate 8 different collections of data points $AE_i(RQ_{\sigma}^{bal})$, each one characterized by a different size according to the LD of the embeddings generated by AE_i. The size of the obtained input datasets varies from about 0.044 Gb for a LD of 48, to 2.31 Gb for a LD of 3,072.

Referring back to Equation (5), the input of $M2$ is composed of two parts: (a) the embedding of D and (b) 8 values representing the MBR of D and of the query window w. Given such structure, we consider two alternative approaches for the architecture of model $M2$:

— In the first one (called $M2_{dnn}$) the input (a) is processed by a dense (DNN) model composed of three fully connected layers, then the obtained result is concatenated with input (b) and two additional fully connected layers produce the final estimate.
— Conversely, in the second architecture ($M2_{cnn}$) input (a) is processed instead by two convolutional (CNN) layers and the result is concatenated with (b) and given to the same final two dense layers.

In both cases, we tested different configurations of the hyperparameters in combinations with the 8 sets of data points mentioned above. Moreover, each model has been initially trained and tested with only synthetic data (i.e., $AE_i(RQ_{\sigma}^{bal})$) and then with both real and synthetic data. In this regard, a new collection of data points is obtained by joining $AE_i(RQ_{\sigma}^{bal})$ and $AE_i(RQR_{\sigma}^{bal})$; in particular, we chose 32, 000 points from the first set and an equal amount from the second set.

Table 5 reports for each choice of $M2$ architecture, namely $M2_{dnn}$ and $M2_{cnn}$, the best configuration of hyperparameters (column **Hyperpar.**) together with the autoencoder (column **Autoencoder**) used for the generation of the spatial embeddings (see Tables 2–4). Column **M2 Training** distinguishes the case in which $M2$ is trained and tested with only synthetic data (set $AE_i(RQ_{\sigma}^{bal})$, containing around 64,000 data points) or with both synthetic and real data (about 64,000 data points where an half comes from $AE_i(RQ_{\sigma}^{bal})$ and the other from $AE_i(RQR_{\sigma}^{bal})$). Column **Baseline** reports the error of the BL using WMAPE metrics. As BL, we use the theoretical formula proposed in [5] for estimating the selectivity of range query, that has been applied to the test set. Notice that all experiments use the same test set. Detailed results are reported in Table 14 and Table 15 of Appendix A for the case $M2_{dnn}$ and $M2_{cnn}$, respectively.

Considering the case in which the model is trained and tested with synthetic data only, the best results are obtained with the architecture $M2_{dnn}$ and the autoencoder AE_{S1}, with a WMAPE equal to 0.0782. As you can notice, in any case the proposed model performs better than the BL method which produces on the corresponding test set a WMAPE of 0.691. Conversely, when also real data

are used both in training and testing, the best predictions are obtained with the architecture $M2_{cnn}$ with embeddings generated by autoencoder AE_{S4} producing a WMAPE of 0.2007 when executed on the test set.

The results obtained with the range query are used also for discarding some model combinations to be used in the following experiments regarding the spatial join. In particular, based on the complete set of experiments reported in Table 15 of Appendix A, we selected a stacked and a convolutional autoencoder for training and testing on synthetic data, which are, respectively, AE_{S1} and AE_{C2}, and a stacked and convolutional autoencoder for training and testing on both synthetic and real data, namely AE_{S4} and AE_{C3}. These are the best embeddings for the estimation of the range query selectivity in each respective category.

5.3 Spatial Self-join

The self-join operation runs on a single dataset. Given a dataset D, it finds all pairs of distinct records that have overlapping geometries. To perform this operation in Spark, we first partition the dataset using the R*-Grove partitioner [35] while replicating boundary objects to all overlapping partitions. Then, we process each partition independently with a plane-sweep self-join algorithm to find overlapping records and remove duplicates.

To perform the experiments related to the two parameters of the self join, namely the selectivity and the number of MBR tests, we consider the synthetic and real datasets reported in Table 1 and for each of them we perform the join with itself, collecting both parameters of interest. As done for the range query operation, before using these data for training $M2$, we need to extract a subset of cases which represent a balanced training set. For this purpose we use again an undersampling technique which produces four sets of data points: $S\bar{J}N_{\sigma}^{bal}$, $S\bar{J}NR_{\sigma}^{bal}$, $S\bar{J}N_{\#MT}^{bal}$, and $S\bar{J}NR_{\#MT}^{bal}$. In particular, $S\bar{J}N_{\sigma}^{bal}$ contains about 4,190 data points describing the selectivity of the self-join performed on a selection of the synthetic datasts, while $S\bar{J}N_{\sigma}^{bal}$ contains about 4,784 elements and is obtained by adding to the previous set the data regarding the selectivity computed on the real datasets. Similarly, $S\bar{J}N_{\#MT}^{bal}$ and $S\bar{J}NR_{\#MT}^{bal}$ have the same cardinality and structure of the previous one, but collect as parameter the number of MBR tests performed during the self-join execution. As described at the end of the previous section, for the experiments on self-join, we do not use all autoencoders considered for the range query, but we use the results obtained in these previous experiments to select the best ones to apply also for the estimation of the self-join parameters. In particular, we consider the following 4 autoencoders: AE_{S1}, AE_{C2} and AE_{S4} and AE_{C3}, the first two are used for training and testing $M2$ only on synthetic data, while the last two are used for training and testing $M2$ on both synthetic and real data. Indeed, given the four collections of data points: $S\bar{J}N_{\sigma}^{bal}$, $S\bar{J}NR_{\sigma}^{bal}$, $S\bar{J}N_{\#MT}^{bal}$, and $S\bar{J}NR_{\#MT}^{bal}$, we need to apply on them the chosen autoencoders to produce the embeddings needed as input of $M2$. This produces a collection of 8 data points, the first 4 for synthetic data: $AE_{S1}(S\bar{J}N_{\sigma}^{bal})$, $AE_{S1}(S\bar{J}N_{\#MT}^{bal})$, $AE_{C2}(S\bar{J}N_{\sigma}^{bal})$, $AE_{C2}(S\bar{J}N_{\#MT}^{bal})$, and the other 4 for real data: $AE_{S4}(S\bar{J}NR_{\sigma}^{bal})$, $AE_{S4}(S\bar{J}NR_{\#MT}^{bal})$, $AE_{C3}(S\bar{J}NR_{\sigma}^{bal})$, and $AE_{C3}(S\bar{J}NR_{\#MT}^{bal})$.

Table 6 reports the best configurations for the selectivity estimation, while Table 7 reports the best configurations for the estimation of the number of MBR tests. Detailed results about all the other considered configurations are contained in Tables 16 and 17 of Appendix A. Notice that since the number of MBR tests has been divided by $|D \times D|$, this metric is always less than one. In the experiments we use as the BL devised in [7] which estimates the cost of the **distributed join** (**DJ**) algorithm. To apply it for self join, we use the dataset statistics for the two inputs in the formula.

The obtained results demonstrate that the selected autoencoders perform quite well with the estimation of both parameters considered for the self-join. In particular, convolutional autoencoders seem to better capture the characteristics of the datasets when both synthetic data only, or

Table 6. M2 for **Self-join Selectivity** when it is Trained with Only Synthetic (4,194 Data Points) or Both Real and Synthetic Data (4,784 Data Points) and in Combination with the Spatial Embeddings in Table 4

M2	M2 Training	Emb.	Hyperp.	Time (sec)	WMAPE	BL
$M2_{dnn}$	Synth.	AE_{C2}	dH3	108	0.2138	0.69
$M2_{cnn}$	Synth.	AE_{C2}	cH4	128	0.2211	
$M2_{dnn}$	Synth. + Real	AE_{S4}	dH1	160	0.3557	2.0
$M2_{cnn}$	Synth. + Real	AE_{C3}	cH5	233	0.3010	

Hyperparameters for $M2_{dnn}$ are dH1 = 64,32,32,16,16, and dH3 = 256,128,128,64,64, while for $M2_{cnn}$ are cH4 = 512,256,256,128, cH5 = 1024,512,512,256. **Time** is the amount of time needed for training in seconds.

Table 7. M2 for the **Number of MBR Tests in Self Join** when it is Trained with Only Synthetic or Both Real and Synthetic Data and in Combination with the Spatial Embeddings in Table 4

M2	M2 Training	Emb.	Hyperp.	Time (sec)	WMAPE	BL
$M2_{dnn}$	Synth.	AE_{C2}	dH4	139	0.3128	0.96
$M2_{cnn}$	Synth.	AE_{C2}	cH4	182	0.3232	
$M2_{dnn}$	Synth. + Real	AE_{S4}	dH1	134	0.4452	0.96
$M2_{cnn}$	Synth. + Real	AE_{C3}	cH5	174	0.3001	

Hyperparameters for $M2_{dnn}$ are dH1 = 64,32,32,16,16, and dH4 = 512,256,256,128,128, while for $M2_{cnn}$ are cH4 = 512,256,256,128, and cH5 = 1024,512,512,256. **Time** is the amount of time needed for training in seconds.

synthetic and real datasets together are considered. Relatively to the selectivity of self-join, encoder AE_{C2} provides essentially the same accuracy with both kinds of $M2$ architectures when only synthetic data is considered, while in presence of both synthetic and real datasets the best encoder is again the convolutional one, namely AE_{C3}. Similar results are obtained also with the tests regarding the estimation of the number of MBR tests, where again the best encoder for synthetic data is AE_{C2}, while when real and synthetic data are considered the best performances are obtained with AE_{C3} with a convolutional $M2$ model. The obtained results outperform the ones obtained with the BL in terms of the achieved accuracy, and also the performances. The obtained WMAPE errors are in line with what has been obtained for the estimation of the range query selectivity when synthetic and real data are considered together, confirming that spatial embeddings provides promising generalization capabilities. Notice that the BL we used performs equally well for synthetic and real data since the partitioning step captures data skewness and balances the cost across partitions which makes the cost estimation easier.

5.4 Binary Spatial Join

Given a collection of n input datasets, the amount of spatial join operations that need to be executed for producing a training set covering all possible cases is equal to $(n \times (n-1))/2$. This can lead to a very large number of costly operations to be performed. Therefore, we apply a clustering technique, exploiting also the presence of the computed embeddings, in order to select the most significant pairs on which actually perform the join operation. In particular, starting from about 2,550 synthetic datasets, we select about 11,000 significant pairs, while starting from about 380 real datasets, we generate a subset of around 2,000 pairs.

Given these pairs of datasets (D_i, D_j), we compute the spatial join between D_i and D_j by applying two consolidated implementations of this operation provided by the Beast library on Spark:

Table 8. M2 for **Spatial Join Selectivity** when it is Trained with Only Synthetic (11,000 Data Points) or Both Real and Synthetic Data (13,000 Data Points) and in Combination with the Spatial Embeddings in Table 4

M2	M2 Training	Emb.	Hyperp.	Time (sec)	WMAPE	BL
$M2_{dnn}$	Synth.	AE_{C2}	dH5	431	0.2251	0.7274
$M2_{cnn}$	Synth.	AE_{C2}	cH4	550	0.2810	
$M2_{dnn}$	Synth. + Real	AE_{S4}	dH1	131	0.2636	0.8420
$M2_{cnn}$	Synth. + Real	AE_{C3}	cH5	175	0.2125	

Hyperparameters for $M2_{dnn}$ are dH1 = 64,32,32,16,16, and dH5 = 1024,512,512,256,256, while for $M2_{cnn}$ are cH4 = 512,256,256,128, cH5 = 1024,512,512,256. **Time** is the amount of time needed for training in seconds.

the **Spatial Join Map Reduce** (**SJMR**) [40] and the DJ with Index [11, 38]. The first one is a MapReduce implementation of the **Partition Based Spatial Merge** (**PBSM**) Join [29] and has been designed to efficiently perform a spatial join on non-indexed datasets. The second one is a MapReduce adaptation of the Grid File Spatial Join algorithm [19] and works on previously indexed datasets. The execution of these 11,000 join operations on synthetic datasets and 2,000 join operations on real datasets tooks almost one month for each implementation.

As regards to the selectivity estimation, since this metric does not depend on the algorithm, we use the results of the SJRM execution to produce both a set of 11,000 synthetic data points, denoted as $\mathcal{JN}_{\sigma}^{bal}$, and a set of 2,000 real data points, denoted as $\mathcal{JN}_{\sigma}^{bal}$. Conversely, for the number of MBR tests depends on the selected algorithm. This means that we also generates the following sets of data points: $S\mathcal{J}_{\#MT}^{bal}$ and $S\mathcal{JR}_{\#MT}^{bal}$ for SJMR and $D\mathcal{J}_{\#MT}^{bal}$ and $D\mathcal{JR}_{\#MT}^{bal}$ for DJ. These sets have the same cardinality and regards the same pairs of $\mathcal{JN}_{\sigma}^{bal}$ and $\mathcal{JN}_{\sigma}^{bal}$, but they collect as parameter the number of MBR tests performed by SJMR and DJ algorithms, respectively.

As described done in the previous section, for the experiments on the binary spatial join, we consider the following 4 autoencoders: AE_{S1}, AE_{C2} and AE_{S4} and AE_{C3}, the first two are used for training and testing $M2$ only on synthetic data, while the last two are used for training and testing $M2$ on both synthetic and real data. Indeed, given the six collections of data points: $\mathcal{JN}_{\sigma}^{bal}$, $\mathcal{JN}_{\sigma}^{bal}$, $S\mathcal{J}_{\#MT}^{bal}$, $S\mathcal{JR}_{\#MT}^{bal}$, $D\mathcal{J}_{\#MT}^{bal}$, $D\mathcal{JR}_{\#MT}^{bal}$, we need to apply on them the chosen autoencoder to produce the embeddings needed as input of $M2$. This produced 12 collections of data points, the first 6 for synthetic data: $AE_{S1}(\mathcal{JN}_{\sigma}^{bal})$, $AE_{S1}(S\mathcal{J}_{\#MT}^{bal})$, $AE_{S1}(D\mathcal{J}_{\#MT}^{bal})$, $AE_{C2}(\mathcal{JN}_{\sigma}^{bal})$, $AE_{C2}(S\mathcal{J}_{\#MT}^{bal})$, and $AE_{C2}(D\mathcal{J}_{\#MT}^{bal})$, and othe other 6 for real data: $AE_{S4}(\mathcal{JN}_{\sigma}^{bal})$, $AE_{S4}(S\mathcal{JR}_{\#MT}^{bal})$, $AE_{S4}(D\mathcal{JR}_{\#MT}^{bal})$, $AE_{C3}(\mathcal{JN}_{\sigma}^{bal})$, $AE_{C3}(S\mathcal{JR}_{\#MT}^{bal})$, and $AE_{C3}(D\mathcal{JR}_{\#MT}^{bal})$ These are the sets of data points used in the experiments illustrated in Tables 8, 9, and 18–20 of Appendix A.

In the experiments we use as the BL the model proposed in [34]. Notice that this model has been trained again with the new datasets $\mathcal{JN}_{\sigma}^{bal}$, $\mathcal{JN}_{\sigma}^{bal}$, $S\mathcal{J}_{\#MT}^{bal}$, $S\mathcal{JR}_{\#MT}^{bal}$, $D\mathcal{J}_{\#MT}^{bal}$, and $D\mathcal{JR}_{\#MT}^{bal}$, which are much richer and diverse than the dataset used in the original article. We found that with these new datasets, the performance started to drop due to the more complex data distributions, for example, Sierpinski distribution and real datasets. Table 8 reports the best configurations for the selectivity, while Table 9 reports the best configurations for the number of MBR tests (notice that, since this metric is divided by $|D_1 \times D_2|$, it is always less than one). Detailed results can be found in Table 18 of Appendix A for experiments with selectivity, and in Tables 19 and 20 of Appendix A for experiments with the number of MBR tests int the two implementations.

We can notice that the selected autoencoders perform quite well with the estimation of both parameters and in presence of only synthetic, or both synthetic and real data together. In particular, convolutional autoencoders seam to better capture the characteristics of both parameters and allow to obtain smaller errors when they are used in combination of both kinds of $M2$ models. With

Table 9. M2 for the **Number of MBR Tests in Spatial Join** when it is Trained with Only Synthetic or Both Real and Synthetic Data and in Combination with the Spatial Embeddings in Table 4

Algorithm	M2	M2 Training	Emb.	Hyperp.	Time (sec)	WMAPE	BL
SJMR	$M2_{dnn}$	Synth.	AE_{C2}	dH4	424	0.3401	0.811
	$M2_{cnn}$	Synth.	AE_{C2}	cH3	475	0.3907	
DJ	$M2_{dnn}$	Synth.	AE_{C2}	dH5	107	0.2631	0.37
	$M2_{cnn}$	Synth.	AE_{C2}	cH3	144	0.2846	
SJMR	$M2_{dnn}$	Synth. + Real	AE_{S4}	dH3	140	0.3379	1.008
	$M2_{cnn}$	Synth. + Real	AE_{S4}	cH3	166	0.3205	
DJ	$M2_{dnn}$	Synth. + Real	AE_{S4}	dH4	119	0.3344	0.39
	$M2_{cnn}$	Synth. + Real	AE_{S4}	cH5	199	0.3645	

Hyperparameters for $M2_{dnn}$ are dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, and dH5 = 1024,512,512,256,256, while for $M2_{cnn}$ are cH3 = 256,128,128,64, and cH5 = 1024,512,512,256. **Time** is the amount of time needed for training in seconds.

reference to selectivity of spatial joins, the use of autoencoders AE_{C2} and AE_{C3} allows to obtain the best results with synthetic data only, or with synthetic and real data together, respectively. In particular, the error of this last more complicated case (i.e., 0.2125) is even smaller than the best one obtained in the simpler synthetic case (i.e., 0.2251). A similar behaviour of convolutional autoencoders can be observed also in the estimation of the number of MBR tests where smaller errors are obtained with AE_{C2} on synthetic data. However, for the more complicated case where also real data are used AE_{S4} performs slighltly better than the convolutional autoencoders, obtaining an error of 0.3205 for SJRM and 0.3344 for DJ. The obtained results outperforms the ones obtained with the BL in terms of the achieved accuracy, and as mentioned before, in terms of performances. The obtained WMAPE errors are in line with what has been obtained for the estimation of the range query selectivity when synthetic and real data are considered together, and with self-join parameters in all the cases. Moreover, the best identified autoencorders for the two parameters of the binary spatial join are the same of those identified for the self-join in the previous section. All these considerations confirm the generalization capabilities of spatial embeddings.

5.5 Discussion on Experiment Results

Given the experimental results that we obtained in the previous subsections for both operations, range query and spatial join, we can observe that:

(1) Spatial embeddings, generated by the trained autoencoders, are able to distill the information useful for training successive models which predict parameters for cost evaluation of spatial operations. Indeed, they behave always better than the considered BLs. Moreover, as shown in Figure 7, the size of the training set that is needed to obtain a certain accuracy (misured as the inverse of the WMAPE error) is considerably reduced by the introduction of the embeddings with respect to the one required when histograms are directly used as model inputs.
(2) The autoencoder that in average produces the best results is AE_{C2}, which is produced by a convolutional autoencoder and is the one having the largest LD (3,072). However, also smaller embeddings, like AE_{S1} or AE_{S4} can produce good results both for estimating range query and spatial join selectivity.
(3) Considering that we used a limited collection of datasets for training $M1$ models and a larger set of data points for training $M2$ for range query, the training time of the two experiments are not comparable. However, the set of data points used for training $M2$ models for self-join or binary spatial join and $M1$ were similar in size and also the training time of these

models is similar, around 100 sec. Training a unique model for estimating the range query selectivity starting from histograms of datasets requires: (i) a large training set (around 20 Gb for 100,000 range queries) and (ii) almost one day for executing a first try of training, that did not end. Therefore, the separation between $M1$ and $M2$ represents an effective solution for generating in a reasonable time a tool that is able to predict cost parameters.

(4) The idea that $M1$ can be trained only once is confirmed by the fact that the same generated embeddings can be used for estimating two different parameters for three distinct operations, that is, range query, self-join and binary spatial join, as well as two alternative implementations of the latter one, SJMR and DJ.

(5) Finally, even if the time required for training and testing the entire pipeline could be very long, in particular for model $M1$, this does not affect the applicability of the approach in a real-world query optimizer. Indeed, the training and test will be performed offline and only once for $M1$ and once for each operation for $M2$. Conversely, a query optimizer is affected only by the prediction time which takes essentially few milliseconds to execute.

In order to check whether we can further improve the effectiveness of the autoencoders ($M1$) in generating spatial embeddings that can be used by a successive model ($M2$) which predicts a specific cost parameter, we integrate a previously trained autoencorder in a new model $M2$, applying the *transfer learning* approach, as described in the following subsection.

5.6 Spatial Embeddings vs Transfer Learning

This section illustrates a set of the experiments that check the effect of applying a transfer learning approach in combination with the spatial embedding generation, instead of using the proposed two-model-layer architecture. More specifically, given one of the previously trained autoencorders, namely AE_{C2}, we integrate its *encoding layers* into a new CNN model $M2$, denoted as $M2_{cnnTL}$ with the aim to estimate the selectivity of range queries. In particular, model $M2_{cnnTL}$ is trained with new sets of data points with increasing size. Each of these sets, called $HS(RQ_{\sigma}^{bal})_i$, contains the histograms of a subset of RQ_{σ}^{bal}(see Section 5.2) as input. Indeed, in this architecture, spatial embeddings are not used as input of $M2$, but they are produced by the encoding layers of AE_{C2} that have been integrated in $M2_{cnnTL}$. We built the various $HS(RQ_{\sigma}^{bal})_i$ by considering collection of data points of cardinality from 485 to 15,474, with a corresponding size from 0.45 GB to 1.22 GB. Finally, we compare the accuracy obtained by $M2_{cnnTL}$ for each $HS(RQ_{\sigma}^{bal})_i$ with the one obtained by the corresponding model $M2_{cnn}$ that has been trained with the same collection of datasets, but by substituting their histograms with the corresponding spatial embeddings produced by AE_{C2}. Clearly, in this case despite to the cardinality of the input set, the training with spatial embedding substantially reduces the size in bytes, which becomes from 0.01 GB to 0.41 GB, instead from 0.45 GB to 1.22 GB.

The results of the comparison is illustrated in Figure 8 where the accuracy of the two models is shown in terms of WMAPE error with respect to the cardinality of the training set. Notice that, the transfer learning approach has a positive effect only when the cardinality of the training set is lower than $7,000$ data points. Indeed, $M2_{cnnTL}$ performs better than the configuration $M2_{cnn}$+AE_{C2} only for small training sets. Conversely, when a bigger training set can be generated, the proposed two-layer-model approach, that separates the generation of the spatial embeddings from the training of $M2$, produces a better accuracy.

6 CONCLUSION AND FUTURE WORK

In this article, we propose a new approach for exploiting ML and DL techniques in optimization of spatial queries. In particular, the proposed framework aims at developing a generic model for

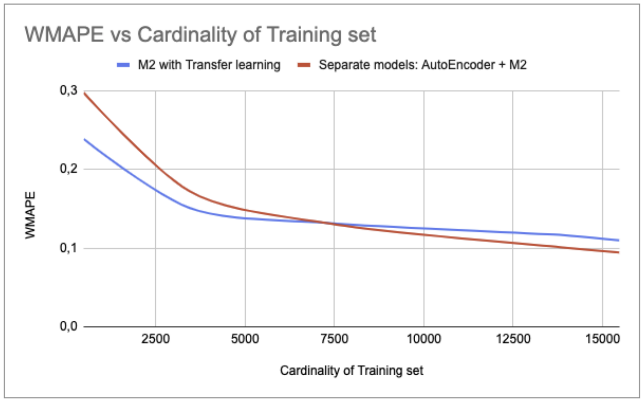


Fig. 8. Comparison between $M2_{cnnTL}$ and AE_{C2}+$M2_{cnn}$ models, the first trained directly with histograms and the second with spatial embeddings. The WMAPE error of both models is reported with respect to the cardinality of the training set.

estimating different cost parameters of several spatial operation implementations. The key idea is to introduce a two level architecture, where a first model, denoted as $M1$, is responsible for generating spatial embeddings of given datasets which are able to distill their relevant features for the following parameter estimation. Model $M1$ needs large amount of training datasets to work properly, but it can be trained only once and this cost can be amortized by the following estimation models. Indeed, the second level of the proposed framework is composed of several models $M2$, one for each spatial operation parameter to be estimated, but since they use as one of their inputs the spatial embeddings, in place of the original multi-faced histograms, the amount and the size of the training set can be drastically reduced.

We demonstrate the this idea can be applied successfully for the estimation of selectivity of range queries and also of selectivity and number of MBR tests of both self-join and binary spatial join. Experiments showed that the predictions of different parameters can be based on the same spatial embeddings and that the obtained accuracy is better than the considered BLs. Moreover, we experimentally show that the use of spatial embeddings drastically reduces the amount and the size of the training set needed to achieve a particular accuracy. Finally, a comparison with another ML technique, called transfer learning, is presented which further highlights the advantages of the proposed framework.

The obtained results are promising in terms of achieved accuracy with respect to the considered BL methods. They encourage the further investigation in this direction and the testing of the proposed framework with other spatial operation implementations and cost parameters. Moreover, since many assumptions that have been made consider only outdoor applications, for instance, relatively to the possible spatial distributions, an interesting future extension would be studying the applicability and extensibility of the proposed methodology also to indoor scenarios.

APPENDIX

A DETAILED EXPERIMENTAL RESULTS

This appendix reports some detailed results about the performed experiments which have not been included in the main article for not cluttering the presentation. Section A.1 is related to the experiments performed about the construction of model $M1$ which produces the spatial embeddings. Section A.2 reports detailed results about the identification of the best model $M2$ which

Table 10. Training of the **Stacked Autoencoder**, with 3 Dense Layers, for Extracting Spatial Embeddings Starting from *Histograms of* $128 \times 128 \times 6$

3 dense layers, epochs = 50, histograms = $128 \times 128 \times 6$						
Hyperp.	**Training time (sec)**	**LOSS**	**VAL LOSS**	**WMAPE**	**WMAPE REAL**	**Selected autoencoder**
latent dimension = 384						
512,256	71	9.5E-04	1.3E-03	0.405	55.71	–
1,024,512	105	9.6E-04	1.5E-03	**0.363**	57.12	AE_{S1}
2,048,1024	165	9.6E-04	1.5E-03	0.400	50.57	–
latent dimension = 768						
512,256	71	9.0E-04	1.3E-03	0.366	56.81	–
1,024,512	103	9.6E-04	1.6E-03	0.370	58.01	–
2,048,1024	163	1.0E-03	1.4E-03	0.370	50.92	–
latent dimension = 1536						
512,256	73	1.2E-03	1.3E-03	0.368	55.85	–
1,024,512	104	1.1E-03	1.6E-03	**0.356**	54.06	AE_{S2}
2,048,1024	164	9.2E-04	1.6E-03	0.423	52.87	–

The training has been performed with 50 epoches and considering only **synthetic datasets**. Column **WMAPE** reports the error obtained by performing the test on synthetic data, while column **WMAPE REAL** contains the error for the test performed on real datasets.

estimates the range query selectivity parameter. Section A.3 illustrates detailed results aboth the $M2$ models for estimating the selectivity and the number of MBR tests for the self-join. Finally, Section A.4 contains detailed results about the models $M2$ built for estimating the selectivity and the number of MBR tests required by the two considered implementations of the binary spatial join, respectively.

A.1 Autoencoders

This section reports the detailed results about the four experiments described in Section 5.1 for the identification of the eight candidate spatial embeddings. In particular, Table 10 reports the detailed results obtained with a stacked autoencoder trained with only synthetic data. The WMAPE is computed on a test set covering the 20% of the data points that the model has not seen before. Moreover, an additional set of data points produced by considering 194 real datasets, from TIGER and OSM sources, have been used as test set producing the WMAPE REAL shown in the 6th column. Similarly, Table 11 reports the detailed results obtained with a convolutional autoencoder trained with only synthetic data and tested on the same synthetic and real datasets considered in the previous case.

Table 12 reports the detailed results obtained with a stacked autoencoder trained with both real and synthetic data. The WMAPE is computed on a test set covering the 20% of the data points that the model has not seen before which includes both synthetic and real datasets, so column **WMAPE REAL** is not necessary in this case. Similarly, Table 13 reports the detailed results obtained with a convolutional autoencoder trained and tested on both real and synthetic data.

A.2 Models Estimating the Selectivity of Range Queries

This section contains the detailed results of the experiments performed to build model $M2$ which estimates the selectivity of range queries. As described in Section 5.2, two different architectures have been tested for this purpose. They differ from the kind of network used to process the input spatial embeddings and are denoted as $M2_{dnn}$ and $M2_{cnn}$, respectively.

Table 11. Training of the **Convolutional Autoencoder**, with 3 CNN Layers, for Extracting Spatial Embeddings Starting from *Histograms of* $128 \times 128 \times 6$

3 CNN layers, epochs = 50, histograms = $128 \times 128 \times 6$						
Hyperp.	**Training time (sec)**	**LOSS**	**VAL LOSS**	**WMAPE**	**WMAPE REAL**	**Selected autoencoder**
latent dimension = 512						
filter(32,16)	78	2.8E-03	3.0E-03	0.501	1.82	–
filter(64,32)	87	2.6E-03	2.4E-03	0.486	1.97	–
filter(128,64)	124	2.9E-03	3.0E-03	0.507	1.59	–
latent dimension = 768						
filter(32,16)	72	2.6E-03	2.8E-03	0.510	1.67	–
filter(64,32)	80	1.6E-03	1.5E-03	0.421	1.94	–
filter(128,64)	121	1.3E-03	1.3E-03	**0.352**	1.28	AE_{C1}
latent dimension = 3072						
filter(32,16)	73	1.4E-03	1.5E-03	0.386	1.42	–
filter(64,32)	80	9.8E-04	9.9E-04	**0.319**	1.48	AE_{C2}
filter(128,64)	121	8.4E-04	9.2E-04	0.326	1.25	–

The training has been performed with 50 epoches and considering only **synthetic datasets**. Column **WMAPE** reports the error obtained by performing the test on synthetic data, while column **WMAPE REAL** contains the error for the test performed on real datasets.

Table 12. Training of the **Stacked Autoencoder**, with 3 Dense Layers, for Extracting Spatial Embeddings Starting from *Histograms of* $128 \times 128 \times 6$

3 dense layers, epochs = 50, histograms = $128 \times 128 \times 6$					
Hyperp.	**Training time (sec)**	**LOSS**	**VAL LOSS**	**WMAPE**	**Selected autoencoder**
latent dimension = 48					
2048,1024	182	8.8E-04	3.2E-03	5.53	–
1024,512	122	8.2E-04	2.6E-03	5.60	–
512,256	84	8.4E-04	3.5E-03	5.72	–
16,32	53	1.7E-03	2.8E-03	**2.84**	AE_{S3}
latent dimension = 192					
2048,1024	182	1.3E-03	3.9E-03	6.52	–
1024,512	123	8.4E-04	3.3E-03	5.12	–
512,256	81	9.6E-04	3.8E-03	5.64	–
16,32	52	1.8E-03	3.3E-03	3.59	–
latent dimension = 384					
2048,1024	184	1.3E-03	3.0E-03	5.54	–
1024,512	122	8.4E-04	2.8E-03	5.28	–
512,256	82	9.3E-04	3.5E-03	6.38	–
16,32	51	1.6E-03	2.6E-03	**2.42**	AE_{S4}

The training has been performed with 50 epochs and considering both **real and synthetic datasets**.

Table 14 reports the results of the application of $M2_{dnn}$ in combination with various input sets containing spatial embeddings generated by different autoencoders. In particular, $M2$ is trained and tested on: (a) only synthetic data points, in this case a set of 64,000 data points is considered; or (b) on both real and synthetic data points, in this case 32,000 data points for synthetic data

Table 13. Training of the **Convolutional Autoencoder**, with 3 CNN Layers, for Extracting Spatial Embeddings Starting from *Histograms of* $128 \times 128 \times 6$

3 dense layers, epochs = 50, histograms = $128 \times 128 \times 6$					
Hyperp.	**Training time (sec)**	**LOSS**	**VAL LOSS**	**WMAPE**	**Selected autoencoder**
latent dimension = 1536					
filter(32,16)	93	9.8E-04	9.1E-04	0.59	–
filter(64,32)	100	7.7E-04	8.2E-04	0.59	–
filter(128,64)	143	7.1E-04	7.0E-04	**0.51**	AE_{C3}
latent dimension = 768					
filter(32,16)	89	1.4E-03	1.6E-03	0.60	–
filter(64,32)	98	1.1E-03	1.1E-03	**0.54**	AE_{C4}
filter(128,64)	144	9.3E-04	9.8E-04	0.55	–
latent dimension = 512					
filter(32,16)	88	1.4E-03	1.4E-03	0.60	–
filter(64,32)	95	1.3E-03	1.3E-03	0.63	–
filter(128,64)	144	1.1E-03	1.1E-03	0.64	–

The training has been performed with 50 epochs and considering both **real and synthetic datasets**.

Table 14. M2 for Estimation of **Selectivity** of Range Queries with a Model of Type **DNN**

Hyperpar.	**Embedding**								**Baseline**
M2 trained and tested on synthetic data									
	AE_{S1} $LD = 384$		AE_{S2} $LD = 1536$		AE_{C1} $LD = 768$		AE_{C2} $LD = 3072$		Theoretical formula [4]
	WMAPE	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	
dH1	0.1701	1,569	0.2038	1,317	0.2001	1,150	0.2268	750	
dH2	0.1120	2,146	0.1268	1,140	0.3409	1,285	0.1088	1,231	
dH3	**0.0782**	2,007	0.1103	989	0.2818	560	0.1554	890	0.691
dH4	0.0802	2,145	0.1225	1,360	0.2361	736	**0.0934**	989	
dH5	0.0818	2,114	0.1231	1,077	0.2335	647	0.1312	507	
M2 trained and tested on synthetic and real data									
	AE_{S3} $LD = 48$		AE_{S4} $LD = 384$		AE_{C3} $LD = 1536$		AE_{C4} $LD = 768$		Theoretical formula [4]
	WMAPE	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	
dH1	0.2367	1,550	0.3208	1,355	0.3449	1,172	0.3306	407	
dH2	**0.2143**	1,176	0.2968	682	0.3463	651	0.3423	954	
dH3	0.2718	1,698	0.2449	1,497	**0.2904**	1,044	0.3251	596	1.32
dH4	0.2237	1,290	0.2649	1,164	0.3386	829	0.4291	343	
dH5	0.2274	1,570	0.3013	571	0.3365	441	0.3047	458	

Hyperparameters of column **Hyperpar.** are as follows: dH1 = 64,32,32,16,16, dH2 = 128,64,64,32,32, dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, dH5 = 1024,512,512,256,256.

are randomly selected from the original 64,000 elements, and additional 32,000 real data points are added to this set by randomly selecting them from the 33,554 points obtained from the range queries computed on OSM Lakes and OSM Parks datasets. The results of case (a) are reported in the first five rows of Table 14, while the results of case (b) are contained in the last five rows of the table.

In Table 14, column **Hyperpar.** reports the number or nodes in each layer, while column **Baseline** reports the error of the BL evaluated with the metrics WMAPE. As BL, we use the theoretical formula proposed in [4] for estimating the selectivity of range query. Notice that, when $M2_{dnn}$

Table 15. M2 for Estimation of **Selectivity** of Range Queries with a Model of Type **CNN**

Hyperpar.	Embedding								Baseline
M2 trained and tested on synthetic data									
	AE_{S1} $LD = 384$		AE_{S2} $LD = 1536$		AE_{C1} $LD = 768$		AE_{C2} $LD = 3072$		Theoretical formula [4]
	WMAPE	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	
cH1	0.1621	1,101	0.1646	2,208	0,1836	1,874	0.1206	2,018	
cH2	0.1167	2,021	0.1201	2,108	0.1333	1,877	0.1304	2,650	
cH3	0.0973	1,150	0.1061	2,222	0.1038	1,877	0.0961	2,350	0.691
cH4	0.1062	1,659	**0.0948**	2,309	0.1005	1,742	**0.0894**	1,835	
cH5	0.0859	2,257	0.1001	2,199	0.1038	1,559	0.0907	3,228	
M2 trained and tested on synthetic and real data									
	AE_{S3} $LD = 48$		AE_{S4} $LD = 384$		AE_{C3} $LD = 1536$		AE_{C4} $LD = 768$		Theoretical formula [4]
	WMAPE	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	**WMAPE**	**sec**	
cH1	0.2619	1,400	0.3092	936	0.2747	1,641	0.3096	1,119	
cH2	0.2726	2,412	0.2379	2,513	0.2379	2,531	0.2542	2,366	
cH3	0.2386	2,492	0.2269	2,530	0.2737	2,851	0.2994	1,540	1.32
cH4	0.2071	2,567	**0.2007**	2,700	**0.2332**	2,815	0.2453	707	
cH5	0.2772	1,542	0.2156	2,159	0.3097	2,820	0.2851	2,788	

Hyperparameters of column Hyperpar. are as follows: cH1 = 64,32,32,16, cH2 = 128,64,64,32, cH3 = 256,128,128,64, cH4 = 512,256,256,128, cH5 = 1024,512,512,256.

is tested on synthetic dataset alone, we use only spatial embeddings generated with autoencorder trained with only synthetic data, that is, AE_{S1}, AE_{S2}, AE_{C1}, and AE_{C2}. On the contrary, when $M2_{dnn}$ is tested also on real datasets, then we use the autoencoders trained also with real data, that is, AE_{S3}, AE_{S4}, AE_{C3}, and AE_{C4}. For each considered autoencoder, its LD is also reported.

Similarly, Table 15 reports the results of the application of $M2_{cnn}$ in combination with various input sets containing spatial embeddings generated by different autoencoders. As in the previous case, $M2_{cnn}$ is initially trained and tested on only synthetic data and then on both synthetic and real data. The same considerations made above for the choice of the autoencoder in the various cases hold also here.

A.3 Models Estimating the Selectivity and Number of MBR Tests of Self-join

This section reports the detailed results of the experiments performed to build the two models $M2$ which are used to estimate the selectivity and the number of MBR tests of the self-join, respectively. In this case, only 4 autoencoders are considered, the ones that perform better in the various configurations identified in the previous experiments done for the range query selectivity. In particular, given the two considered architectures described in Section 5.3 and denoted as $M2_{dnn}$ and $M2_{cnn}$, we consider for each of them a stacked and a convolutional autoencoder.

Table 16 report the results of training and testing the two models $M2_{dnn}$ and $M2_{cnn}$ for the selectivity estimation of the self-join with both synthetic datasets only, and synthetic and real datasets together. In particular, the input set of these models is represented by the spatial embeddings generated by the stacked autoencoder AE_{S1} or the convolutional autoencoder AE_{C2} in the first case, and the stacked autoencoder AE_{S4} or the convolutional autoencoder AE_{C3} in the second case . For each of these cases, we report both the error obtained during the estimation of the spatial join selectivity (i.e., column **WMAPE**) and the required training time in seconds. Various hyperparameter configurations are considered and the best obtained results are highlighted in bold.

Finally, Table 17 reports the estimation errors for $M2_{dnn}$ and $M2_{cnn}$ when they are trained and tested on both synthetic data only, and synthetic and real datasets togehter, with the aim to

Table 16. M2 for Estimating the **Self-join Selectivity** for Both Synthetic Datasets Only, and Synthetic and Real Dataset Together

Net. arch.	Hyper par.	Synthetic data					Synthetic and real data				
		AE_{S1}		AE_{C2}		BL	AE_{S4}		AE_{C3}		BL
		WMAPE	Time	WMAPE	Time		WMAPE	Time	WMAPE	Time	
$M2_{dnn}$	dH1	0.3079	135	0.2974	75	0.69	0.4135	154	0.6954	53	2.0
	dH2	0.3261	135	0.2217	93		0.3958	153	0.6738	36	
	dH3	**0.2646**	134	**0.2138**	198		0.4078	154	0.5959	154	
	dH4	0.2789	138	0.2495	141		0.3847	158	0.6061	105	
	dH5	0.2806	141	0.2761	98		**0.3557**	160	**0.5876**	159	
$M2_{cnn}$	cH1	0.3519	139	0.2727	76	0.69	0.4178	194	0.4562	196	2.0
	cH2	0.3144	139	0.2666	138		0.4095	194	0.7788	39	
	cH3	**0.3006**	136	0.2252	124		0.3694	213	0.3345	199	
	cH4	0.3059	139	**0.2211**	128		**0.3397**	216	0.3163	229	
	cH5	0.3096	144	0.2533	101		0.3691	236	**0.3010**	233	

Hyperparameters of column Hyperpar. are for $M2_{dnn}$ are dH1 = 64,32,32,16,16, dH2 = 128,64,64,32,32, dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, dH5 = 1024,512,512,256,256. While for $M2_{cnn}$ are cH1 = 64,32,32,16, cH2 = 128,64,64,32, cH3 = 256,128,128,64, cH4 = 512,256,256,128, cH5 = 1024,512,512,256. Column **Time** reports the training time in seconds.

Table 17. M2 for Estimating the **Number of MBR Tests for the Self-join** for Both Synthetic Datasets Only, and Synthetic with Real Datasets Together

Net. arch.	Hyper par.	Synthetic data					Synthetic and real data				
		AE_{S1}		AE_{C2}		BL	AE_{S4}		AE_{C3}		BL
		WMAPE	Time	WMAPE	Time		WMAPE	Time	WMAPE	Time	
$M2_{dnn}$	dH1	0.4547	109	0.4562	77	0.96	0.4588	133	0.5199	136	0.96
	dH2	**0.3904**	124	0.3908	90		**0.4452**	134	0.7129	30	
	dH3	0.3947	124	0.3399	138		0.4538	133	**0.4795**	119	
	dH4	0.4135	128	**0.3128**	139		0.4946	136	0.6773	124	
	dH5	0.4093	153	0.3231	109		0.5129	140	0.5262	142	
$M2_{cnn}$	cH1	0.4973	138	0.5057	110	0.96	0.5578	110	0.3983	173	0.96
	cH2	0.4097	158	**0.3029**	182		0.4452	169	0.3517	171	
	cH3	0.4136	174	0.3624	147		0.4327	187	**0.3001**	174	
	cH4	0.3971	175	0.3232	184		0.4266	191	0.3403	202	
	cH5	**0.3932**	191	0.3566	182		**0.4255**	207	0.3724	206	

Hyperparameters of column Hyperpar. for $M2_{dnn}$ are dH1 = 64,32,32,16,16, dH2 = 128,64,64,32,32, dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, dH5 = 1024,512,512,256,256. While for $M2_{cnn}$ are cH1 = 64,32,32,16, cH2 = 128,64,64,32, cH3 = 256,128,128,64, cH4 = 512,256,256,128, cH5 = 1024,512,512,256. Column **Time** reports the training time in seconds.

estimate the number of MBR tests performed during the self-join. In particular, as in the for the selectivity estimation the input embeddings are produced by using autoencoders AE_{S1} and AE_{C2} in the first case, and AE_{S4} and AE_{C3} in the second one.

A.4 Models Estimating the Selectivity and Number of MBR Tests of Binary Spatial Join

This section reports the detailed results of the experiments performed to build the models $M2$ which are used to estimate the selectivity and the number of MBR tests of the binary spatial join, respectively. As in the previous case only four autoencoders are considered, the ones that perform better in the experiments done for the range query selectivity. In particular, given the two considered architectures described in Section 5.4 and denoted as $M2_{dnn}$ and $M2_{cnn}$, we consider for each of them a stacked and a convolutional autoencoder.

Table 18 report the results of training and testing the two models $M2_{dnn}$ and $M2_{cnn}$ for the estimation of binary spatial join selectivity, when we use synthetic datasets only, or synthetic and

Table 18. M2 for Estimating the **Binary Spatial Join Selectivity** for Both Synthetic Datasets Only, and Synthetic and Real Datasets Together

Net. arch.	Hyper par.	Synthetic data					Synthetic and real data				
		AE_{S1}		AE_{C2}		BL	AE_{S4}		AE_{C3}		BL
		WMAPE	Time	WMAPE	Time		WMAPE	Time	WMAPE	Time	
$M2_{dnn}$	dH1	0.3024	403	0.2862	441	0.7274	**0.2636**	131	0.4502	132	0.8424
	dH2	0.2794	406	0.2507	408		0.2863	126	0.2683	95	
	dH3	0.2495	401	0.2555	409		0.3096	131	0.4507	101	
	dH4	0.2409	411	0.2865	420		0.2832	75	0.3993	134	
	dH5	0.2431	432	**0.2251**	431		0.2795	135	0.4217	142	
$M2_{cnn}$	cH1	0.3490	341	0.3122	518	0.7274	0.3459	126	0.3639	127	0.8424
	cH2	0.3184	481	0.2859	551		0.2665	157	0.2637	174	
	cH3	0.3101	483	0.2811	575		0.2302	155	0.2171	163	
	cH4	0.3115	510	**0.2810**	550		0.2521	149	0.2255	174	
	cH5	0.3035	522	0.2907	588		0.2502	168	**0.2125**	175	

Hyperparameters of column Hyperpar. for $M2_{dnn}$ are dH1 = 64,32,32,16,16, dH2 = 128,64,64,32,32, dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, dH5 = 1024,512,512,256,256. While for $M2_{cnn}$ are cH1 = 64,32,32,16, cH2 = 128,64,64,32, cH3 = 256,128,128,64, cH4 = 512,256,256,128, cH5 = 1024,512,512,256. Column **Time** reports the training time in seconds.

Table 19. M2 for Estimating the **Number of MBR Tests for the SJMR Implementation of the Binary Spatial Join** Trained with Both Synthetic Datasets Only, and Synthetic with and Real Datasets Together

Net. arch.	Hyper par.	Synthetic data					Synthetic and real data				
		AE_{S1}		AE_{C2}		BL	AE_{S4}		AE_{C3}		BL
		WMAPE	Time	WMAPE	Time		WMAPE	Time	WMAPE	Time	
$M2_{dnn}$	dH1	0.4032	407	0.8587	54	0.8115	0.3681	140	0.6719	143	1.0085
	dH2	0.3904	403	0.3897	407		0.4275	111	0.4937	87	
	dH3	0.4137	408	0.3672	419		**0.3379**	140	0.7307	123	
	dH4	0.3477	418	**0.3401**	424		0.4321	124	0.6962	147	
	dH5	0.4045	365	0.3533	405		0.4559	108	0.6843	132	
$M2_{cnn}$	cH1	0.8556	98	0.4540	520	0.8115	0.3626	164	0.5321	168	1.0085
	cH2	0.8567	97	0.4100	547		0.3784	166	0.5141	177	
	cH3	0.4587	485	**0.3907**	465		**0.3205**	166	0.4819	191	
	cH4	0.4427	514	0.4140	538		0.3223	169	0.3696	189	
	cH5	0.4722	496	0.4564	571		0.3261	153	0.3307	190	

Hyperparameters of column Hyperpar. are for $M2_{dnn}$ are dH1 = 64,32,32,16,16, dH2 = 128,64,64,32,32, dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, dH5 = 1024,512,512,256,256. While for $M2_{cnn}$ are cH1 = 64,32,32,16, cH2 = 128,64,64,32, cH3 = 256,128,128,64, cH4 = 512,256,256,128, cH5 = 1024,512,512,256. Column **Time** reports the training time in seconds.

real data together. In particular, the input set of these models is in the first case represented by the spatial embeddings generated by the stacked autoencoder AE_{S1} or the convolutional autoencoder AE_{C2}, while in the second case it is represented by the embeddings generated by AE_{S4} and AE_{C3}, respectively . For each of these cases, we report both the error obtained during the estimation and the time required by the training. Various hyperparameter configurations are considered and the best obtained results are highlighted in bold.

Table 19 reports the estimation errors for $M2_{dnn}$ and $M2_{cnn}$ when they are trained and tested on both synthetic datasets only, and synthetic and real datasets together with the aim to estimate the number of MBR tests for the SJMR implementation of the binary spatial join. In particular, in this case the input embeddings are produced by using autoencoders AE_{S1} and AE_{C2} in the first case, and by AE_{S4} and AE_{C3} in the second one. In a similar way, Table 20 reports the same result but for the estimation of the number of MBR tests performed by the DJ implementation of the binary spatial join.

Table 20. M2 for Estimating the **Number of MBR Tests for the DJ Implementation of the Binary Spatial Join** Trained with Both Synthetic Datasets Only, and Synthetic with and Real Datasets Together

Net. arch.	Hyper par.	Synthetic data					Synthetic and real data				
		AE_{S1}		AE_{C2}		BL	AE_{S4}		AE_{C3}		BL
		WMAPE	Time	WMAPE	Time		WMAPE	Time	WMAPE	Time	
$M2_{dnn}$	dH1	0.3768	99	0.2917	93	0.37	0.4119	134	0.4543	101	0.39
	dH2	0.3381	97	0.2767	96		0.3429	116	0.4265	119	
	dH3	**0.2953**	142	0.2723	99		0.3478	1 17	0.4197	119	
	dH4	0.2985	98	0.2696	99		**0.3344**	119	0.4234	122	
	dH5	0.3012	99	**0.2631**	107		0.3391	120	**0.4058**	124	
$M2_{cnn}$	cH1	0.3681	117	0.2998	143	0.37	0.4063	144	0.4063	203	0.39
	cH2	0.3556	119	0.2979	143		0.3765	143	0.4334	144	
	cH3	0.3543	129	**0.2846**	144		0.3884	160	0.3904	177	
	cH4	**0.3382**	136	0.2909	154		0.3699	164	**0.3822**	171	
	H5	0.3485	127	0.2887	263		**0.3645**	199	0.4222	231	

Hyperparameters of column Hyperpar. are for $M2_{dnn}$ are dH1 = 64,32,32,16,16, dH2 = 128,64,64,32,32, dH3 = 256,128,128,64,64, dH4 = 512,256,256,128,128, dH5 = 1024,512,512,256,256. While for $M2_{cnn}$ are cH1 = 64,32,32,16, cH2 = 128,64,64,32, cH3 = 256,128,128,64, cH4 = 512,256,256,128, cH5 = 1024,512,512,256. Column **Time** reports the training time in seconds.

ACKNOWLEDGMENTS

This work is supported in-part by the National Science Foundation under grants, IIS-1838222, CNS-1924694, IIS-1954644, and IIS-2046236.

REFERENCES

[1] Ildar Absalyamov, Michael J. Carey, and Vassilis J. Tsotras. 2018. Lightweight cardinality estimation in LSM-based systems. In *Proceedings of the 2018 International Conference on Management of Data*. Association for Computing Machinery, 841–855. DOI: https://doi.org/10.1145/3183713.3183761

[2] Ahmed M. Aly, Ahmed R. Mahmood, Mohamed S. Hassan, Walid G. Aref, Mourad Ouzzani, Hazem Elmeleegy, and Thamir Qadah. 2015. AQWA: Adaptive query workload aware partitioning of big spatial data. *Proceedings of the VLDB Endowment* 8, 13 (2015), 2062–2073. DOI: https://doi.org/10.14778/2831360.2831361

[3] W. Aref and H. Samet. 1994. A cost model for query optimization using R-Trees. In *Proceedings of the Second ACM Workshop on Advances in Geographic Information Systems, ACM-GIS*. ACM, 60–67.

[4] Walid G. Aref and Hanan Samet. 1993. Estimating selectivity factors of spatial operations. In *Proceedings of the 5th Workshop on Foundations of Models and Languages for Data and Object (Informatik-Berichte des IfI, Vol. 93/9)*, Andreas Heuer and Marc H. Scholl (Eds.). 31–43.

[5] Alberto Belussi and Christos Faloutsos. 1998. Self-spacial join selectivity estimation using fractal concepts. *ACM Transactions on Information Systems* 16, 2 (1998), 161–201. DOI: https://doi.org/10.1145/279339.279342

[6] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2018. Detecting skewness of big spatial data in SpatialHadoop. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. 432–435. DOI: https://doi.org/10.1145/3274895.3274923

[7] A. Belussi, S. Migliorini, and A. Eldawy. 2020. Cost estimation of spatial join in SpatialHadoop. *GeoInformatica* 24, 4 (2020), 1021–1059. DOI: https://doi.org/10.1007/s10707-020-00414-x

[8] Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2022. Spatial embedding: A generic machine learning model for spatial query optimization. In *Proceedings of the 30th International Conference on Advances in Geographic Information Systems*. Association for Computing Machinery. DOI: https://doi.org/10.1145/3557915.3560960

[9] Nicolas Bruno, Surajit Chaudhuri, and Luis Gravano. 2001. STHoles: A multidimensional workload-aware histogram. In *Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 211–222. DOI: https://doi.org/10.1145/375663.375686

[10] Ahmed Eldawy et al. 2021. Beast: Scalable exploratory analytics on spatio-temporal data. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*. ACM, 3796–3807. DOI: https://doi.org/10.1145/3459637.3481897

[11] A. Eldawy and M. F. Mokbel. 2015. SpatialHadoop: A MapReduce framework for spatial data. In *Proceedings of the 2015 IEEE 31st International Conference on Data Engineering*. 1352–1363. DOI: https://doi.org/10.1109/ICDE.2015.7113382

[12] Christos Faloutsos, Bernhard Seeger, Agma Traina, and Caetano Traina. 2000. Spatial join selectivity using power laws. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 177–188. DOI: https://doi.org/10.1145/342009.335412

[13] Yanjie Fu, Pengyang Wang, Jiadi Du, Le Wu, and Xiaolin Li. 2019. Efficient region embedding with multi-view spatial networks: A perspective of locality-constrained spatial autocorrelations. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence and 31st Innovative Applications of Artificial Intelligence Conference and 9th AAAI Symposium on Educational Advances in Artificial Intelligence (AAAI'19/IAAI'19/EAAI'19)*. Article 112, 8 pages. DOI: https://doi.org/10.1609/aaai.v33i01.3301906

[14] Shohedul Hasan, Saravanan Thirumuruganathan, Jees Augustine, Nick Koudas, and Gautam Das. 2020. Deep learning models for selectivity estimation of multi-attribute queries. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1035–1050. DOI: https://doi.org/10.1145/3318464.3389741

[15] Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. 2020. DeepDB: Learn from data, not from queries! *Proceedings of the VLDB Endowment* 13, 7 (2020), 992–1005. DOI: https://doi.org/10.14778/3384345.3384349

[16] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the dimensionality of data with neural networks. *Science* 313, 5786 (2006), 504–507. DOI: https://doi.org/10.1126/science.1127647

[17] Porter Jenkins, Ahmad Farag, Suhang Wang, and Zhenhui Li. 2019. Unsupervised representation learning of spatial data via multimodal embedding. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*. 1993–2002. DOI: https://doi.org/10.1145/3357384.3358001

[18] Puloma Katiyar, Tin Vu, Sara Migliorini, Alberto Belussi, and Ahmed Eldawy. 2020. SpiderWeb: A spatial data generator on the web. In *Proceedings of the 28th International Conference on Advances in Geographic Information Systems*. 465–468. DOI: https://doi.org/10.1145/3397536.3422351

[19] Jin-Deog Kim and Bonghee Hong. 2000. Parallel spatial joins using grid files. In *Proceedings of the 7th International Conference on Parallel and Distributed Systems, ICPADS*. IEEE Computer Society, 531–536. DOI: https://doi.org/10.1109/ICPADS.2000.857739

[20] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter A. Boncz, and Alfons Kemper. 2019. Learned cardinalities: Estimating correlated joins with deep learning. In *Proceedings of the Conference on Innovative Data Systems Research*. Retrieved from http://cidrdb.org/cidr2019/papers/p101-kipf-cidr19.pdf

[21] Sanjay Krishnan, Zongheng Yang, Ken Goldberg, Joseph Hellerstein, and Ion Stoica. 2019. Learning to optimize join queries with deep reinforcement learning. Retrieved from http://arxiv.org/abs/1808.03196

[22] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*. JMLR.org, 1558–1566.

[23] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Learning to steer query optimizers. In *Proceedings of the 2021 International Conference on Management of Data*. 1275–1288. DOI: https://doi.org/10.1145/3448016.3452838

[24] Ryan Marcus and Olga Papaemmanouil. 2018. Deep reinforcement learning for join order enumeration. In *Proceedings of the 1st International Workshop on Exploiting Artificial Intelligence Techniques for Data Management*. ACM, 3:1–3:4. DOI: https://doi.org/10.1145/3211954.3211957

[25] Ryan C. Marcus et al. 2019. Neo: A learned query optimizer. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1705–1718. DOI: https://doi.org/10.14778/3342263.3342644

[26] Yossi Matias, Jeffrey Scott Vitter, and Min Wang. 1998. Wavelet-based histograms for selectivity estimation. *SIGMOD Record* 27, 2 (1998), 448–459. DOI: https://doi.org/10.1145/276305.276344

[27] Sara Migliorini, Alberto Belussi, Mauro Negri, and Giuseppe Pelagatti. 2016. Towards massive spatial data validation with SpatialHadoop. In *Proceedings of the 5th ACM SIGSPATIAL International Workshop on Analytics for Big Geospatial Data*. 18–27. DOI: https://doi.org/10.1145/3006386.3006392

[28] Tomás Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. In *1st International Conference on Learning Representations, ICLR*. Retrieved from http://arxiv.org/abs/1301.3781

[29] Jignesh M. Patel and David J. DeWitt. 1996. Partition based spatial-merge join. *SIGMOD Record* 25, 2 (1996), 259–270. DOI: https://doi.org/10.1145/235968.233338

[30] Viswanath Poosala, Peter J. Haas, Yannis E. Ioannidis, and Eugene J. Shekita. 1996. Improved histograms for selectivity estimation of range predicates. *SIGMOD Record* 25, 2 (1996), 294–305. DOI: https://doi.org/10.1145/235968.233342

[31] Ibrahim Sabek and Mohamed F. Mokbel. 2017. On Spatial Joins in MapReduce. In *Proceedings of the 25th International Conference on Advances in Geographic Information Systems*. Article 21, 10 pages. DOI: https://doi.org/10.1145/3139958.3139967

[32] Samriddhi Singla and Ahmed Eldawy. 2022. Flexible computation of multidimensional histograms. In *Proceedings of the Spatial Gems, Volume 1* (1 ed.). Association for Computing Machinery, 119–130.

[33] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2020. Using deep learning for big spatial data partitioning. *ACM Transactions on Spatial Algorithms and Systems* 7, 1 (2020), 3:1–3:37. DOI: https://doi.org/10.1145/3402126

[34] Tin Vu, Alberto Belussi, Sara Migliorini, and Ahmed Eldawy. 2021. A learned query optimizer for spatial join. In *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*. Association for Computing Machinery, New York, NY, USA, 458–467. DOI: https://doi.org/10.1145/3474717.3484217

[35] Tin Vu and Ahmed Eldawy. 2018. R-Grove: Growing a family of R-trees in the big-data forest. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. Association for Computing Machinery, 532–535. DOI: https://doi.org/10.1145/3274895.3274984

[36] Tin Vu, Sara Migliorini, Ahmed Eldawy, and Alberto Belussi. 2022. Spatial data generators. In *Spatial Gems, Volume 1* (1 ed.). Association for Computing Machinery, New York, NY, USA, 13–24. DOI: https://doi.org/10.1145/3548732.3548736

[37] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One cardinality estimator for all tables. *Proceedings of the VLDB Endowment* 14, 1 (2020), 61–73. DOI: https://doi.org/10.14778/3421424.3421432

[38] J. Yu, J. Wu, and M. Sarwat. 2015. 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*. 70:1–70:4. DOI: https://doi.org/10.1145/2820783.2820860

[39] Wenchao Yu, Guangxiang Zeng, Ping Luo, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. 2013. Embedding with autoencoder regularization. In *Proceedings of the Machine Learning and Knowledge Discovery in Databases*. Springer Berlin Heidelberg, 208–223.

[40] Shubin Zhang, Jizhong Han, Zhiyong Liu, Kai Wang, and Zhiyong Xu. 2009. SJMR: Parallelizing spatial join with MapReduce on clusters. In *Proceedings of the CLUSTER*. IEEE Computer Society, New Orleans, LA, 1–8. DOI: https://doi.org/10.1109/CLUSTR.2009.5289178

Received 15 March 2023; revised 9 December 2023; accepted 4 April 2024