



Anagnostopoulos, C. and Triantafillou, P. (2017) Efficient Scalable Accurate Regression Queries in In-DBMS Analytics. In: IEEE International Conference on Data Engineering (ICDE), San Diego, CA, USA, 19-22 Apr 2017, (doi:[10.1109/ICDE.2017.111](https://doi.org/10.1109/ICDE.2017.111))

This is the author's final accepted version.

There may be differences between this version and the published version. You are advised to consult the publisher's version if you wish to cite from it.

<http://eprints.gla.ac.uk/136690/>

Deposited on: 10 February 2017

Enlighten – Research publications by members of the University of Glasgow
<http://eprints.gla.ac.uk33640>

Efficient Scalable Accurate Regression Queries in In-DBMS Analytics

Christos Anagnostopoulos, Peter Triantafillou

School of Computing Science, University of Glasgow

Email: {christos.anagnostopoulos; peter.triantafillou}@glasgow.ac.uk

Abstract—Recent trends aim to incorporate advanced data analytics capabilities within DBMSs. Linear regression queries are fundamental to exploratory analytics and predictive modeling. However, computing their exact answers leaves a lot to be desired in terms of efficiency and scalability. We contribute a novel predictive analytics model and associated regression query processing algorithms, which are efficient, scalable and accurate. We focus on predicting the answers to two key query types that reveal dependencies between the values of different attributes: (i) mean-value queries and (ii) multivariate linear regression queries, both within specific data subspaces defined based on the values of other attributes. Our algorithms achieve many orders of magnitude improvement in query processing efficiency and near-perfect approximations of the underlying relationships among data attributes.

I. INTRODUCTION

Predictive Modeling and Analytics (PMA) concerns data exploration, model fitting, and prediction model learning. Predictive models like linear regression are typically desired for exploring data subspaces of a d -dim. data space of interest in \mathbb{R}^d . In in-DBMS analytics, such data subspaces are identified using selection operators over the values of attributes of interest. Selection operators include *radius* (a.k.a. *distance near neighbor* (dNN)) queries, which are of high importance in many applications: location-based search, searching for statistical correlations of spatially close objects (within a radius), spatial analytics focusing on the construction of semi-variograms in a specific geographical region, earth analytics monitoring regions of interest from sensors’ acoustic signals, multimedia analytics for learning images’ similarity, and environmental monitoring for chemical compounds correlation analysis given a geographical area. The analytics process then is as follows: Data analysts interact with in-DBMS analytics tools by issuing selection queries (i.e., dNN queries) to define data subspaces $\mathbb{D} \subset \mathbb{R}^d$. Then, the local dependencies among the features in those subspaces are extracted *and* regression models are evaluated for their goodness of fit over those \mathbb{D} , i.e., by identifying the model that is most likely to have generated those data in \mathbb{D} . For concreteness, we focus on defining data subspaces of interest $\mathbb{D}(\mathbf{x}_0, \theta)$ using a dNN query, notated by Q , as the convex subset of d -dim. data points (vectors) $\mathbf{x} = [x_1, \dots, x_d] \in \mathbb{R}^d$ lying within a hypersphere (ball) with center \mathbf{x}_0 and a scalar radius θ , i.e., $\mathbb{D}(\mathbf{x}_0, \theta)$ contains all $\mathbf{x} \in \mathbb{R}^d : \|\mathbf{x} - \mathbf{x}_0\|_2 \leq \theta$; $\|\mathbf{x}\|_2$ is the Euclidean norm.

We focus on two important queries for in-DBMS analytics: mean-value and linear regression queries. For example, consider Figure 1 (left): Seismologists issue *analytics query* Q_1 over a 3-dim space $(u, x_1, x_2) \in \mathbb{R}^3$, which returns the mean y value of feature u (seismic signal; P-wave speed) of those

$(x_1, x_2) \in \mathbb{D}(\mathbf{x}_0, \theta) \subset \mathbb{R}^2$ projections (referring to surface longitude and latitude) within a disc of center \mathbf{x}_0 and radius θ . Query Q_1 is central to PMA because the average y can be used as a linear sufficient statistic for subspace \mathbb{D} and it is the best linear predictor of u based on $(x_1, x_2) \in \mathbb{D}(\mathbf{x}_0, \theta)$ [4].

A *linear regression* query Q_2 calculates the coefficients of a linear regression function within defined data subspaces. For example (see Figure 1 (left)), consider geophysicists issuing queries Q_2 over a 3-dim space $(u, x_1, x_2) \in \mathbb{R}^3$, which returns the seismic P-wave velocity u -intercept (b_0) and the coefficients b_1 and b_2 for x_1 (longitude) and x_2 (latitude), where $\mathbf{x} = [x_1, x_2]$ points belong to a subspace $\mathbb{D}(\mathbf{x}_0, \theta) \in \mathbb{R}^2$. By estimating the linear coefficients, e.g., vector $\mathbf{b} = [b_0, b_1, b_2]$, we can then interpret the relationships among features \mathbf{x} and u and assess the statistical significance of each feature of \mathbf{x} within $\mathbb{D}(\mathbf{x}_0, \theta)$. The output of Q_2 (i) refers to the dependency of u with \mathbf{x} , which in our example is approximated by a 2-dim. plane $u \approx b_0 + b_1x_1 + b_2x_2$, and (ii) quantifies how well the local linear model fits the data. Query Q_2 is important in PMA because (i) supports model fitting through e.g., Piece-wise Linear Regression (PLR) [12], and (ii) provides confidence whether linear models fit well or not the underlying data. (Please refer to Appendix IV in¹ for the SQL syntax of Q_1 and Q_2 queries.) To evaluate queries Q_1 and Q_2 the system (i) must access the data to establish the data subspace $\mathbb{D}(\mathbf{x}_0, \theta)$, and then (ii) take the average value of u in that subspace for Q_1 (e.g., the average seismic signal speed in San-Andreas, CA, region) and invoke a multivariate linear regression algorithm [4] for Q_2 . Q_1 and Q_2 type queries are provided by all modern PMA like Matlab² and many DBMS systems, e.g., XLeratorDB [5] of Microsoft SQL Server [7] and Oracle UTL_NLA [6].

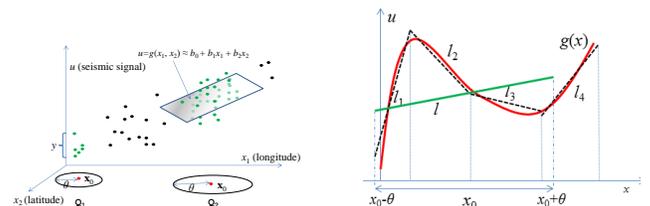


Fig. 1. (Left) Analytics Q_1 and linear regression Q_2 queries over $(u, x_1, x_2) \in \mathbb{R}^3$; (right) Non-linearity of the data function g , global linear l and local linear l_1, \dots, l_4 approximations of $g(x) : |x - x_0| \leq \theta$.

¹<https://www.dropbox.com/s/tiblnj9har3znu/af.pdf?dl=0>

²<https://www.mathworks.com>

A. Motivation and the Problem

We focus on in-DBMS complex analytics with PMA using models and algorithms for query types Q1 and Q2. Specifically, the aim is to meet the following desiderata, providing answers to the following analytics questions: (D1) Are there linear dependencies among attributes in unexplored data subspaces, and which are such subspaces? (D2) If there are data subspaces, where linear approximations fit well with high confidence, can the system provide these linear regression models efficiently and more scalably? (D3) If in some subspaces linear approximations do not fit well w.r.t. analysts needs, can the system provide fitting models through piece-wise local linear approximations? However, our solution must also meet *scalability, efficiency, and accuracy* desiderata (D4) as well.

Concerning desiderata D2 and D3, the derivation of several *local* linear approximations (as opposed to a single linear approximation over the *whole broad* data subspace), can provide more accurate and significant further insights. The key issue to note here is that a ‘global’ (single) linear approximation interpolating among all items of the whole data subspace \mathbb{D} leaves, in general, much to be desired: The analysts, presented with a single ‘global’ linear approximation, might have an inaccurate view, due to missing ‘local’ statistical dependencies within *unknown local data subspaces that comprise* \mathbb{D} . This will surely lead to prediction errors and approximation inaccuracies, when using e.g., Q1 and Q2. For instance, consider the (u, x) 2-dim. space in Figure 1 (right) and the actual data function $u = g(x)$ (in red). A Q2 query over the data subspace $\mathbb{D}(x_0, \theta)$ will calculate the intercept b_0 and slope b_1 of the linear approximation $u \approx b_0 + b_1x$ (the green line l) over those $x \in \mathbb{D}(x_0, \theta)$. Evidently, such a line shows a very coarse and unrepresentative dependency between u and x , since u and x do not linearly depend on each other within the entire $\mathbb{D}(x_0, \theta)$. The point is that we should obtain a finer grained (and more accurate) dependency between u and x . The *principle of local linearity* [22] states that linear approximations of the underlying data function in *certain* data subspaces fit the global non-linearity better in the entire data subspace of interest. In Figure 1 (right), we observe four local linear approximations l_1, \dots, l_4 in a data subspace. Therefore, it would be great if, as a result of Q2, the analysts were provided with a list of the local lines $\mathcal{S} = \{l_1, \dots, l_4\}$, a.k.a. piece-wise linear regression. These ‘local’ lines better approximate the linearity of u .

Concerning desideratum D1, the analysts do not know before issuing a query how the data function behaves within an *ad hoc* defined $\mathbb{D}(x_0, \theta)$. When Q2 is issued, it is not known (i) whether g behaves with the same linearity throughout the entire $\mathbb{D}(x_0, \theta)$ or not, and (ii) within which subspaces, if any, g changes its trend and u and x exhibit linear dependencies. Thus, the central goals for D1 are to (i) learn the boundaries of these local subspaces within $\mathbb{D}(x_0, \theta)$ and (ii) within each local subspace, derive the linear dependency between u and x . This would arm analysts with much more accurate knowledge on how $g(x)$ behaves within $\mathbb{D}(x_0, \theta)$. Hence, decisions on further data exploration w.r.t. complex model selection and/or validation can be taken by the analysts.

Our motivations rest on learning and predicting how attributes are correlated differently in certain data subspaces, within a greater data subspace defined by many issued queries. Alas, state of the art methods leave a lot to be desired in terms

of efficiency and scalability. Our key insight and contribution in this direction lies in the development of learning models which can deliver the above functionality for Q1 and Q2 in a way that is highly accurate and insensitive to the sizes of the underlying data spaces, and thus by definition scalable. The essence of the novel idea we put forward rests on (i) *exploiting previously executed queries and their answers* obtained from the DBMS/PMA system to train a model and then (ii) use that model to (ii.a) predict the output of any (unseen, new) query and (ii.b) predict the list \mathcal{S} of linear regression coefficients corresponding to different linear models, that best explain (fit) the underlying data function over data subspaces. In the prediction phase (that is, after training) no access to the underlying data system is required (ensuring D4).

In Figure 2 we show the system context within which our contributions unfold. A DBMS serves analytics queries from a large user community. Over time, all users will have issued a large number of queries ($\mathcal{Q} = \{Q_1, Q_2, \dots, Q_n\}$), and the system would have produced responses (e.g., y_1, y_2, \dots, y_n for Q1 queries). Our key idea is to inject a novel statistical learning model and novel query processing algorithms in between users and the DBMS that monitors queries and responses and learns to associate a query with its response. After training, say after the first $m < n$ queries $\mathcal{T} = \{Q_1, \dots, Q_m\}$ then, for any *new/unseen* query Q_t with $m < t \leq n$, i.e., $Q_t \in \mathcal{V} = \mathcal{Q} \setminus \mathcal{T} = \{Q_{m+1}, \dots, Q_n\}$, our model predicts (i) its response \hat{y}_t and (ii) the list \mathcal{S} of local linear regression coefficients *without accessing the underlying DBMS*. The efficiency and scalability benefits of our approach are evident. Computing the answers to Q1 and Q2 can be very time-consuming, especially for large data subspaces. So if this model and algorithms can deliver highly-accurate answers, query processing times will be dramatically reduced. Scalability is also ensured for two reasons. Firstly, in the *data dimension*, as Q1 and Q2 executions (after training) do not involve data accesses, even dramatic increases in DB size do not impact query execution. Secondly, in the *query-throughput dimension*, avoiding DBMS internal resource utilization (that would be required if all Q1 and Q2 queries were executed over the DBMS data) saves resources that can be devoted to support larger numbers of queries at any given point in time.

Overall, in this work, with **one** model, given a query over $\mathbb{D}(x_0, \theta)$, we contribute how to: (i) predict the average value y of $u = g(x)$ with $x \in \mathbb{D}(x_0, \theta)$, (ii) deliver a list \mathcal{S} of the local linear coefficients of as many linear models as required to best explain the underlying data function g within $\mathbb{D}(x_0, \theta)$, and (iii) predict data value $\hat{u} \approx g(x)$ for each $x \in \mathbb{D}(x_0, \theta)$.

Challenges: Several challenges exist: (C1) predict the average value of y for unseen Q1; (C2) identify the number and boundaries of the data subspaces with local linearities and deliver the local linear approximations for each subspace identified, i.e., predict the list \mathcal{S} for unseen Q2. Clearly, C2 copes further with the following problems: the boundaries of these data subspaces are (i) unknown and (ii) cannot be determined even if we could scan all of the data (which in any case would be inefficient and less scalable. It is worth-noting that this cannot be achieved solely by accessing the data, as we need information on which are the users’ ad-hoc defined subspaces of interest. It is possible to provide this information a priori for *all* possible data subspaces of interest to analysts,

i.e., consider all possible centre points \mathbf{x}_0 and all possible θ values. However, this is clearly impractical—this knowledge is obtained *after* the analysts have issued queries over the data, thus, reflecting their sub-spaces of interest and exploration.

B. Related Work

Outwith DBMS environments, packages, like Matlab and R³ support fitting regression functions. However, their algorithms for doing so are inefficient and hardly scalable, if at all. Moreover, they badly lack support for relational, declarative, Q1 and Q2 queries. So, if data are already in a RDBMS, they would need to be moved back and forth between external analytics environments and the RDBMS, resulting in considerable inconveniences and performance overheads, (if at all possible for big datasets). At any rate, modern DBMSs should provide analysts with rich support for PMA.

An increasing number of major database vendors include in their products data mining and machine learning analytic tools. Moreover, PostgreSQL, MySQL, MADLib (over PostgreSQL) [16] and commercial tools like Oracle Data Miner [9], IBM Intelligent Miner [8] and Microsoft SQL Server Data Mining [5] provide SQL-like interfaces for analysts to specify regression tasks. Academic efforts include MauveDB [11], which integrates regression models into a RDBMS, while, similarly, FunctionDB [10] allows analysts to directly pose regression queries against a RDBMS. Also, BISMARCK [17] integrates and supports in-RDBMS analytics, while [19] integrates and supports least squares regression models over training datasets defined by arbitrary join queries on database tables. All such works that also support Q1 and Q2 queries can serve as the DBMS within Figure 2. However, in the big data era exact Q1, Q2 computations leave much to be desired in efficiency and scalability, as the system must first execute the selection, establishing the data subspaces per query, and then access all tuples in Q1, Q2.

Our approach is the first that accurately supports both: (i) *predicting* the result of analytics Q1 queries and (ii) *predicting*

³<https://www.r-project.org/>

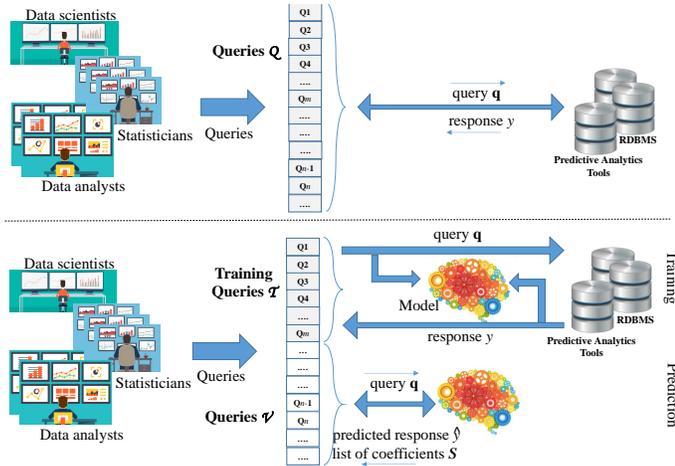


Fig. 2. The System Context: (Upper) User community queries \mathcal{Q} without our model; (Lower) Our model learns from past queries \mathcal{T} and predicts future query results \mathcal{V} .

the (multiple) local linear coefficients of multivariate linear regression Q2 queries. And, it does so while achieving high prediction accuracy and goodness of fit after training without executing Q1 and Q2, thus, without accessing data. This ensures a highly efficient and scalable solution, which is *independent* of the data sizes, as Section VI will show.

C. Contribution

The contribution of this work lies in efficient and scalable models and algorithms to obtain highly accurate results for mean-value & linear regression queries. This rests on learning the principal local linear approximations of g . Our approach is *query-driven*: Queries are exploited to partition the queried data space into subspaces. In each subspace, we incrementally approximate the data function based on a novel local approximation methodology. After training, we can (i) deliver information about the behavior of g over different subspaces; (ii) predict the output of unseen queries; (iii) predict multiple linear coefficients of subspaces. The detailed technical contributions are:

- Novel training algorithms for approximating data over multi-dim. spaces;
- Novel joint optimization algorithm for processing linear regression and mean-value analytics queries;
- Convergence analyses of the algorithms;
- Prediction algorithms for PMA queries;
- Comparative performance assessment.

II. FORMAL PROBLEM ANALYSIS

Let $\mathbf{x} = [x_1, \dots, x_d] \in \mathbb{R}^d$ denote a multivariate random *input* row vector, and $u \in \mathbb{R}$ a univariate random *output* variable, with (unknown) joint probability distribution $P(u, \mathbf{x})$. We denote $g : \mathbb{R}^d \rightarrow \mathbb{R}$ with $\mathbf{x} \mapsto u$ the unknown underlying data function from input \mathbf{x} to output $u = g(\mathbf{x})$.

Definition 1: The linear regression function of input $\mathbf{x} \in \mathbb{R}^d$ onto output $u \in \mathbb{R}$ is: $u = b_0 + \sum_{i=1}^d b_i x_i + \epsilon = b_0 + \mathbf{b}\mathbf{x}^\top + \epsilon$, where: ϵ is a random error with mean $\mathbb{E}[\epsilon] = 0$ and variance $Var(\epsilon) = \sigma^2 > 0$, $\mathbf{b} = [b_1, \dots, b_d]$ is the **slope** row vector of real coefficients, and b_0 is the **intercept**.

Definition 2: The p -norm (L_p) distance between two input vectors \mathbf{x} and \mathbf{x}' from \mathbb{R}^d for $1 \leq p < \infty$, is $\|\mathbf{x} - \mathbf{x}'\|_p = (\sum_{i=1}^d |x_i - x'_i|^p)^{\frac{1}{p}}$ and for $p = \infty$, is $\|\mathbf{x} - \mathbf{x}'\|_\infty = \max_{i=1, \dots, d} \{|x_i - x'_i|\}$.

Consider now a scalar $\theta > 0$, hereinafter referred to as *radius*, and a dataset \mathcal{B} consisting of n pairs (\mathbf{x}_i, u_i) .

Definition 3: Given $\mathbf{x} \in \mathbb{R}^d$ and θ , a data subspace $\mathbb{D}(\mathbf{x}, \theta)$ is the convex subspace of \mathbb{R}^d , which includes vectors $\mathbf{x}_i : \|\mathbf{x}_i - \mathbf{x}\|_p \leq \theta$ with $(\mathbf{x}_i, u_i) \in \mathcal{B}$.

Definition 4: Query Q1. Given a vector $\mathbf{x} \in \mathbb{R}^d$ and θ , the (average distance near neighbors) Q1 query over a dataset \mathcal{B} returns the average of the outputs $u_i = g(\mathbf{x}_i)$, whose corresponding input vectors $\mathbf{x}_i \in \mathbb{D}(\mathbf{x}, \theta)$, i.e.,

$$y = \frac{1}{n_\theta(\mathbf{x})} \sum_{i \in [n_\theta(\mathbf{x})]} u_i : \|\mathbf{x}_i - \mathbf{x}\|_p \leq \theta \quad (1)$$

where $n_\theta(\mathbf{x})$ is the cardinality of the set $\{\mathbf{x}_i : \|\mathbf{x}_i - \mathbf{x}\|_p \leq \theta\}$ and $(\mathbf{x}_i, u_i) \in \mathcal{B}$. We represent a query as the $(d+1)$ -dim. row vector $\mathbf{q} = [\mathbf{x}, \theta] \in \mathbb{Q} \subset \mathbb{R}^{d+1}$. The $(d+1)$ -dim. space \mathbb{Q} is referred to as query vectorial space. We adopt the compact notation $i \in [n]$ as for $i = 1, \dots, n$.

Definition 5: The L_2^2 distance or similarity measure between queries $\mathbf{q}, \mathbf{q}' \in \mathbb{Q}$ is $\|\mathbf{q} - \mathbf{q}'\|_2^2 = \|\mathbf{x} - \mathbf{x}'\|_2^2 + (\theta - \theta')^2$.

Definition 6: The queries \mathbf{q}, \mathbf{q}' , which define the subspaces $\mathbb{D}(\mathbf{x}, \theta)$ and $\mathbb{D}(\mathbf{x}', \theta')$, respectively, overlap if for the boolean indicator $\mathcal{A}(\mathbf{q}, \mathbf{q}') \in \{\text{TRUE}, \text{FALSE}\}$ holds true that: $\mathcal{A}(\mathbf{q}, \mathbf{q}') = (\|\mathbf{x} - \mathbf{x}'\|_p \leq \theta + \theta') = \text{TRUE}$.

A query $\mathbf{q} = [\mathbf{x}, \theta]$ defines a data subspace $\mathbb{D}(\mathbf{x}, \theta)$ w.r.t. dataset/relation \mathcal{B} . Formally, our challenges are **C1**: *predict* the outcome \hat{y} of a random query $\mathbf{q} = [\mathbf{x}, \theta]$. We seek a function $f : \mathbb{Q} \subset \mathbb{R}^{d+1} \rightarrow \mathbb{R}$ with $(\mathbf{x}, \theta) \mapsto y$ to predict $y = f(\mathbf{x}, \theta)$ for unseen query \mathbf{q} . **C2**: *identify* the local linear approximations of the unknown data function $u = g(\mathbf{x})$ over the data subspaces $\mathbb{D}(\mathbf{x}_t, \theta_t)$ defined by past executed queries $\mathbf{q}_t = [\mathbf{x}_t, \theta_t]$ on \mathcal{B} .

A. Problem Formulation

Consider C1 and let us adopt the squared prediction error function $(y - f(\mathbf{x}, \theta))^2$ for penalizing errors in prediction. This leads to a criterion for choosing a function f , which minimizes the *Expected Prediction Error* (EPE):

$$\mathbb{E}[(y - f(\mathbf{x}, \theta))^2] = \mathbb{E}_{\mathbf{x}, \theta}[\mathbb{E}_y[(y - f(\mathbf{x}, \theta))^2 | \mathbf{x}, \theta]] \quad (2)$$

Before finding the family of this function that minimizes the EPE in (2), we rest on the *law of iterated expectations* for the dependent variable y from \mathbf{x} and θ , i.e., $\mathbb{E}[y] = \mathbb{E}[\mathbb{E}[y | \mathbf{x}, \theta]]$, where y breaks into two pieces, as follows:

Theorem 1 (Decomposition): $y = \mathbb{E}[y | \mathbf{x}, \theta] + \epsilon$, where ϵ is mean-independent of \mathbf{x} and θ , i.e., $\mathbb{E}[\epsilon | \mathbf{x}, \theta] = 0$ and therefore ϵ is uncorrelated with any function of \mathbf{x} and θ .

For proof of Theorem 1, please refer to [4]. According to Theorem 1, y can be decomposed into (i) a conditional expectation function $\mathbb{E}[y | \mathbf{x}, \theta]$, hereinafter referred to as *regression function*, which is explained by \mathbf{x} and θ , and (ii) a left over (noisy component) which is orthogonal to (i.e., uncorrelated with) any function of \mathbf{x} and θ . In our context, the regression function is a good candidate for minimizing the EPE in (2) envisaged as a *local* representative value for y over the subspace $\mathbb{D}(\mathbf{x}, \theta)$. Therefore, the conditional expectation function is the best predictor of y given $\mathbb{D}(\mathbf{x}, \theta)$:

Theorem 2 (Conditional Expectation Function): Let $f(\mathbf{x}, \theta)$ be any function of \mathbf{x} and θ . The conditional expectation function $\mathbb{E}[y | \mathbf{x}, \theta]$ solves the optimization problem: $\mathbb{E}[y | \mathbf{x}, \theta] = \arg \min_{f(\mathbf{x}, \theta)} \mathbb{E}[(y - f(\mathbf{x}, \theta))^2]$, i.e., it is the minimum mean squared error predictor of y given \mathbf{x}, θ .

For proof of Theorem 2, please refer to [4]. We rely on Theorems 1 & 2 to build our methodology.

Outcome from C1: The solution to (2) is $f(\mathbf{x}, \theta) = \mathbb{E}[y | \mathbf{x}, \theta]$, i.e., the conditional expectation of y over $\mathbb{D}(\mathbf{x}, \theta)$. However, the number of pairs $n_\theta(\mathbf{x})$ is finite, thus, such conditional expectation is approximated by averaging all u_i 's

conditioning at $\mathbf{x}_i \in \mathbb{D}(\mathbf{x}, \theta)$. We propose a novel model to approximate $\mathbb{E}[y | \mathbf{x}, \theta]$ and predict y given query $\mathbf{q} = [\mathbf{x}, \theta]$.

Outcome from C2: The result y of a query \mathbf{q} refers to the best regression estimator over $\mathbb{D}(\mathbf{x}, \theta)$. Each query provides information to locally *learn* the dependency between u and \mathbf{x} . In this context, *similar* queries w.r.t. L_2 provide insight for g over *overlapped* subspaces. The model from C1's outcome is exploited to estimate the linear regression coefficients (intercept and slope) between u and \mathbf{x} by interpolating among multiple local approximations in $\mathbb{D}(\mathbf{x}, \theta)$.

Assume a continuous stream $\{(\mathbf{q}_1, y_1), \dots, (\mathbf{q}_t, y_t)\}$ of pairs (query, answer) through the interactions between the users and the system, with $\mathbf{q}_t = [\mathbf{x}_t, \theta_t]$ and answer y_t .

Problem 1: Approximate the conditional expectation function $f(\mathbf{x}, \theta)$ and predict the outcome \hat{y} of an unseen query.

Problem 2: Based on $f(\mathbf{x}, \theta)$, approximate the data function $g(\mathbf{x})$ by interpolating local linear regression functions over multiple data subspaces.

Remark 1: Due to space limitations, the authors have moved the proofs of the theorems of this paper to appendix: <https://www.dropbox.com/s/tlibnj9har3znu/af.pdf?dl=0>.

B. Preliminaries

Stochastic Gradient Descent (SGD) [14] is an optimization method for minimizing an objective function $\mathcal{E}(\alpha)$, where α is a parameter and α^* minimizes \mathcal{E} . SGD leads to fast convergence to α^* by adjusting the estimated α so far in the direction (negative gradient $-\nabla \mathcal{E}$) that improves the minimization of \mathcal{E} . SGD gradually changes α upon reception of a new training sample. SGD computes the gradient of \mathcal{E} using only a single training pair at step t . The update of α_t at step t is given by: $\Delta \alpha_t = -\eta_t \nabla_{\alpha_t} \mathcal{E}(\alpha_t)$. The *learning rate* $\{\eta_t\} \in (0, 1)$ is a step-size schedule, which defines a slowly decreasing sequence of scalars that satisfy: $\sum_{t=1}^{\infty} \eta_t = \infty$ and $\sum_{t=1}^{\infty} \eta_t^2 < \infty$. Usually, we adopt a hyperbolic schedule: $\eta_t = \frac{1}{t+1}$ [14].

Adaptive Vector Quantization (AVQ) refers to a clustering algorithm [15]. AVQ partitions a d -dim. space \mathbb{R}^d into a fixed number of K subspaces. AVQ distributes K prototypes $\mathbf{w}_1, \dots, \mathbf{w}_K$ in \mathbb{R}^d . A prototype \mathbf{w}_k represents a subspace of \mathbb{R}^d and behaves as a *quantization vector*. An AVQ algorithm learns as \mathbf{w}_k changes in response to random vector $\mathbf{x} \in \mathbb{R}^d$. Competition selects which \mathbf{w}_i the training \mathbf{x} modifies. The j -th prototype 'wins' if \mathbf{w}_j is the closest (under L_p) of the K prototypes to \mathbf{x} . During partition, points \mathbf{x} are projected onto their winning prototypes and prototypes adaptively move around the space to form optimal partitions (subspaces of \mathbb{R}^d) that minimize the *Expected Quantization Error* (EQE): $\mathbb{E} \left[\min_k \|\mathbf{x} - \mathbf{w}_k\|_p^p \right]$, with winner prototype \mathbf{w}_j such that $\|\mathbf{w}_j - \mathbf{x}\|_p = \min_k \|\mathbf{w}_k - \mathbf{x}\|_p$.

III. SOLUTION FUNDAMENTALS

We first proceed with a solution for Q1 queries (Problem 1) to approximate function f . Then, we utilize such solution to address Q2 queries (Problem 2) to approximate the data function g . Concerning Problem 1 and Theorem 2, we approximate $f(\mathbf{x}, \theta) = \mathbb{E}[y | \mathbf{x}, \theta]$ that minimizes (2). However, y in

(1) involves the average of the images $g(\mathbf{x}_i) = u_i, i \in [n_\theta(\mathbf{x})]$. Hence, $f(\mathbf{x}, \theta)$ is a non trivial compound function of $g(\mathbf{x})$ for an arbitrary θ and L_p norm. Moreover, the non-linearity of g over certain subspaces is further propagated to f by definition of y in (1). Thus, we must identify those data subspaces where g behaves *almost* linearly, which should be reflected in the f approximation. This will provide key insight on approximating both f and g through a finite set of local linear functions, known as Local Linear Mappings (LLMs).

In Problem 1, we approximate $f(\mathbf{x}, \theta)$ with a set of LLMs, each of which is constrained to a local region of the query space \mathbb{Q} , defined by *similar* queries w.r.t. L_2 . Similar queries are those queries with similar centers \mathbf{x} and similar radii θ . Our general idea for those LLMs is the quantization of the query space \mathbb{Q} into a finite number of query subspaces \mathbb{Q}_k such that the function f can be linearly approximated by an LLM $f_k, k = 1 \dots, K$. Those query subspaces may be rather large in areas of \mathbb{Q} where f indeed behaves approximately linear and must be smaller where this is not the case. The total number K of such query subspaces depends on the desired approximation (goodness of fit) and prediction accuracy and may be limited by the available issued queries since over-fitting might occur. Fundamentally, we incrementally quantize the query space \mathbb{Q} over a series of issued queries through quantization vectors (prototypes) in \mathbb{Q} and associate each \mathbb{Q}_k with a LLM f_k in the output space, where f behaves approximately linear.

In Problem 2, principally each query subspace \mathbb{Q}_k associates with a data subspace \mathbb{D}_k , i.e., for a query $\mathbf{q} \in \mathbb{Q}_k \subset \mathbb{R}^{d+1}$, its corresponding center $\mathbf{x} \in \mathbb{D}_k \subset \mathbb{R}^d$. This implies that the input vector \mathbf{x} (of query \mathbf{q}) is constrained to be drawn *only* from the k -th data subspace \mathbb{D}_k . Based on that association, we utilize the LLM f_k to estimate the local intercept and slope of the data function g over the k -th data subspace \mathbb{D}_k .

A. Local Linear Mapping for Queries

Our solutions to Problems 1 and 2 are based on LLMs. A LLM $f_k : \mathbb{Q}_k \rightarrow \mathbb{R}, k \in [K]$, approximates the dependency between y and \mathbf{q} over \mathbb{Q}_k defined by similar queries under L_2 . The multivariate first-order Taylor expansion of the scalar-valued function $f(\mathbf{q}) = f(\mathbf{x}, \theta) = f(x_1, \dots, x_d, \theta)$ for a query \mathbf{q} near a query vector $\mathbf{q}_0 = [\mathbf{x}_0, \theta_0]$ is:

$$f(\mathbf{q}) \approx f(\mathbf{q}_0) + \nabla f(\mathbf{q}_0)(\mathbf{q} - \mathbf{q}_0)^\top, \quad (3)$$

where $\nabla f(\mathbf{q}_0)$ is the gradient of f at \mathbf{q}_0 , i.e., the $1 \times (d+1)$ matrix of partial derivatives $\frac{\partial f}{\partial x_i}, i \in [d]$ and $\frac{\partial f}{\partial \theta_k}$. We derive a LLM f_k from Taylor's approximation around the *local expectation query* $\mathbb{E}[\mathbf{q}] = [\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta]]$ of queries $\mathbf{q} \in \mathbb{Q}_k$:

$$f_k(\mathbf{x}, \theta) \approx f_k(\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta]) + \nabla f_k(\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta])([\mathbf{x}, \theta] - [\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta]])^\top \quad (4)$$

Specifically, the coefficients of LLM f_k are:

- The **local intercept**, with components: (i) local expectation of y , i.e., $\mathbb{E}[y] = f_k(\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta])$, notated by the scalar coefficient y_k ; (ii) local expectation query $\mathbb{E}[\mathbf{q}] = [\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta]]$ notated by the vectorial coefficient $\mathbf{w}_k = [\mathbf{x}_k, \theta_k] \in \mathbb{Q}_k$, with $\mathbf{x}_k = \mathbb{E}[\mathbf{x}]$ and $\theta_k = \mathbb{E}[\theta]$ such that $[\mathbf{x}, \theta] \in \mathbb{Q}_k$. Hereinafter, \mathbf{w}_k is referred to as the **prototype** of the query subspace \mathbb{Q}_k .

- The **local slope** $\mathbf{b}_k = [\mathbf{b}_{X,k}, b_{\Theta,k}]$ of f_k over \mathbb{Q}_k , which denotes the gradient $\nabla f_k(\mathbb{E}[\mathbf{x}], \mathbb{E}[\theta])$ of f_k at the local expectation query \mathbf{w}_k .

Based on these constructs, f_k can be rewritten as:

$$f_k(\mathbf{x}, \theta) \approx y_k + \mathbf{b}_{X,k}(\mathbf{x} - \mathbf{x}_k)^\top + b_{\Theta,k}(\theta - \theta_k). \quad (5)$$

Each f_k estimates its parameter $\alpha_k = (y_k, \mathbf{b}_k, \mathbf{w}_k)$ in light of minimizing the EPE:

$$\alpha_k^* = \arg \min_{y_k, \mathbf{b}_k, \mathbf{w}_k} \mathbb{E}[(y - y_k - \mathbf{b}_k([\mathbf{x}, \theta] - \mathbf{w}_k)^\top)^2] \quad (6)$$

subject to $y_k = f_k(\mathbf{x}_k, \theta_k)$ and $[\mathbf{x}, \theta] \in \mathbb{Q}_k$.

B. Linear Predictive Modeling

We propose a joint optimization problem of incrementally identifying within-subspaces linearities and then estimating therein the LLM coefficients. Firstly, we should identify the subspaces \mathbb{Q}_k , i.e., determine their prototypes \mathbf{w}_k , their number K , and their coefficients y_k and \mathbf{b}_k , in which f can be well approximated by LLMs. We identify the prototypes \mathbf{w}_k (associated with $\mathbb{Q}_k, k \in [K]$) by incrementally partitioning the query space $\mathbb{Q} = \cup_{k=1}^K \mathbb{Q}_k$.

Example 1: Figure 3 (left) shows 1,000 issued queries $\mathbf{q}_t = [\mathbf{x}_t, \theta_t]$ over the 2D input space $\mathbf{x} = (x_1, x_2) \in [-1.5, 1.5]^2$. Each query is represented by a disc with center \mathbf{x}_t and radius θ_t . Figure 3 (right) shows the five query prototypes $\mathbf{w}_k = [\mathbf{x}_k, \theta_k], k \in [5]$ projected onto the 2D input space. Note, centers \mathbf{x}_k of the prototypes \mathbf{w}_k correspond to Voronoi sites under L_2 onto the data space.

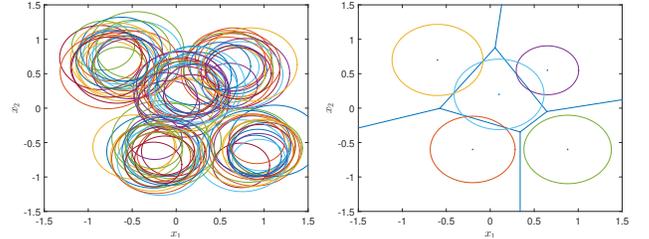


Fig. 3. Example 1. (left) 2D representation of queries and (right) their query prototypes onto the input space $[-1.5, +1.5]^2$.

In parallel, within each \mathbb{Q}_k , we incrementally earn the coefficients (y_k, \mathbf{b}_k) of each f_k . These coefficients are learned *only* from similar query-answer pairs whose queries belong to \mathbb{Q}_k . We propose a hybrid model by (Task 1) partitioning \mathbb{Q} into K (unknown) subspaces \mathbb{Q}_k (unsupervised leaning of \mathbf{w}_k) and, (Task 2) supervised learning of the coefficients y_k and \mathbf{b}_k . Each query subspace \mathbb{Q}_k associates f_k of the output space with \mathbf{w}_k of the input space, as shown in Figure 4 (see Example 2). Both joint optimization Tasks 1 and 2 incrementally minimize two objective functions: the EQE \mathcal{J} and EPE \mathcal{H} upon receiving a new query-answer pair (\mathbf{q}, y) :

$$\mathcal{J}(\{\mathbf{w}_k\}) = \mathbb{E} \left[\min_k \|\mathbf{q} - \mathbf{w}_k\|_2^2 \right], \quad (7)$$

$$\mathcal{H}(\{y_k, \mathbf{b}_k\}) = \mathbb{E} \left[(y - y_j - \mathbf{b}_j(\mathbf{q} - \mathbf{w}_j)^\top)^2 \Big| j \right] \text{ s.t. } j = \arg \min_k \|\mathbf{q} - \mathbf{w}_k\|_2 \text{ and } y_k = f_k(\mathbf{x}_k, \theta_k), \forall k \quad (8)$$

Objective function (7) corresponds to partitioning the query space into K partitions, each with a prototype. Objective function (8) corresponds to an EPE **conditioned** on the j -th prototype \mathbf{w}_j , which is the closest to \mathbf{q} .

The quantization of \mathbb{Q} operates as a mechanism to project an unseen query \mathbf{q} to the closest query subspace \mathbb{Q}_k w.r.t. L_2 distance from prototype \mathbf{w}_k , wherein we learn the dependency between y with \mathbf{x} and θ . Hence, concerning Problem 1, the prediction \hat{y} of an unseen query \mathbf{q} is provided by (neighboring) LLMs f_k , as will be elaborated later. Concerning Problem

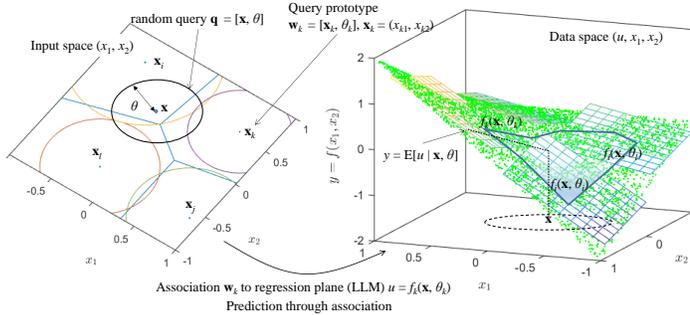


Fig. 4. Examples 2 & 3. Association of a query prototype \mathbf{w}_j with the LLM f_j in the 3D data space (u, x_1, x_2) with data function $u = g(x_1, x_2) = x_1(x_2 + 1)$.

2, we first derive the linear dependency (intercept and slope) between u and \mathbf{x} over subspace \mathbb{D} given an LLM. Then, we approximate g by interpolating LLMs. Based on Theorems 1 & 2 we have that $u = g(\mathbf{x}) = \mathbb{E}[u|\mathbf{x}] + \epsilon$. In that context, we can approximate $g(\mathbf{x})$ over \mathbb{D}_k directly from its LLM f_k .

Theorem 3: The data function $g(\mathbf{x})$ in \mathbb{D}_k is approximated by the linear regression function:

$$u = g(\mathbf{x}) \approx y_k + \mathbf{b}_{X,k}(\mathbf{x} - \mathbf{x}_k)^\top = f_k(\mathbf{x}, \theta_k),$$

with slope: $\mathbf{b}_{X,k}$ and intercept $y_k - \mathbf{b}_{X,k}\mathbf{x}_k^\top$.

Proof: For proof, see Appendix I. ■

Example 2: Figure 4 depicts the *association* from the query space to the 3D data space. A query prototype \mathbf{w}_j , a disc on the input space (x_1, x_2) , is now associated with LLM $f_j(\mathbf{x}, \theta)$ and its corresponding regression plane $u_j = f_j(\mathbf{x}, \theta_j)$ on data space (u, x_1, x_2) , which approximates the actual data function $u = g(x_1, x_2) = x_1(x_2 + 1)$. Note, in each *local* plane, we learn the local intercept y_j and slope \mathbf{b}_j where \mathbf{x}_j is the representative of data subspace \mathbb{D}_j (see Theorems 7, 8, 9).

It is worth noting that the data function g based on Theorem 3 is approximated only by executed queries. In the following, we propose a query processing algorithm through which all LLM parameters α_k minimize both (7) and (8). Then, we propose an algorithm to predict the output y of an unseen query based on LLMs. Finally, we propose an algorithm that utilizes LLMs to approximate the function g over a data subspace.

IV. QUERY-BASED MODEL TRAINING

Let us focus on EQE \mathcal{J} in (7) and liaise with Example 1 (Figure 3). We seek the best possible approximation (in L_2)

of a random query \mathbf{q} out of the set $\{\mathbf{w}_k\}_{k=1}^K$ of (finite) K query prototypes. We consider the closest neighbor projection of \mathbf{q} to a prototype \mathbf{w}_j , which represents the j -th subspace $\mathbb{Q}_j \subset \{\mathbf{q} \in \mathbb{Q} : \|\mathbf{q} - \mathbf{w}_j\|_2 = \min_k \|\mathbf{q} - \mathbf{w}_k\|_2\}$. Based on AVQ, we incrementally minimize \mathcal{J} with the presence of a random query \mathbf{q} and update the winning prototype \mathbf{w}_j . However, the number of query subspaces (and, thus, prototypes) $K > 0$ is completely unknown and not necessarily constant. The key problem is to decide on an appropriate K value. In the literature a variety of AVQ methods exists however not suitable for incremental implementation, because K must be supplied in advance.

We propose a *conditionally growing* AVQ algorithm under L_2 in which the prototypes are (i) sequentially updated with the incoming queries and (ii) their number is adaptively growing, i.e., K increases if a criterion holds true. Given that K is not available a-priori, our quantization algorithm minimizes \mathcal{J} with respect to a threshold value ρ . This threshold determines the current number of prototypes K . Initially, the query space has a unique (random) prototype, i.e., $K = 1$. Upon the presence of a query \mathbf{q} , our algorithm (i) finds the winning \mathbf{w}_j and (ii) updates \mathbf{w}_j only if the condition $\|\mathbf{q} - \mathbf{w}_j\|_2 \leq \rho$ holds true. Otherwise, \mathbf{q} is currently considered as a *new* prototype, thus, increasing K by one. Through this conditional quantization, our algorithm leaves the random queries to self-determine the resolution of quantization. Evidently, a high ρ value would result in coarse query space quantization (low resolution) while low ρ values yield a fine-grained quantization. Parameter ρ is associated with the stability-plasticity dilemma a.k.a. *vigilance* in Adaptive Resonance Theory [13]. In our case, the vigilance ρ represents a threshold of similarity between queries and prototypes, thus, guiding our algorithm in determining whether a new prototype should be formed.

To give a physical meaning to ρ , we express it through a set of percentages $a_i \in (0, 1)$ and $a_\theta \in \theta$ of the value ranges of each dimension x_i of $\mathbf{x} \in \mathbb{R}^d$ and θ , respectively. Then, $\rho = \|[a_1, \dots, a_d]\|_2 + a_\theta$ and if we let $a_i = a_\theta = a, \forall i$, then $\rho = a(d^{1/2} + 1)$. A high a value over high dimensional data results in a low number of prototypes and vice versa.

Let us focus on EPE \mathcal{H} in (8) and liaise with Examples 1 & 2 (Figure 3 & 4). \mathcal{H} is conditioned on the winning query-prototype index $j = \arg \min_k \|\mathbf{q} - \mathbf{w}_k\|_2$. We incrementally learn the LLM coefficients y_j and \mathbf{b}_j of f_j , which associate with the winning prototype $\mathbf{w}_j \in \mathbb{Q}_j$ for a random \mathbf{q} . We incrementally minimize both \mathcal{J} and \mathcal{H} given a series of issued (\mathbf{q}_t, y_t) to estimate the unknown parameters set $\alpha = \cup_{k=1}^K \alpha_k$, with $\alpha_k = (y_k, \mathbf{b}_k, \mathbf{w}_k)$ through SGD. Our algorithm processes successive pairs (\mathbf{q}_t, y_t) until a termination criterion $\max(\Gamma_t^{\mathcal{J}}, \Gamma_t^{\mathcal{H}}) \leq \gamma$. Specifically, $\Gamma_t^{\mathcal{J}}$ and $\Gamma_t^{\mathcal{H}}$ refer to the distance between successive estimates at steps $t-1$ and t of prototypes w.r.t. \mathcal{J} and LLM coefficients w.r.t. \mathcal{H} , respectively. The algorithm stops at the first step t where $\max(\Gamma_t^{\mathcal{J}}, \Gamma_t^{\mathcal{H}}) \leq \gamma$, with: $\Gamma_t^{\mathcal{J}} = \sum_{k=1}^K \|\mathbf{w}_{k,t} - \mathbf{w}_{k,t-1}\|_2$ and $\Gamma_t^{\mathcal{H}} = \sum_{k=1}^K \|\mathbf{b}_{k,t} - \mathbf{b}_{k,t-1}\|_2 + |y_{k,t} - y_{k,t-1}|$. The update rules for α in our SGD dual EQE/EPE optimization of \mathcal{J} and \mathcal{H} objective functions are provided in Theorem 4.

Theorem 4: Given a pair query-answer (\mathbf{q}, y) and its winning prototype \mathbf{w}_j , the parameter α converges to the optimal

α^* if updated as: If $\|\mathbf{q} - \mathbf{w}_j\|_2 \leq \rho$ then

$$\begin{aligned}\Delta \mathbf{w}_j &= \eta(\mathbf{q} - \mathbf{w}_j) \\ \Delta \mathbf{b}_j &= \eta(y - y_j - \mathbf{b}_j(\mathbf{q} - \mathbf{w}_j)^\top)(\mathbf{q} - \mathbf{w}_j) \\ \Delta y_j &= \eta(y - y_j - \mathbf{b}_j(\mathbf{q} - \mathbf{w}_j)^\top).\end{aligned}$$

If $\|\mathbf{q} - \mathbf{w}_j\|_2 > \rho$ then $\Delta \mathbf{w}_j = \mathbf{0}, \Delta \mathbf{b}_j = \mathbf{0}, \Delta y_j = 0$ For any prototype \mathbf{w}_k , which is not winner ($k \neq j$):

$$\Delta \mathbf{w}_k = \mathbf{0}, \Delta \mathbf{b}_k = \mathbf{0}, \Delta y_k = 0,$$

where the learning rate $\eta \in (0, 1)$ is defined in Section II-B.

Proof: For proof, see Appendix I. \blacksquare

The training Algorithm 1 processes a (random) pair of query-answer one at a time from a training set \mathcal{T} ; see Figure 2. In the initialization phase, there is only one prototype \mathbf{w}_1 , i.e., $K = 1$, which corresponds to the first query, while the associated LLM coefficients \mathbf{b}_1 and y_1 are initialized to $\mathbf{0}$ and 0 , respectively. For the t -th random pair (\mathbf{q}_t, y_t) and onwards with $t \geq 2$, the algorithm: (i) updates the closest prototype to \mathbf{q}_t (out of the so far K prototypes) if their L_2 distance is less than ρ , otherwise (ii) a *new* prototype is added (increasing K by one) and *new* LLM coefficients are initialized. The algorithm stops updating the prototypes and LLM coefficients at the first step t where $\max(\Gamma_t^{\mathcal{J}}, \Gamma_t^{\mathcal{H}}) \leq \gamma$. At that time and onwards, the algorithm returns the parameters set α and no further modification is performed.

Input: vigilance ρ , convergence threshold γ

Result: LLM parameters set α

begin

Get first pair (\mathbf{q}, y) ;

Init.: $\alpha = \{(y_1 = 0, \mathbf{b}_1 = \mathbf{0}, \mathbf{w}_1 = \mathbf{q})\}, K \leftarrow 1$;

repeat

Get next pair (\mathbf{q}, y) ;

Find closest prototype $j = \arg \min_k \|\mathbf{w}_k - \mathbf{q}\|_2$;

if $\|\mathbf{w}_j - \mathbf{q}\|_2 \leq \rho$ **then**

 | *Update* $y_j, \mathbf{b}_j, \mathbf{w}_j$ using Theorem 4.

else

 | $K \leftarrow K + 1$;

 | *Initialize* $(y_K, \mathbf{b}_K) = (0, \mathbf{0}), \mathbf{w}_K \leftarrow \mathbf{q}$;

 | $\alpha \leftarrow \alpha \cup \{(y_K, \mathbf{b}_K, \mathbf{w}_K)\}$;

end

 | *Calculate* $\Gamma^{\mathcal{J}}, \Gamma^{\mathcal{H}}$;

until $\max(\Gamma^{\mathcal{J}}, \Gamma^{\mathcal{H}}) \leq \gamma$;

end

Algorithm 1: Training Algorithm.

Through the incremental training of parameters set $\alpha = \{(y_k, \mathbf{b}_k, \mathbf{w}_k)\}_{k=1}^K$, each LLM f_k has estimated its triplets of parameters. At prediction phase, no change is done in the parameter set α . The approximation error bound for f_k around the prototype \mathbf{w}_k depends on the dimension d and curvature (second derivative) of f_k in \mathbb{Q}_k as provided in Theorem 5. The approximation depends on the resolution of quantization K ; the more prototypes K (see Theorem 6), the better the approximation of f by LLMs.

Theorem 5: For a random query \mathbf{q} with closest prototype \mathbf{w}_k , the conditional expected approximation error bound for

f_k in \mathbb{Q}_k around \mathbf{w}_k is: $\mathbb{E}[|f(\mathbf{x}, \theta) - f_k(\mathbf{x}, \theta)| | \mathbf{w}_k] \leq C_k O(d)$ with $C_k \geq \frac{1}{2} \max_{i \in [d+1]} \left| \frac{\partial^2 f(\mathbf{q})}{\partial q_i^2} \right|_{\mathbf{q}=\mathbf{w}_k}$.

Proof: For proof, see Appendix I. \blacksquare

Theorem 6: For a random query \mathbf{q} , the expected approximation error given K LLMs $f_k, k \in [K]$ is bounded by $\sum_{k \in [K]} C_k O(\frac{d}{K})$, where C_k is defined in Theorem 5.

Proof: For proof, see Appendix I. \blacksquare

V. QUERY PROCESSING ALGORITHMS

Our query processing entails the use of the previously trained LLMs from training (query-response) pairs in \mathcal{T} to predict answers to unseen Q1 and Q2 from set \mathcal{V} ; see Figure 2. We adopt the principle of nearest neighbors regression for prediction; the notion of neighborhood here is materialized by the overlapping of an unseen query with the prototypes in the quantized space \mathbb{Q} (see Example 3, Figure 4). By Definition 6, $\mathbf{q} = [\mathbf{x}, \theta]$ and $\mathbf{q}' = [\mathbf{x}', \theta']$ overlap if $\mathcal{A}(\mathbf{q}, \mathbf{q}') = \text{TRUE}$. To quantify a degree of overlapping between those hyper-spheres, we require that the two balls are partially intersected. Let us define the ratio between the L_2 distance of the centers of data subspaces $\mathbb{D}(\mathbf{x}, \theta)$ and $\mathbb{D}(\mathbf{x}', \theta')$ over the distance of their radii, i.e., $\frac{\|\mathbf{x} - \mathbf{x}'\|_2}{\theta + \theta'}$. This ratio takes values in $[0, 1]$ in the case of overlapping, with a value of unity when both spheres just meet each other. In the concentric case, the degree of overlapping should also take into consideration the remaining area from this perfect inclusion. We define the *degree of overlapping* for two queries as the normalized ratio $\delta(\mathbf{q}, \mathbf{q}') \in [0, 1]$:

$$\delta(\mathbf{q}, \mathbf{q}') = \begin{cases} 1 - \frac{\max(\|\mathbf{x} - \mathbf{x}'\|_2, |\theta - \theta'|)}{\theta + \theta'}, & \text{if } \mathcal{A}(\mathbf{q}, \mathbf{q}') = \text{TRUE} \\ 0, & \text{if } \mathcal{A}(\mathbf{q}, \mathbf{q}') = \text{FALSE}. \end{cases} \quad (9)$$

The subspaces $\mathbb{D}(\mathbf{x}, \theta)$ and $\mathbb{D}(\mathbf{x}_k, \theta_k)$ defined by query \mathbf{q} and prototype $\mathbf{w}_k = [\mathbf{x}_k, \theta_k]$, respectively, correspond to the highest overlap when $\delta(\mathbf{q}, \mathbf{w}_k) = 1$. We define the *overlapping* prototypes set $\mathcal{W}(\mathbf{q})$ of subspaces \mathbb{Q}_k corresponding to data subspaces \mathbb{D}_k given a query $\mathbf{q} = [\mathbf{x}, \theta]$:

$$\mathcal{W}(\mathbf{q}) = \{\mathbf{w}_k = [\mathbf{x}_k, \theta_k] : \delta(\mathbf{q}, \mathbf{w}_k) > 0\}. \quad (10)$$

The average value (Q1) and linear regression query (Q2) are based on the neighborhood $\mathcal{W}(\mathbf{q})$ for unseen query \mathbf{q} .

Example 3: Figure 4 shows the average value and regression query prediction: An unseen $\mathbf{q} = [\mathbf{x}, \theta]$ is projected onto input space $\mathbf{x} = (x_1, x_2)$ to derive the neighborhood $\mathcal{W}(\mathbf{q}) = \{\mathbf{w}_i, \mathbf{w}_k, \mathbf{w}_l\}$. Then, we access the LLMs f_i, f_k, f_l to predict \hat{y} for Q1 (Algorithm 2) and retrieve the regression planes coefficients \mathcal{S} of f_i, f_k, f_l for Q2 (Algorithm 3).

A. Query Q1: Average Value Prediction

Our algorithm predicts the average value y given an unseen query $\mathbf{q} = [\mathbf{x}, \theta]$ over a data subspace $\mathbb{D}(\mathbf{x}, \theta)$. The function f between \mathbf{q} and y over \mathbb{Q} is approximated by K local hyperplanes over each \mathbb{Q}_k ; see Figure 5 (right). Given a query \mathbf{q} , we derive the overlapping prototypes set $\mathcal{W}(\mathbf{q})$. For those prototypes $\mathbf{w}_k \in \mathcal{W}(\mathbf{q})$, we access the local coefficients $(y_k, \mathbf{b}_k, \mathbf{w}_k)$ of LLM f_k . Then, we pass $\mathbf{q} = [\mathbf{x}, \theta]$ as input to

each f_k to predict the average value \hat{y} through a weighted average based on the normalized degrees of overlapping $\tilde{\delta}(\mathbf{q}, \mathbf{w}_k)$, which is $\tilde{\delta}(\mathbf{q}, \mathbf{w}_k) = \frac{\delta(\mathbf{q}, \mathbf{w}_k)}{\sum_{\mathbf{w}_k \in \mathcal{W}(\mathbf{q})} \delta(\mathbf{q}, \mathbf{w}_k)}$. The prediction \hat{y} derives from the weighted $\mathcal{W}(\mathbf{q})$ -nearest neighbors regression:

$$\hat{y} = \sum_{\mathbf{w}_k \in \mathcal{W}(\mathbf{q})} \tilde{\delta}(\mathbf{q}, \mathbf{w}_k) f_k(\mathbf{x}, \theta), \quad (11)$$

$$f_k(\mathbf{x}, \theta) = y_k + \mathbf{b}_{X,k}(\mathbf{x} - \mathbf{x}_k)^\top + b_{\Theta,k}(\theta - \theta_k). \quad (12)$$

In the case where $\mathcal{W}(\mathbf{q}) \equiv \emptyset$, we extrapolate the similarity of the query \mathbf{q} with the closest prototype to associate the answer with the estimation \hat{y} derived only from the $f_j(\mathbf{q}, \theta)$ with \mathbf{w}_j being closest to \mathbf{q} . Through this projection, i.e., $j = \arg \min_{k \in [K]} \|\mathbf{q} - \mathbf{w}_k\|_2$, we get the local slope and intercept of the local mapping of \mathbf{q} onto y .

The prediction of the query answer depends entirely on the query similarity and \mathcal{W} neighborhood. The average value prediction algorithm is shown in Algorithm 2. Figure 5 (right) shows how accurately the $K = 7$ LLMs (as green covering surfaces/planes over f) approximate the linear parts of $f(\mathbf{x}, \theta)$ over 2D query space \mathbb{Q} defined by (x, θ) .

Input: unseen query $\mathbf{q} = [\mathbf{x}, \theta]$
Result: average prediction \hat{y}
begin
 Calculate overlapping set $\mathcal{W}(\mathbf{q})$ using (10);
 if $\mathcal{W}(\mathbf{q}) \equiv \emptyset$ **then**
 Find closest prototype $j = \arg \min_k \|\mathbf{w}_k - \mathbf{q}\|_2$;
 Predict $\hat{y} = f_j(\mathbf{q}, \theta)$ using (12);
 else
 Calculate $\tilde{\delta}(\mathbf{q}, \mathbf{w}_k)$, $\mathbf{w}_k \in \mathcal{W}(\mathbf{q})$ using (9);
 Predict \hat{y} using (11) and (12);
 end
end

Algorithm 2: Q1 Query Processing.

B. Query Q2: Linear Regression Query

The algorithm returns a list of local linear regressions of the underlying function g over data subspace $\mathbb{D}(\mathbf{x}, \theta)$, given an unseen query $\mathbf{q} = [\mathbf{x}, \theta]$ (see Example 3). An unseen \mathbb{D} might either (Case 1) partially overlap with several identified convex data subspaces \mathbb{D}_k (corresponding to query subspaces \mathbb{Q}_k) or (Case 2) be contained or contain a \mathbb{D}_k , or (Case 3) be outside of any \mathbb{D}_k . In Cases 1 and 2, the algorithm returns the derived local linear regressions of g interpolating over the overlapping data subspaces, using the corresponding LLMs, as proved in Theorem 3. In Case 3, the best possible linear approximation is returned through extrapolation of the data subspace \mathbb{D}_k whose prototype \mathbf{w}_k is closest to the query \mathbf{q} . For Cases 1 and 2 we exploit the $\mathcal{W}(\mathbf{q})$ of $\mathbf{q} = [\mathbf{x}, \theta]$. For Case 3, we select the closest \mathbf{w}_j to \mathbf{q} since, in this case, $\mathcal{W}(\mathbf{q}) \equiv \emptyset$.

The approximation of $g(\mathbf{x})$ over $\mathbb{D}(\mathbf{x}, \theta)$ involves both the radius θ and the center \mathbf{x} using their similarity with θ_k and the point \mathbf{x}_k , respectively from $\mathcal{W}(\mathbf{q})$. For the Cases 1 and 2, the set of the local linear approximations of $g(\mathbf{x})$ is provided directly from those LLMs f_k , whose $\mathbf{w}_k \in \mathcal{W}(\mathbf{q})$. That is, for $\mathbf{x} \in \mathbb{D}_k(\mathbf{x}_k, \theta_k)$, we obtain:

$$u = g(\mathbf{x}) = f_k(\mathbf{x}, \theta_k) \approx y_k + \mathbf{b}_{X,k}(\mathbf{x} - \mathbf{x}_k)^\top, \quad (13)$$

$\forall \mathbf{w}_k \in \mathcal{W}(\mathbf{q})$, where the u intercept in \mathbb{D}_k is: $y_k - \mathbf{b}_{X,k} \mathbf{x}_k^\top$ and the u slope in \mathbb{D}_k is: $\mathbf{b}_{X,k}$. For the Case 3, the local linear approximation of $g(\mathbf{x})$ derives by extrapolating the linearity trend of $u = g(\mathbf{x}) = f_j(\mathbf{x}, \theta_j) : j = \arg \min_k \|\mathbf{q} - \mathbf{w}_k\|_2$ over the data subspace, with u intercept: $y_j - \mathbf{b}_{X,j} \mathbf{x}_j^\top$ and the u slope: $\mathbf{b}_{X,j}$. The data function coefficients prediction algorithm is shown in Algorithm 3. Figure 5 (left) shows how function $u = g(x)$ is accurately approximated by $K = 6$ LLMs (green interpolating local lines) and the global linear regression function (REG in red) over the subspace $\mathbb{D}(0.5, 0.5)$. We also illustrate the K linear models derived by Piece-wise Linear Regression (PLR). **Note:** Unlike our model, PLR, needs access to the data and is thus very expensive; specifically, it involves a forward/backward iterative approach to produce the multiple linear models [21]. Our model, instead, incrementally derives the LLMs, based on the optimization problems in (7) and (8). Note that the predicted LLMs are highly accurate.

Input: unseen query $\mathbf{q} = [\mathbf{x}, \theta]$
Result: set of local linear (intercepts, slopes) \mathcal{S}
begin
 $\mathcal{S} \leftarrow \{\}$;
 Calculate overlapping set $\mathcal{W}(\mathbf{q})$ using (10);
 if $\mathcal{W}(\mathbf{q}) \equiv \emptyset$ **then**
 Find closest prototype $j = \arg \min_k \|\mathbf{w}_k - \mathbf{q}\|_2$;
 Get coefficients of $u = g(\mathbf{x}) = f_j(\mathbf{x}, \theta_j)$;
 $\mathcal{S} = \{(y_j - \mathbf{b}_{X,j} \mathbf{x}_j^\top, \mathbf{b}_{X,j})\}$;
 else
 foreach $\mathbf{w}_k \in \mathcal{W}(\mathbf{q})$ **do**
 Get coefficients of $u = g(\mathbf{x}) = f_k(\mathbf{x}, \theta_k)$;
 $\mathcal{S} \leftarrow \mathcal{S} \cup \{(y_k - \mathbf{b}_{X,k} \mathbf{x}_k^\top, \mathbf{b}_{X,k})\}$;
 end
 end
end

Algorithm 3: Q2 Query Processing.

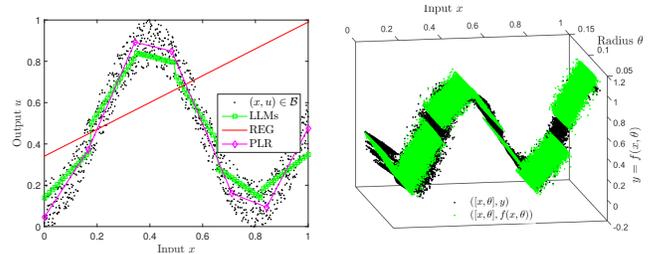


Fig. 5. (Left) The $K = 6$ LLMs $f_k(x, \theta_k) \approx g(x)$, a PLR approximation with $K = 6$ linear models, and a global linear approximation (REG) of $u = g(x)$ over a 2D data subspace \mathbb{D} ; (right) the $y = f(x, \theta)$ approximated by $K = 7$ LLMs $f_k(x, \theta)$ over 3D query space \mathbb{Q} .

Convergence & Complexity: Given a Q1 and a Q2, we require $O(dK)$ to calculate the \mathcal{W} set and deliver the LLMs, respectively, i.e., independent on the data size. We require $O(dK)$ space to store the prototypes and LLM coefficients; Appendix II reports on convergence theorems 7, 8, and 9.

VI. PERFORMANCE EVALUATION

Performance Metrics: Being a PMA model, we should assess the **predictability** and **goodness-of-fit** of our model. Predictability refers to the capability of a model to *predict*

an output given an unseen input, i.e., not provided during the model’s training phase. Measures of prediction focus on the differences between **values predicted** and values actually observed. Goodness-of-fit describes how well a model *fits* a set of observations, which were provided in the model’s training phase. Measures of goodness of fit summarize the discrepancy between actual/observed values during training and the **values approximated** under the model in question. We compare our model against its ground truth counterparts: the multivariate linear regression model (over data subspaces), hereafter referred to as REG and the Piecewise Linear Model (PLR), both of which have full access to the data. We demonstrate how effectively LLM *approximates* the ground truth models. Specifically, we compare against (i) REG using RDBMS PostgreSQL and the Matlab and (ii) PLR using the ARESLab (Matlab) toolbox⁴ for building PLR models based on the multivariate adaptive regression splines method in [21]. We show that our model is scalable and efficient and as (or even more than) accurate than REG, w.r.t. predictability and goodness-of-fit, and close to the accuracy obtained by PLR. However, our model is dramatically more scalable and efficient as, unlike REG and PLR, it does not need access to data, yielding up to 6 orders of magnitude faster query execution.

Predictability: Mean Value Accuracy (A1) refers to the prediction of the average value \hat{y} given unseen Q1 queries. Based on EPE (2), A1 is the Root Mean-Square Error (RMSE) $e = (\frac{1}{M} \sum_{i=1}^M (y_i - \hat{y}_i)^2)^{1/2}$, where y and \hat{y} is the actual and the predicted average value of u from Algorithm 2 given M unseen Q1 queries. **Data Value Accuracy (A2)** refers to the prediction of the data output \hat{u} given an unseen input \mathbf{x} . Here, LLM is exploited to predict u by approximating $g(\mathbf{x})$ as in (13) by aggregation of neighboring LLMs $f_k(\mathbf{x}, \theta_k)$. Let u and \hat{u} be the actual and the predicted value of $g(\mathbf{x})$ given M unseen points \mathbf{x} . Based on (11) and (13) we predict \hat{u} as:

$$\hat{u} = \sum_{\mathbf{w}_k \in \mathcal{W}(\mathbf{q})} \tilde{\delta}(\mathbf{q}, \mathbf{w}_k) f_k(\mathbf{x}, \theta_k). \quad (14)$$

$f_k(\mathbf{x}, \theta_k)$ provides the intercept y_k and slope $\mathbf{b}_{\mathbf{x},k}$ over the input space by setting $\theta = \theta_k$ in f_k . The A2 is then $v = (\frac{1}{M} \sum_{i=1}^M (u_i - \hat{u}_i)^2)^{1/2}$.

Goodness-of-fit: Given an unseen Q2 with $\mathbb{D}(\mathbf{x}, \theta)$, we evaluate how well our method approximates the data function g comparing with REG and PLR, over the same \mathbb{D} . For *goodness of fit* we adopt **Fraction of Variance Unexplained (FVU)** s and **Coefficient of Determination (CoD)** R^2 . FVU indicates the fraction of variance of the dependent variable u , which cannot be explained, i.e., which is not correctly predicted by the explanatory attributes \mathbf{x} . Given a subspace $\mathbb{D}(\mathbf{x}, \theta)$, consider the data $(\mathbf{x}_i, u_i) : \mathbf{x}_i \in \mathbb{D}$, and approximations \hat{u}_i for each \mathbf{x}_i . The Sum of Squared Residuals (SSR) and the Total Sum of Squares (TSS) over $\mathbb{D}(\mathbf{x}, \theta)$ are: $SSR = \sum_i (u_i - \hat{u}_i)^2$ and $TSS = \sum_i (u_i - \bar{u})^2$, respectively, where \bar{u} is the average value $\bar{u} = \frac{1}{n_\theta(\mathbf{x})} \sum_i u_i$, $i \in [n_\theta(\mathbf{x})]$. The FVU and CoD are defined as: $s = \frac{SSR}{TSS}$ and $R^2 = 1 - s$, respectively. FVU indicates how closely the approximation of g over \mathbb{D} matches the actual g . If FVU is greater than 1, the explanatory variables \mathbf{x} do not convey any information about u in the sense that the predictions \hat{u} do not covary with the actual u . In this case,

the approximation function is a *bad* fit. The approximation is considered *good* when FVU has a (low) value less than 1. Given an unseen \mathbf{q} over $\mathbb{D}(\mathbf{x}, \theta)$, we measure FVU and CoD for REG and PLR, and the average FVU $s = \frac{1}{|\mathcal{S}|} \sum_{\ell=1}^{|\mathcal{S}|} s_\ell$ of FVUs s_ℓ (and CoDs) corresponding to the set of local linear approximations $\mathcal{S} : |\mathcal{S}| \geq 1$ derived by our Algorithm 3.

A. Experimental Setup

Our goal is to evaluate both accuracy (predictability and goodness-of-fit) and efficiency/scalability. For accuracy, using A1, A2, FVU, and CoD metrics, we intentionally sought multivariate real data functions g to exhibit extreme non-linearity in many data subspaces. For this reason, to assess A1 for Q1 queries, A2 for data output predictions, and FVU/CoD for Q2 queries, we opted use the real dataset [18] with 6-dim. feature vectors and added extra vectors with Gaussian noise, thus, in total such dataset R1 contains $15 \cdot 10^6$ vectors. With R1 we wished to delve into accuracy issues and this dataset was chosen because its data exhibits non-linear relationships among features. All d -dim. real-valued vectors are scaled in $[0,1]$ ($d \in \{2, 3, 5\}$) with significant non-linear dependencies among the features, evidenced by a high FVU = 4.68. This indicates that a linear approximation of the entire data space is definitely to no avail, presenting a challenging test dataset for our approach. On the other hand, to evaluate scalability/efficiency along with accuracy, we now use a *big* synthetic dataset deriving from a benchmark function to ensure also significant non-linearity. The R2 synthetic dataset (u, \mathbf{x}) contains 10^{10} d -dim. real data generated by the Rosenbrock function [20] $u = g(\mathbf{x}) = \sum_{i=1}^{d-1} 100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2$, $\mathbf{x} = [x_1, \dots, x_d]$, attribute domain $|x_i| \leq 10$ and global minimum is 0 at $x_i = 1, \forall d$. Obviously, there is no linear dependency among features in the data space evidenced by a FVU = 12.45. In addition, we generate 10^{10} vectors adding noise $\epsilon \sim \mathcal{N}(0, 1)$ to each feature. R1 and R2 are stored in a PostgreSQL server with 2x Intel Xeon E5645, RAM 96 GB, HD: Seagate Constellation 1TB, 32MB cache. We use both R1 and R2 to assess (i) the Q1 prediction accuracy of y (A1 metric), corresponding to the average of the 6-th features in R1 and to the average u of the Rosenbrock in R2; (ii) the linear approximation(s) of g in R1 and R2 with $d \in \{2, 3, 5\}$ over Q2 (metrics: FVU, CoD); (iii) data value prediction accuracy (metric A2); (iv) the scalability and efficiency of our method compared against PostgreSQL with a B-tree index on input \mathbf{x} ($d \in \{2, 5\}$ over Q1, $d = 2$ over Q2) and Matlab ($d = 5$) over Q2 using the `regress` function on the server.

We generate training files \mathcal{T} and different testing files \mathcal{V} (for predictions) of various sizes: $|\mathcal{T}| \in \{10^3, \dots, 10^4\}$ and $M = |\mathcal{V}| \in \{10^3, \dots, 2 \cdot 10^4\}$, respectively, of pairs (\mathbf{q}, y) over the R1 and R2 (Figure 2). Random queries $\mathbf{q} = [\mathbf{x}, \theta]$ are generated with uniformly distributed centers $\mathbf{x} \in [0, 1]^d$ for R1 or in $[-10, 10]^d$ for R2. Radius θ affects the training time and the prediction quality. In brief, a larger (smaller) θ implies shorter (longer) training times. For each query, $\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$ is generated from a Gaussian distribution with

⁴<http://www.cs.rtu.lv/jekabsons/>

mean μ_θ , variance σ_θ^2 . We set $\theta \sim \mathcal{N}(0.1, 0.01)$ (R1) and $\sim \mathcal{N}(1, 0.25)$ (R2), covering $\sim 20\%$ in each feature data range; the justification for this setting is discussed later.

We train our model with \mathcal{T} and evaluate and compare it with the ground truths REG (ProstgreSQL & Matlab) and PLR (Matlab) with \mathcal{V} . The granularity of quantization for our model is tuned by the percentage coefficient $a \in [0.05, 1]$, involved in vigilance parameter $\rho = a(d^{1/2} + 1)$ (see Section IV); a value $a = 1$ corresponds to just one prototype, i.e., $K = 1$ (coarse quantization), while $a < 1$ (fine grained quantization) corresponds to a variable number of prototypes $K > 1$ depending on the underlying (unknown) data distribution. The default value for $a = 0.25$, convergence threshold $\gamma = 0.01$ (Algorithm 1), and learning rate schedule: $\eta_t = (1 + t)^{-1}$ [14]. To fairly compare against PLR, we set its maximum numbers of the automatically discovered linear models (in the forward building phase) to K and the generalized cross-validation penalty per PLR knot to 3 as suggested in [21].

B. Model Training

Figure 6 examines the termination criterion (of training Algorithm 1) $\Gamma = \max(\Gamma^{\mathcal{J}}, \Gamma^{\mathcal{H}})$ vs. number of training pairs in \mathcal{T} for $d \in \{2, 5\}$ over R1 and R2; $a = 0.25$. Training terminates at the first instance when $\Gamma \leq \gamma$, which is obtained for $|\mathcal{T}| \approx 5300$ training pairs. The total average training time, which includes (i) Q1 execution time and (ii) model updates time is (0.41, 2.38) hours for R1 and R2, respectively. This should not be perceived as overhead of our approach, as 99.62% of the training time is devoted to executing the queries over the RDBMS/statistical system, which we cannot avoid that anyway even in the typical case as shown in Figure 2. Any traditional approach would thus also pay 99.62% of this cost. This only affects how early our approach switches to using the trained model vs executing the queries against the system. Our experiments show that excellent quality results can be produced using a reasonable number of past queries for training. Obviously, this can be tuned by setting different convergence threshold γ values. We set $\gamma = 0.01$, where Γ is (stochastically) trapped around 0.0046, with deviation 0.0023 in R1; in R2, Γ is strictly less than γ for $|\mathcal{T}| > 5300$.

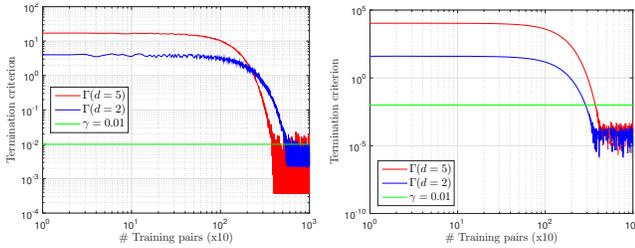


Fig. 6. Learning termination criterion $\Gamma = \max(\Gamma^{\mathcal{J}}, \Gamma^{\mathcal{H}})$ of Algorithm 1 vs. number of training pairs $|\mathcal{T}|$ for (left) R1 and (right) R2; $d \in \{2, 5\}$.

C. Query Evaluation

Q1 Evaluation: Figure 7 shows the RMSE of y (A1 metric) against the resolution a over R1 and R2 for Q1 queries from \mathcal{V} . For different a values, our model identifies subspaces where $f(\mathbf{x}, \theta)$ behaves almost linearly, thus, LLMs approximate such regions with high accuracy. Figure 10 (right) shows the number of prototypes required, e.g., $K = 450$ for $a = 0.25$: Only 450

prototypes are required to accurately predict y for $d = 5$. As $a \rightarrow 1$, then we quantize f into fewer linear approximations, thus, yielding higher RMSE. Figure 8 shows the robustness of the our model w.r.t. predictability with various testing file sizes $|\mathcal{V}|$ for both R1 and R2. Once LLM has converged, it provides a low and constant prediction error for different data dimensions. To assess efficiency and scalability for Q1, Figure 12 (left) shows in log scale the average Q1 execution time over R2 for LLM ($a = 0.25$) corresponding to $K = 92$ and $K = 450$ prototypes for $d \in \{2, 5\}$. Our method requires just 0.18 ms per query over massive data spaces, offering up to five orders of magnitude speedup (0.18ms vs up to 10^5 ms/query).

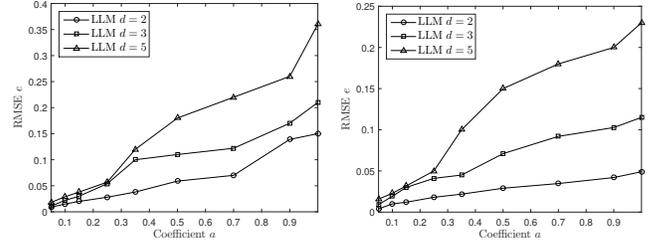


Fig. 7. Q1: RMSE of y of LLM vs. coefficient a over (left) R2 and (right) R1; $d \in \{2, 3, 5\}$.

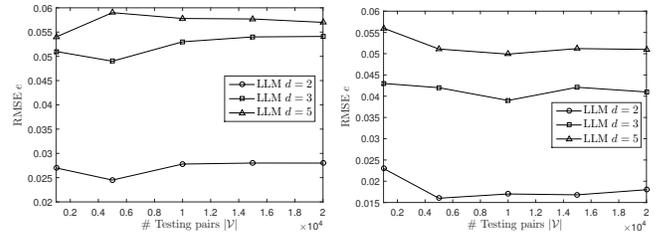


Fig. 8. Q1: RMSE of y of LLM vs. number of testing pairs ($|\mathcal{V}|$ size) over (left) R2 and (right) R1; $d \in \{2, 3, 5\}$, $a = 0.25$.

Q2 Evaluation: As shown in Figure 9 (left), $FVU < 1$ for our model while for REG, $FVU > 1$. As $a \rightarrow 1$, our model approaches the FVU of REG since, we force it to generate fewer LLMs; one LLM when $a = 1$. As expected, PLR achieves the lowest FVU, capturing the non-linearity with multiple linear basis functions. For $a < 1$, we achieve low FVU and capture the non-linearity of g by autonomously deriving multiple LLMs, which is very close to PLR for $a < 0.1$. This cannot be achieved by REG, since it cannot be expressed by a ‘global’ line within $\mathbb{D}(\mathbf{x}, \theta)$. PLR shows superior FVU performance, but while being dramatically inefficient compared to our model, as shown in Figure 12 (right).

In Figure 9 (right) g in R1 does not behave linearly in all the random data subspaces, since FVU for REG is relatively close to/over 1 for $d = 2$ and $d = 5$. This information is unknown a priori to analysts, hence results using REG would be fraught with approximation errors. It is worth noting that, the average number of LLMs that are returned to the analysts for all queries in \mathcal{V} is $|\mathcal{S}| = 4.62$ per query with variance 3.88. This denotes the non-linearity behavior of g and the fine grained and accurate explanation of g within a $\mathbb{D}(\mathbf{x}, \theta)$. Here, PLR achieves the lowest FVU, as expected, but note that this is also achieved by LLMs with $a < 0.1$.

Figure 10 (left) shows CoD R^2 for LLM, REG, and LLM

over R1 (similar results exist for R2). A positive value of R^2 close to 1 depicts that a linear approximation is a good fit for the unknown g . A value of 0, and especially, a negative value of R^2 indicates a *bad* fit signaling inaccuracy. With $K > 60$ prototypes, our model achieves high and positive R^2 , thus, better explains random $\mathbb{D}(\mathbf{x}, \theta)$ compared with the provided explanation of REG over exactly the same data subspaces. REG has low R^2 values (including negative ones), thus is inappropriate for predictions. As $a \rightarrow 1$, our model increases its R^2 , thus, providing more accurate, linear models. Again PLR achieves the highest CoD (at the cost of high insufficiency; see Figure 12(right)). Regardless, note that our model can catch PLR’s CoD value by increasing K .

Figure 12 (right) shows the Q2 execution time over R2 for LLM ($a = 0.25$, i.e., $K = (92, 450)$ for $d = (2, 5)$) through Algorithm 3, REG from PostgreSQL ($d = 2$) (REG-DBMS), REG from Matlab ($d = 5$) (REG-Matlab), and PLR against dataset size. Our model is highly scalable (note the flat curves) and highly efficient, achieving 0.56 ms/query (even for massive datasets)—up to six orders of magnitude better than REG and PLR. The full picture is then that our model provides ultimate scalability (being independent of the size of the dataset) and many orders of magnitude higher efficiency, while it ensures great goodness of fit (CoD, FVU), similar to that of PLR.

PLR with data sampling techniques could also be considered as an effective efficiency-accuracy trade-off. Figure 12, however, shows the efficiency limitations of such an approach. PLR (even over a very small random sample of size $10^6 = 0.01\%$ of the 10^{10} dataset) is shown to be > 3 orders of magnitude less efficient than our model. Also, recall that PLR here is implemented over Matlab (with all data in memory), hiding the performance costs of a full in-DBMS implementation for (i) the selection operator (computing the data subspace); (ii) the sampling of the data space; and (iii) the PLR algorithm (whose performance is shown in the figure). All of this is in stark contrast to the $O(1)$ cost of our model. Finally, note that, to our knowledge, PLR is not currently implemented within DBMSs, regardless of its cost.

Data Value Prediction: We compare LLM vs REG and PLR for providing accurate predictions w.r.t. A2. We use LLM for u predictions over unseen $\mathbb{D}(\mathbf{x}, \theta)$ using (13) and (11). Figure 11 shows the RMSE v for LLM, REG, and PLR over R1 and R2 vs. testing set size $|\mathcal{V}|$. LLM can successfully predict u by being robust in terms of $|\mathcal{V}|$ and assume comparable or, even, lower prediction error than REG. This denotes that our model, by fusing different LLMs, which better capture the characteristics of the data function, provides better u -prediction than a ‘global’ REG over subspace \mathbb{D} . PLR achieves the lowest RMSE by accessing the data and captures the non-linearity through its linear models. However, this is achieved with relatively high complexity, higher than REG. Note, the prediction times for LLM, REG, and PLR in this experiment are the same presented in Figure 12: LLM executes Algorithm 2 by replacing $\theta = \theta_k$ in (12), $\forall \mathbf{w}_k \in \mathcal{W}(\mathbf{q})$, REG creates the linear approximation over \mathbb{D} , and PLR adaptively finds the best linear models for data fitting in *each prediction*.

Impact of radius θ : We experiment with different mean values μ_θ of the radius $\theta \sim \mathcal{N}(\mu_\theta, \sigma_\theta^2)$ having a fixed variance σ_θ^2 to examine the impact on training, quality of prediction, and approximation. We examine the number of training pairs, $|\mathcal{T}|$,

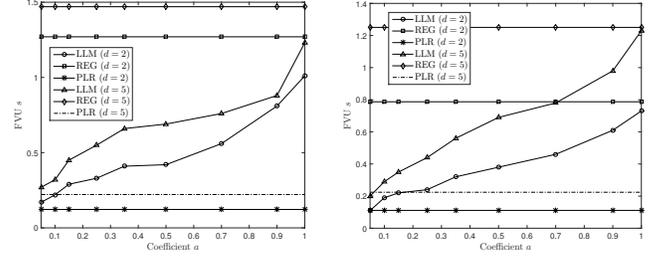


Fig. 9. Q2: FVU s of REG, PLR, and LLM vs. coefficient a over (left) R2 and (right) R1; $d \in \{2, 5\}$.

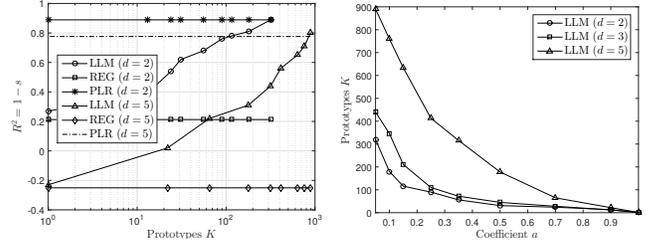


Fig. 10. Q2: (Left) CoD R^2 of LLM, PLR, and REG vs. prototypes K for R1; (right) Prototypes K vs. coefficient a over R1; $d \in \{2, 5\}$.

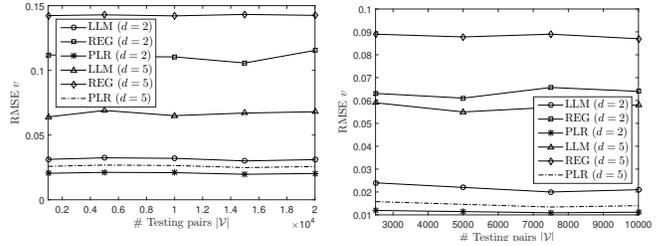


Fig. 11. Q2: RMSE v for data output u of LLM, PLR, and REG vs. number of testing pairs ($|\mathcal{V}|$ size) over (left) R2 and (right) R1; $d \in \{2, 5\}$, $a = 0.25$.

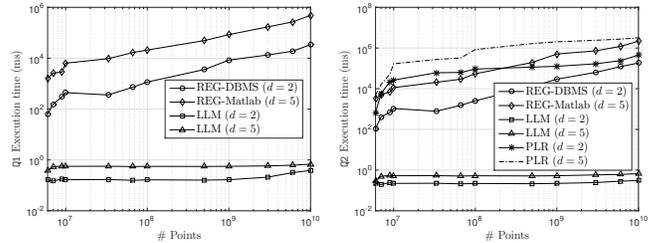


Fig. 12. Query execution time (ms) vs. # points for (left) Q1 and (right) Q2 for LLM, PLR, and REG over R2; $d \in \{2, 5\}$.

our method requires to reach the termination threshold $\gamma = 0.01$. We also examine the impact of θ on RMSE and CoD. Hence, three issues ($|\mathcal{T}|$, RMSE, and CoD) are influenced by θ . We experiment with $\mu_\theta \in \{0.01, \dots, 0.99\}$ over R1 (similar results are obtained in R2). Consider queries with high θ drawn from Gaussian $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$ with high μ_θ . Then, θ nearly covers the *entire* input data range and answer y is close to the average value of u for all queries, i.e., n_θ contains all \mathbf{x} input points. In this case, all prototypes \mathbf{w}_k correspond to constant lines $f_k(\mathbf{x}, \theta) \approx y_k = y$, where $y = \mathbb{E}[u]$ unconditioned to \mathbf{x} and θ . Hence, the training and convergence of all LLMs is *easy* since there is no any specificity to be extracted from each f_k . Our method converges with a low number of training pairs $|\mathcal{T}|$ as

shown in Figure 13 (right). On the other hand, a small θ refers to learning ‘meticulously’ all the specificities for all LLMs. In this case, our method requires a relatively high number of training pairs $|\mathcal{T}|$ to converge; see Figure 13 (right). In terms of accuracy, the higher the θ is, the lower the RMSE e becomes. With high θ , all LLMs refer to constant functions with the extreme case where $f_k \approx \mathbb{E}[u], \forall k$ as discussed above, thus, $e = \sqrt{\frac{1}{M} \sum (y_i - \hat{y}_i)^2} \rightarrow 0$ with $y_i \approx \mathbb{E}[u]$ due to the fact that n_θ contains all input data and, thus, $\hat{y}_i \approx \mathbb{E}[u]$ (see Figure 13 (left) with $|\mathcal{T}| = 5359$ training pairs required for convergence w.r.t. $\gamma = 0.01$). However, this comes at the expense of a low CoD R^2 since the data function g is approximated ‘solely’ by a constant $g(\mathbf{x}) \approx \mathbb{E}[u]$ (see Figure 13 (right)). When θ is small, we attempt to estimate f over (\mathbf{x}, θ) and, thus, approximate function $g(\mathbf{x})$. This, however, requires many training pairs $|\mathcal{T}|$; see Figure 13(right). Overall, there is a trade-off in the number of training pairs $|\mathcal{T}|$ with approximation & accuracy capability. To obtain quality approximation, CoD should be strictly greater than zero. This is achieved by setting $\mu_\theta < (0.4, 0.5)$ for $d \in (5, 2)$. Then, we can compensate RMSE and training time (number of training pairs $|\mathcal{T}|$) as shown in Figure 13. In addition, there is a trade-off between training effort and predictability. As shown in Figure 13 and explained above, low μ_θ results to high RMSE and training effort ($|\mathcal{T}|$ size). By combining those trade-offs, for a reasonable training effort, to achieve low RMSE and high goodness of fit (a high positive CoD), we set $\mu_\theta = 0.1$, which corresponds to $\sim 20\%$ of the data range for $\sigma_\theta^2 = 0.01$. Finally, Figure 14 shows the impact (trajectory) of μ_θ on the training set size $|\mathcal{T}|$, prediction accuracy w.r.t RMSE e , and goodness of fit w.r.t CoD R^2 for $d \in (2, 5)$ for R1; we obtain similar results for R2.

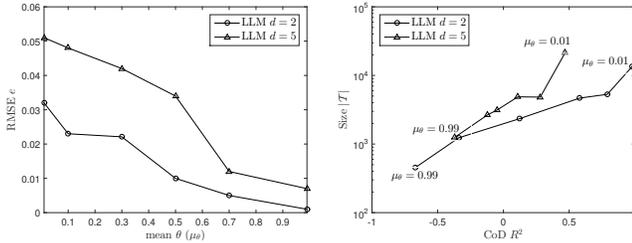


Fig. 13. Trade-off: (left) RMSE e vs. mean θ (μ_θ); $d \in \{2, 5\}$, $a = 0.25$; (right) Size $|\mathcal{T}|$ vs. CoD R^2 with μ_θ ; $d \in \{2, 5\}$, $a = 0.25$.

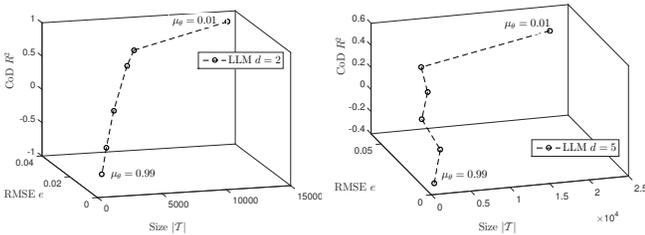


Fig. 14. Impact of μ_θ size $|\mathcal{T}|$, RMSE e , and CoD R^2 for $d = 2$ (left) and $d = 5$ (right) on R1; $a = 0.25$.

VII. CONCLUSIONS

We focused on average-value and linear-regression queries, which are central to in-DBMS analytics. We have begun a novel investigation route, whereby results from previously

executed queries are exploited to train novel models which predict future query results. Our approach yields highly accurate answers and data function approximation via multiple local linear regression models while being efficient and scalable. We contribute the training models and query processing algorithms. The performance evaluation revealed very promising results: Our model is shown to be (i) highly accurate, (ii) extremely efficient in computing query results (with sub-millisecond latencies even for massive datasets, yielding up to 6 orders of magnitude improvements compared to computing exact answers, and (iii) scalable, as predictions during query processing do not require access to the DBMS engine, thus being insensitive to dataset sizes. Our future agenda includes non-linear approximations, prediction of high-order moments, and adaptations to data space updates.

REFERENCES

- [1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, Q. Le, M. Mao, A. Senior, P. Tucker, K. Yang. ‘Large scale distributed deep networks’. In NIPS, 25, 1232–1240, 2012.
- [2] F. A. Niohaft, M. Massie, T. Danford, Z. Zhang, U. Laserson, C. Yeksigian, J. Kottalam, A. Ahuja, J. Hammerbacher, M. Linderman, M. J. Franklin, A. D. Joseph, D. A. Patterson. Rethinking dataintensive science using scalable analytics systems. In ACM SIGMOD 2015, 631–646.
- [3] S. Venkataraman, Z. Yang, M. Franklin, B. Recht, I. Stoica. 2016. Ernest: efficient performance prediction for large-scale advanced analytics. 13th USENIX NSDI’16, CA, USA, 363–378.
- [4] T. Hastie, R. Tibshirani, J. Friedman. (2001). *The Elements of Statistical Learning*. USA: Springer Inc.
- [5] XLeatorDB, URL: <http://www.westclintech.com/>
- [6] Oracle UTL_NLA, URL: https://docs.oracle.com/cd/B19306_01/appdev.102/b14258/u_nla.htm
- [7] MS SQL Server Clustering, <https://msdn.microsoft.com/en-us/library/cc280445.aspx>
- [8] D. S. TKach, ‘Information Mining with the IBM Intelligent Miner’. IBM White Paper. pp. 1–29, 1998.
- [9] D. Wong. (2013). ‘Oracle Data Miner’. Oracle White Paper, pp:1–31.
- [10] A. Thiagarajan, S. Madden. ‘Querying continuous functions in a database system’. ACM SIGMOD, pp. 791–804, 2008.
- [11] A. Deshpande, S. Madden. ‘MauveDB: supporting model-based user views in database systems’. ACM SIGMOD, pp. 73–84, 2006.
- [12] B. Ari, H. A. Gvenir. 2002. ‘Clustered linear regression’, Knowledge-Based Systems, 15(3):169–175.
- [13] G. Carpenter, S. Grossberg. (2003). ‘Adaptive Resonance Theory’, Handbook of Brain Theory and Neural Networks, pp.87–90. MIT Press.
- [14] L. Bottou. (2012) ‘Stochastic Gradient Tricks’. In Neural Networks, Tricks of the Trade, Springer, p. 430–445.
- [15] P. Schneider, B. Hammer, M. Biehl. (2009). ‘Adaptive Relevance Matrices in Learning Vector Quantization’, Neural Comp. 21:3532–3561.
- [16] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, C. Welton. 2009. MAD skills: new analysis practices for big data. VLDB Endow. 2(2):1481–1492.
- [17] X. Feng, A. Kumar, B. Recht, C. Re. 2012. Towards a unified architecture for in-RDBMS analytics. ACM SIGMOD ’12, 325–336.
- [18] I. Rodriguez-Lujan, J. Fonollosa, A. Vergara, M. Homer, R. Huerta. (2014) ‘On the calibration of sensor arrays for pattern recognition using the minimal number of experiments’, Chemometrics and Intelligent Laboratory Systems, 130:123–134.
- [19] M. Schleich, D. Olteanu, R. Ciucanu. (2016) ‘Learning Linear Regression Models over Factorized Joins’. ACM SIGMOD ’16, pp:3–18.
- [20] H. Rosenbrock, ‘An Automatic Method for Finding the Greatest or Least Value of a Function’, Computer J. 3:175–184, 1960.
- [21] J. H. Friedman. (1991) ‘Multivariate Adaptive Regression Splines’. Annals of Statistics. 19(1):1–141.
- [22] J. Fan, I. Gijbels. (1996). *Local Polynomial Modelling and Its Applications*, Chapman and Hall, London.