1 Introduction

Since the 80s, SQL database systems gradually became the foundation of Business Intelligence (BI) and On Line Analytical Processing (OLAP) applications. Their success is primarily attributed to *declarative querying* (via SQL), *automatic query optimization* and *physical-logical independence*, which are explained in Section 1.1. These healthy features of SQL led to increased developer and analyst productivity. They also led to specialized roles for the participants, whereas each participant (database developer, database administrator) needs to master only one layer of the overall architecture.

However, SQL systems fail in the space of predictive analytics that involve sensors generating measurements in the space and time dimensions.[1] Section 1.2 argues that at the core of the failure of SQL databases in the management and analytics of sensor spatiotemporal data is the lack of a critical abstraction, which is the *real world model*, which captures a stochastic process that generates the measurements.[2] The signal processing area has deeply built in its foundations the real world model concept, which leads to great results in the cleaning, compression and reconstruction of signals. However, this area's approaches have been directly applicable only to problems with (a) low diversity of data (e.g., when the entire data set of interest comes from a single type of sensor and there are no "metadata" and/or other alphanumeric data to provide context to the sensor data analysis and its results), (b) low volume of data, with no complex needs on how to effectively persist the data set over time and (c) a single type of "hardwired" analysis. It is apparent that databases and signal processing techniques should be coupled but, unfortunately, currently they do not mix well. The result is reduced productivity, very high requirements on the analysts (must be simultaneously experts in signal processing, statistics and big data management) and projects that cannot easily incorporate many types of data and many types of analysis.

Section 1.3 introduces the proposed Plato database system, which brings the real world model concept into SQL databases, therefore combining the healthy productivity aspects of SQL systems (declarative queries, multiple levels of abstraction, automated optimization) with the sensor data processing benefits of signal processing techniques. Central to Plato are the reduced-noise, additive models (Section 1.4) that can be derived from common learning algorithms (in the field of signal processing) confer both a data quality benefit and a huge efficiency benefit, by virtue of providing very compressed representations of the model. Query processing algorithms can run directly on these compressed representations.

Use cases are discussed in Section 1.5. Section 2 summarizes the project's impact and Section 3 (intellectual merit) lists a summary of the problems that must be solved.

1.1 The healthy aspects of SQL databases in analytics

In a typical database-driven application architecture (see Figure 1), the business intelligence application issues an SQL query over tables. As far as the application is concerned, the tables are *logical* in the sense that they are just mathematical relations. In contrast, the underlying data structures where the tables are stored, as well as the indices on the tables, constitute the *physical* layer, whose knowledge is not needed for writing queries. The queries are declarative in the sense that they only describe the desired result without describing the algorithm that computes the result. A query optimizer automatically finds a good plan (in effect, algorithm) for executing the query, taking into consideration the specifics of the physical layer. Declarativeness and automatic optimization increase *developer productivity* and *lower the sophistication bar* needed for writing analytics. Furthermore, they are a key enabler behind visual OLAP systems, where the (non-programming) user can easily aggregate, correlate, drill-down in the data, pivot, etc. During run-time, such systems (e.g., Cognos, Microstrategy) face the relatively easy task of translating clicks into declarative queries, while the problem of optimizing these queries is left to the database.

A database administrator may "tune" the physical layer from time to time in order to boost the performance of the observed workload. However, logical/physical separation says that such tuning changes at the physical

[1]Indeed, one may argue that some of the discussed failures and respective remedies proposed by Plato, also apply to predictive analytics of non-sensor, non-spatiotemporal data. Nevertheless, the focus of this project is exclusively on spatiotemporal sensor data.

[2]The real world model is frequently called the "physical model" as it corresponds to the physics of the real world. This proposal uses the term "real world model" (instead of the term "physical model") to avoid the confusion caused by the two different meanings of the word "physical" in the sensors' signal processing and the database community.

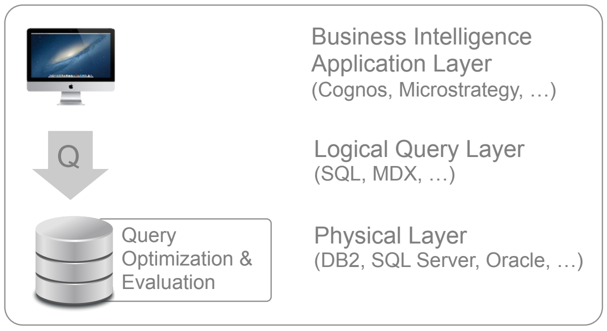

Figure 1: Database-driven analytics architecture

layer do not require rewriting of the logical layer. Rather, the query processing speed automatically benefits. Notice how logical/physical separation divides the expertise and tasks needed for the maintenance of the overall system, therefore making its maintenance and evolution very easy and inexpensive.

Further levels of abstraction are achieved with *views*, which are tables that capture the result of filtering, combining and aggregating the data of the logical tables in a way that simplifies the queries issued by the applications. A database administrator may elect to materialize some views, in order to precompute some of the computations that the live queries must perform. Again, notice that when the database administrator changes a view from virtual to materialized, the developer need not change her queries.

1.2 The failure of SQL databases in spatiotemporal sensor data analytics

In conventional SQL databases, the data contained in the database have a one-to-one mapping to real world objects and events. For example, a database may represent persons, which have an address, a name and a telephone number. Each represented person is supposed to be a corresponding unique entity in the world and vice versa, as far as the database and the applications running on it are concerned, the represented entities, attributes and relationships constitute the *ground truth*. A single mistake in this ground truth can cause the whole system to fail. In contrast to a conventional database, the sensor data are mere *measurements* (samples) of the physical quantities. The accuracy of these measurements is limited and they are typically a combination of *signal* corresponding to the physical quantity to be measured and *noise* which is the result of many random influences on the sensor. The measurements are therefor not the *ground truth* but rather are noisy reflections of ground truth. Reality wares a veil. In order to lift that veil, scientists and signal processing experts use *models* of the real world. Such models embed the laws of physics, both deterministic and stochastic, and are used to increase the signal to noise ratio of the raw measurements. It also represent physical reality in continuous time and space, whereas measurements are always taken at discrete times and locations. For example, when a data manager expects a signal to be periodic, she might use the Fourier transform to discover the dominant frequencies. Given the amplitude and phase of these frequencies, the model can predict a value for any time.

A second cornerstone of statistical sensor data processing, also missed by SQL systems, is the assumption that the real world process that leads to the measurements is a stochastic process. Consequentially an issued query will typically not return a single value; rather, it would return a *distribution* over values. This distribution is often simplified by providing a confidence interval for the desired value.

In the absence of real world models in database systems, various sub-optimal approaches are used. An approach is to use a database simply as a “dumb” storage of measurements. Then given an analytics request, one has to first issue queries that retrieve the relevant measurements. Then she deploys signal processing learning algorithms on them to compute the model’s parameters. Finally, the results are moved into the database. In a variation to this approach, one may not use a SQL database at all (since SQL does not buy her significant functionality anyway) and instead rely on file systems (including the highly scalable distributed file systems).

In either case, the healthy aspects of SQL database systems discussed in Section 1.1 are no longer observed. Instead, the above process is characterized by low productivity as data move in an out of the database in hardwired ways. The approach does not scale to cases with high diversity of data (e.g., when there are "metadata" and/or other alphanumeric data that provide context to the sensor data analysis and its results) and the analytics application may issue multiple types of requests. Furthermore, unlike the ecosystem of conventional SQL systems, the analysts of sensor data must be experts of many areas: signal processing, statistics, databases and often big data management and processing. As sensor data proliferate, the short supply of such multiarea experts and the low productivity in developing and maintaining sensor analytics will lead to a world where a lot of sensor data will be collected but only limited analysis will be performed at them, at the cost of missing the insights that these data can offer.

In recent years databases have been extended (typically via UDFs) with machine learning procedures that can allow one to execute such procedures without moving data in-and-out of the database. While such extensions are useful, we argue next that they miss the optimization opportunities that become available once models appear in the database as first class citizens and the query language and query processing are adjusted accordingly. Furthermore, they do not account yet for the probabilistic nature of the data and the statistical nature of the queries.

Recent research systems have expanded SQL databases to probabilistic facts (as opposed to the conventional merely Boolean facts). As we discuss next, the proposed system assumes that the models are also associated with probabilities.

1.3 Plato: Models as a first class citizen of the database and query language

The proposed Plato database has models as a "first class citizen" abstract type of the database. Ignoring the stochastic aspect of the data, a Plato model is a continuous function over time and/or space that predicts a quantity of interest (eg, temperature, velocity, acceleration, air pollutants density etc) for any time/space vector in a prescribed domain. Alternately, a database-minded person may perceive a Plato model as a table whose attributes are time and/or space coordinates and one or more types of quantities. The table has infinitely many tuples and the time/space coordinates form a key, i.e., given values for the time/space coordinates, each quantity has a unique associated value.

In order to capture the stochastic aspect of a data collection process, Plato defines a *distribution* over functions, representing the uncertainty it has about the true state of the outside world.

Consequently, Plato expands the applicability of the successful architecture of Figure 1 into statistical, spatiotemporal sensor data management and analytics. The application issues declarative "Plato SQL" queries, which utilize the models without being concerned about the specifics of how the models are actually represented in the storage nor the intricacies of how to most efficiently execute the queries. (Representation in storage and efficient query execution is introduced next, in Section 1.4). Section 6.1 provides multiple examples of Plato SQL queries. In one example, a query predicts quantities at specific coordinates; in the stochastic counterpart of the example, the query also asks for the confidence interval, given pi value 0.05. In another example, models are correlated. In a third example, queries drill into specific time segments of the models; e,g., the afternoon times. In yet another example, models capturing the daily paths of individual persons are composed with models that provide the distribution of atmospheric pollutants in San Diego County and the result is models of the pollutant levels that these persons breathed; the query that composes the models is essentially just a simple SQL join. These model-related queries are seamlessly combined with the rest of the database, therefore making it easy to draw results about various segments of the entities represented in the database. For example, a single query can test an hypothesis regarding the inhalation of an atmospheric pollutant by individuals and the asthma attack incidents (as recorded in the conventional medical record) of those individuals.

1.4 Reduced-noise, additive model representations towards data quality and processing speed

While models corresponds to functions over continuous domains, the *storage-level representation* (from now on called simply *representation*) of the models cannot be infinite. A powerful way to overcome this problem comes from functional analysis. Restricted sets of functions can be represented as a weighted sum over an orthonormal basis. popular basis functions are the Fourier basis, various wavelet bases, and the Singular value decomposition (SVD).

Fortunately, vast prior work in signal processing research teaches a fruitful connection between good models and high compression. Good models can be approximated with high accuracy with much fewer bits than the original measurements data. Intuitively, the effectiveness of such representations is connected to the physics of the measured physical reality. For example, the frequency domain representations generated by Fourier and wavelet transforms capture the periodicity of certain signals. In another example, ARMA models capture the differential equations that govern the connection between neighbor points in time and space.

Plato will be based on a particular variation of models and model representations. The model representation will be *additive*, in the sense that the first bits of the representation will provide the most dominant components. Subsequent bits will provide decreasingly important model components. This approach raises the opportunity for a new type of top-k algorithms, which read just enough model components to achieve the level of confidence required by the query.

Eventually, the model representation stops storing additional components when the residual information is white noise. (The project can also expand to other types of noise, e.g., pink noise.) The model components that have been collected up to that point are essentially a more faithful representation of the true underlying real world reality than the original measurements. Practically, this means a data quality advantage: The values predicted by the model are free of the unimportant noise that may have been present in the measurements. These *reduced-noise* models are the data quality benefit of a Plato model (and the also the reason for naming the project Plato, in an allusion to Plato's Cave allegory[3]

1.5 Use cases

This project (and many of this proposal's examples) have been inspired by interactions of the PIs with the sensor data sets of two prior UCSD projects: the Energy Dashboard project (`http://energy.ucsd.edu`) and the DELPHI project (`http://delphi.ucsd.edu`).

The UCSD San Diego Energy Dashboard project (PI'd by Professor Rajesh Gupta) has planted 80,000 "smart building" sensors in UCSD and collects a vast volume of data. The PIs of the Plato proposal obtained the data during the recent months[4] and ran preliminary experiments that validate the appropriateness of these data for the purposes of Plato. In particular, the PIs' team ran SVD and wavelet transformations on the acquired data and validated that the resulting highly compressed, reduced-noise model representations were sufficient for efficiently answering queries on expected values and correlations. While the limited extent of the experimentation does not yet constitute a proof-of-concept of the validity of Plato's approach, this data set is clearly appropriate for Plato's experimentation. While the sensor analytics cooperation between SDSC and co-PI Freund will currently proceed with traditional statistical signal processing techniques, we will later perform the same analytics with Plato, in the interest of assessing the productivity gain.

The NSF-funded DELPHI project (Data E-platform Leveraged for Patient Empowerment and Population Health Improvement, PI'd by Professor Kevin Patrick) collects environmental sensor data. Some sensors belong to the San Diego County and measure the concentration of multiple air polutants. An atmosperic polutants model has been obtained from the San Diego Supercomputing Center: Given the measurements, it predicts the concentration of a pollutant at given space coordinates and time. Other sensors belong to the individuals and track [5] the places where an individual walks or runs every day. The project will soon also obtain asthma attack-related data from patients who have the respective inhaling activity sensor. The "hardwired" approach by which these data are going to be correlated with each other (i.e., how do asthma attacks correlate with the air an individual breathes), analyzed and visualized reduces the productivity of population level studies. Since Plato cannot be ready by the time that these population studies/analytics will need to be contacted, the first round of DELPHI analytics will be built using the current state of the art in

[3]In Plato's Cave, a few prisoners are bound, since they were born, to look at a wall where shadows of the real world's objects appear. The prisoners perceive only the 2-dimensional world of the shadows and so they miss the deeper insights that a comprehension of the full 3-dimensional reality would offer them. Similarly, the sensor measurements that appear in a database are the mere projection of the real world. The insight in the world is captured by physical models (i.e., models that capture the world's physics).

[4]co-PI Freund collaborates with researchers from the San Diego Super Computer Center and with Research firm OSI-Soft Inc. on a large scale system for the analysis of the 500 TB of sensor measurements collected from buildings in UCSD over the last two years.

[5]The PI of the Plato project is one of the five co-PIs of the DELPHI project

sensor analytics. The Plato project's goal is to later replicate the same analytics using Plato and compare both results' quality and analyst/developer productivity.

2 Broad Impact

Sensor data of diverse types and large volumes will soon need to be combined with databases and lead to a new generation of analytics. We argue that this new generation of analytics must be based on the same healthy database technology cornerstones that the prior business intelligence and On Line Analytical Processing software (OLAP) was based: Declarative queries, automatic optimization, efficient storage representations and multiple layers of abstraction can lead to high productivity, which is currently absent from sensor data analytics. At the same time, this new generation must build upon the wealth of knowledge in statistical signal processing.

Upon completion, Plato will deliver the envisioned productivity gains and will lower the technical sophistication bar needed for acting in the space, therefore opening the gates to many scientists. We already see that Plato can have a large impact on the projects that it was inspired from (smart buildings and population health studies) and we expect equally large impact in more areas.

Education This proposal is part of a broad UCSD initiative in Big Data analytics. One of the components of this effort is a new graduate degree titled "master of advanced studies in data science and engineering" or MAS-DSE for short. The graduate program is targeted at continuing education for students that are currently working in industry and whose goal is to become data scientists. The target audience, defined after extensive market research, will consist of people with batchelor degrees in computer science or statistics or people with a degree in the physical sciences that have extensive experience in scientific data analysis. The MAS-DSE course plan combines courses in Statistics, Data management and visualization and the scientific method culminating with a capstone project in which teams will analyze a large data set.

Co-PI Freund directs the program and both he and PI Papakonstantinou teach courses in the MAS-DSE program. Plato will be an invaluable framework for MAS-DSE as it brings together declarative data queries and statistical modeling.

3 Intellectual Merit

The delivery of Plato requires innovative solutions to the following problems:

1. Design and implementation of a model-aware data model (where models are continuous functions) and respective query language features that allow seamless combination of conventional SQL querying with statistical signal processing. Notice that the design must be aware of the stochastic nature of the models. In particular, the query language must allow expression of the required accuracy/confidence of the query results (see Sections 5 and 6).
2. Design and implementation of learning algorithms that learn the model components of reduced-noise, additive model representations. In particular, we will adapt the wavelet, SVD and ARMA transforms to their reduced-noise and additive versions. Consequently, the automated query processing algorithms should try to (a) operate directly on the compressed model representations[6] - as opposed to decoding the model into multidimensional arrays of samples and (b) utilize the fewest number of bits from the additive model representation. Indeed, they should use the minimum number of bits that ensures the query required confidence. In this way, the queries can be most efficient. This goal will lead to a new generation of top-k algorithms, where the goal is to minimize the amount read by the additive model representation (see Sections 7.1-7.4 and 8).
3. Design and implementation of semiautomated algorithms that further compress the model representations by considering the dependencies (mutual entropy) between the models. The process should be iterative, in the sense that the model administrator provides ideas on dependencies between various models and Plato figures out the effectiveness of these ideas on compression and noise reduction (see Section 7.5).
4. Design and implementation of algorithms for the efficient incremental maintenance of the additive representations (i.e., algorithms that update them as additional data emerge). These algorithms will have to

[6] For example, consider two models represented by their Fourier transform and a query that asks for their correlation. It is most efficient to compute directly on the frequency domain rather than bringing back to time domain.

trade off between the speed of convergence to an accurate revised model and the amount of computation needed in order to do so (see Section 7.6).

5. Provide a theory and tools that examine the question of what is the right schema and models for a models-aware database. In a sense, this question expands classical database design into the models space (see Section 9).
6. Exercise and experiment with Plato on large scale statistical sensor data processing cases, such as the ones presented by the Energy Dashboard and the DELPHI project. The experimentation will measure the time to develop analyses as well as the runtime efficiency of the analyses (see Section 1.5).

4 Related Work

Priori work from signal processing and machine learning Work on adaptive signal processing starts in the 50's with the the Wiener filter, followed by the Kalman filter and the Widrow-Hoff algorithm [15]. Enormous progress has been made since that time and adaptive signal processing is now textbook material [13, 9]. Closely related is data compression theory and practice [12]. Recent work of the database field in this area is summarized in [3].

The last decade has seen the rapid development in the area of online learning. This is a subarea of machine learning that is based on ideas from adaptive signal processing and has made interesting and fruitful connection to information theory and game theory. PI-Freund is active in this area of research [2].

Model-based data infrastructure The idea of incorporating models in database systems was first presented in the context of MauveDB [6, 4, 5]. This proposal extends these ideas in several important directions: First, MauveDB argues that models need to be discretized in the coordinates' grid, before they can be queried. In this proposal we show that a fully virtual approach, where the model is perceived as a continuous function, is both easier for the data analyst and more opportune for the query optimizer. For instance, consider two models represented by their Fourier transform and a query that asks for their correlation. It is most efficient to compute this query directly on the frequency domain rather than bring it back to the time domain. Second, while MauveDB showcases some of the challenges that arise in model-based systems and presents solutions for specific models, it lacks a general API that would allow one to plug in arbitrary models (which is one of the main goals of Plato).

Apart from MauveDB, several works studied particular point problems related to the use of models to represent sensor data, such as comparing existing models in terms of compression or designing indices that would be useful for such models [8, 11]. However, neither of these works described a general extensible database platform that can accomodate and offer extensive query capabilities over a variety of models.

Efficient query processing on models The idea of evaluating queries directly on the representation of a model without discretizing them first, was presented in the context of FunctionDB [14]. The work showed that for a broad class of polynomial functions, faster processing is achieved by evaluating queries directly on the algebraic representation of the functions. Plato's goal is to provide the platform infrastructure that enables such optimizations for additional classes of statistical models.

5 Models

The proposed Plato database aims to solve the discussed problems by introducing models as first class citizens. A model is a mathematical representation of the world that predicts a quantity of interest (e.g., temperature) at every point of the spatiotemporal domain. In the basic case, a *model* is a continuous function $f : \mathcal{D} \mapsto \mathcal{Q}$ from a (generally multidimensional) spatiotemporal domain $\mathcal{D}$ to $\mathcal{Q}$. In this work we consider domains $\mathcal{D}$ that are subspaces of the four-dimensional spatiotemporal domain (that includes the three space dimensions X, Y, Z and the temporal dimension T). For instance, the domain of a model can be a sphere of (x, y, t) points around a center (x_0, y_0, t_0), a square of (x, y) points, etc.

EXAMPLE 5.1 For instance, a model for temperature may be a function $f : D_{t\_2012\_now} \mapsto Q_{float}$, which given a point in time t between 2012 and now returns a float, corresponding to the *predicted* temperature at time t. $\square$

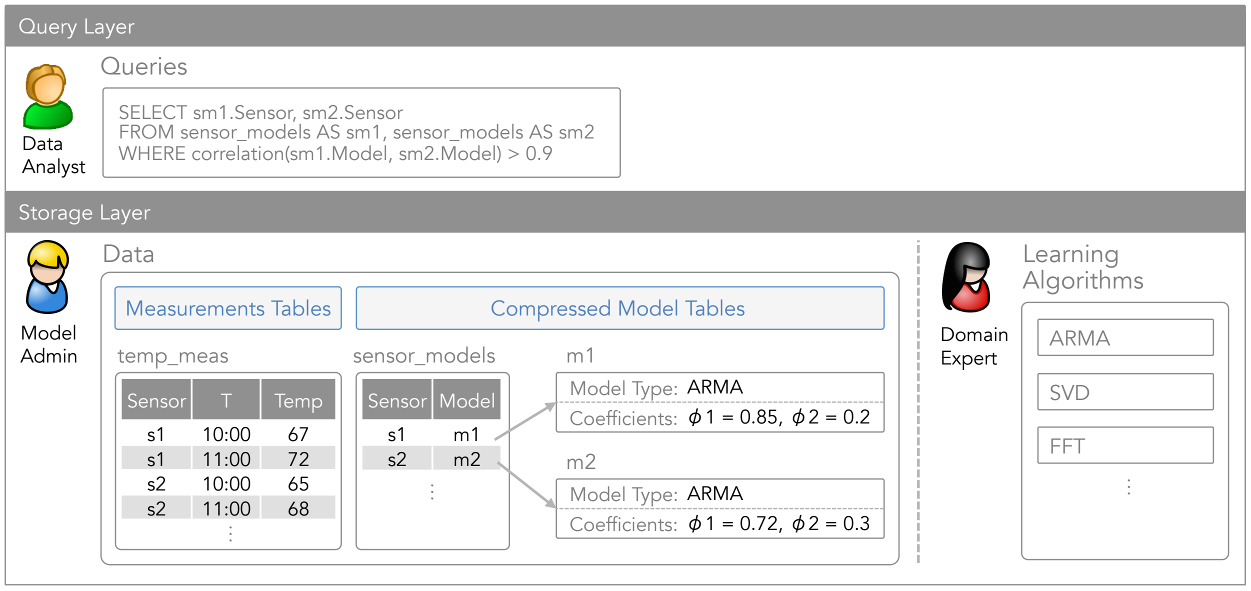

Figure 2: Plato Architecture

6 Plato: An extensible model-aware database

Platon facilitates the integration of models in a general database system through the architecture shown in Figure 2 and discussed next.
Raw sensor data measurements are stored in conventional SQL tables, which we refer to as *measurements tables*. Every measurements table contains among others a subset of the spatiotemporal attributes X, Y, Z and T (representing the three spatial dimensions and the temporal dimension, respectively).

EXAMPLE 6.1 Consider a measurements table with schema `temp_meas(Sensor, T, Temp)` containing tuples of the form (s, t, m), denoting that temperature sensor s provided at time t the temperature measurement m. Attribute `Sensor` is a foreign key to another table `temp_sensor(ID, X, Y, Installed_on, ...)` providing the properties of each sensor, including among others its latitude and longitude. [7] $\square$

To facilitate easy data analysis, the model administrator can build models on top of the raw measurements. To generate a model, the model administrator can utilize *learning algorithms*, which take as input the measurements and potentially background knowledge about the real world and return a model. Learning algorithms are designed by domain specialists, who have knowledge of the domain and the underlying physical properties of the sensor data. For instance, weather experts can design a weather learning algorithm. Although some of the learning algorithms will be domain specific (such as the weather example above), others are general algorithms proposed in the machine learning literature and shown to be applicable to a wide range of domains. This includes among others algorithms for learning an ARMA (Autoregressive-moving-average) model (which is suitable for readings from sensors measuring natural phenomena, such as temperature), an SVD (Singular Value Decomposition) model etc. To bootstrap the system and facilitate the quick generation of models for many common use cases, Plato comes preloaded with several such learning algorithms.
A learning algorithm g has the general form of $g(R;p)$, taking as input a measurements table R and a tuple $\bar{p}$ of parameter values. Since a learning algorithm should be adaptable to different scenarios (e.g., to domains

[7] The data administrator can decide what is the best schema for the measurements data. For example, instead of a single table carrying all the temperature sensor measurements, the administrator may choose one table for each sensor.

of different dimensionality), it is in general polymorphic by accepting a parameter vector p of variable length.

EXAMPLE 6.2 For instance, the learning algorithm $arma(R; \bar{A}_{cont}; \bar{A}_{meas})$ takes as input a measurements table R together with the sets $\bar{A}_{cont}$, $\bar{A}_{meas}$ of attributes of R that correspond to the spatiotemporal attributes and measurement attributes of R, respectively and creates an ARMA model $f : \mathcal{D} \mapsto \mathcal{Q}$, where $\mathcal{D}$ is the subspace of the spatiotemporal domain defined by attributes $\bar{A}_{cont}$ and $\mathcal{Q}$ is the equal to $Q_1 \times Q_2 \ldots \times Q_n$, where Q_i the domain of the i-th attribute in $\bar{A}_{meas}$. $\Box$

Models created through learning algorithms can be used as values in tables. To this end, Plato extends SQL's data types (e.g., string, integer, etc.), with a new *model data type* that comes with an associated model signature $\mathcal{D} \mapsto \mathcal{Q}$. An attribute of such a type is called a *model attribute* and accepts as values models conforming to the corresponding function signature. We will refer to a table that contains at least one model attribute as a *model table*. Model tables are defined by SQL view definitions that involve *learning algorithm* invocations.

EXAMPLE 6.3 For instance, in order to create a model table, containing sensor IDs and an ARMA model, representing the predicted temperatures for the particular sensor, one can write the following statement:

```
CREATE MATERIALIZED VIEW sensor_models AS
SELECT Sensor, arma(G; T; Temp) AS Model
FROM temp_meas GROUP BY Sensor AS G(T, Temp)
```

Note that for ease of exposition, we use an extension of SQL that allows the generation of nested relational tables.[8] In particular the GROUP BY operator creates for each sensor appearing the the measurements table `temp_meas` a nested table G containing all measurements for the particular sensor. These measurements are given as input to the ARMA learning algorithm to create the corresponding ARMA model. The resulting model table `sensor_models(Sensor, Model)` contains tuples of the form (s, f), where s is a sensor and f the corresponding temperature ARMA model. $\Box$

A model table may be virtual in the sense that SQL view definitions are virtual, i.e., no actual computation is performed until a query uses them. However, in practice, the administrator is motivated to materialize models (and the respective model tables) in order to benefit from the data compression that models enable. This can be achieved by adding the MATERIALIZED keyword in the view definition as shown above.
Note, that the administrator has in general to choose in general between multiple model algorithms, which come with different compression levels and corresponding guarantees regarding information preservation. Indeed, a major research issue, discussed in Section 7, is the semiautomation of the choice of model algorithms, which should take into consideration the query/analytics workload.

EXAMPLE 6.4 In a modification of the running example, the administrator can choose to capture temperature in a single model `full_model` that predicts the temperature based on `x`, `y` and `t` and is defined as follows:

```
CREATE MATERIALIZED VIEW full_model AS
arma(SELECT X, Y, T, Temp
FROM temp_meas, temp_sensor WHERE Sensor = ID; X, Y, T; Temp)
```

The reasons for doing so could be: (a) The individual sensor models may be heavily correlated based on the `X` and `Y` sensor coordinates. In such case, a single model can have a much more compact representation than the collection of individual sensor models. (b) The analytics may need temperature predictions for specific locations, which do not coincide with any individual sensor. $\Box$

[8] Extending SQL to nested tables has been a well-studied topic in the database management field and recently it is featured in multiple "Big Data" databases that feature semistructured models.

Probabilistic models For ease of exposition, models have been defined above as returning absolute values. However, since they are inherently approximations of the underlying signal, models return in reality probability distributions of the quantities, rather than absolute values. Therefore a model is a function $f : \mathcal{D} \mapsto \mathcal{H}_{\mathcal{Q}}$, where $\mathcal{H}_{\mathcal{Q}}$ is the space of probability distributions over the quantity domain $\mathcal{Q}$. The probability distributions capture the certainty of the predicted values. For example, the model of Example 6.3 can easily benefit from returning probability distributions of the predicted temperatures. A common way of displaying the probability distribution of a quantity to the end user is through a triple (p, v, ϵ), denoting that the the quantity's value falls within the interval $[v - \epsilon, v + \epsilon]$ with probability p.

6.1 Queries and Optimized Query Evaluation

A key success factor of database systems has been the declarative SQL query language where the user/developer expresses the desired result of his analysis without having to specify the algorithm that computes this result and without having to refer to the data structures where the data are stored. The database discovers the optimal plan to compute the result, making best use of the available data structures.
In the spirit of declarative programming, Plato offers an extension of SQL that enables declarative querying of model tables. A data analyst can query model tables in two different ways.

Querying models through functions According to the first approach, a SQL query over the model tables can operate on model attributes through special functions that take as input one or more models and return a scalar, a tuple, or a new model. For instance, a correlation function takes as input two models $M1$ and $M2$ and returns a scalar corresponding to their correlation. Similarly, a multiplication function takes as input a model M and a scalar c and returns a new model whose values are the values M multiplied by c. Plato supports many common such functions, while additional functions can be added through a suitable API.

EXAMPLE 6.5 The function $correlation(M1, M2)$ takes as input two models $M1$ and $M2$ and computes their correlation. Continuing our running example, one can utilize this function to compute all pairs of temperature sensors whose models have a strong (i.e., greater than 0.9) correlation as follows:

```
SELECT sm1.Sensor, sm2.Sensor
FROM sensor_models AS sm1, sensor_models AS sm2
WHERE correlation(sm1.Model, sm2.Model) > 0.9
```

□

The data analyst may compose these functions to create more complex queries, as long as the top-level function that appears in the query's SELECT or WHERE clause returns a scalar or a tuple (since SQL cannot interpret model attributes). However, function composition can prevent the system from applying certain optimizations, since the query imposes a certain order of execution. For instance, specifying a three-way join through function composition imposes a certain (non-optimal) join ordering.

Querying models by considering them as infinite tables To address this issue, Plato allows data analysts to also query model tables by considering models as infinite tables on which they can execute regular SQL queries. In particular, a model $f : \mathcal{D} \mapsto \mathcal{Q}$ from vector (x, y, z, t) to vector $(q_1, q_2, \ldots q_n)$ can be seen as a table of schema $f(X, Y, Z, T, Q_1, Q_2, \ldots, Q_n)$, where (X, Y, Z, T) is a primary key. Conceptually this table is an infinite table containing a tuple predicting the value of the quantities $Q_1, Q_2, \ldots, Q_n$ for each value in $\mathcal{D}$.

EXAMPLE 6.6 For instance, each ARMA model in our running example can be seen as a table over schema (T, Temp). Thus the model table `sensor_models` defined above is a table over schema $sensor\_models(sensor, (T, Te$
where $(T, Temp)$ is the schema of the nested table corresponding to the model. Using this representation, we can ask for the temperature of all sensors at midnight of 01/01/2014 through the following query:

```
SELECT sm1.Sensor, m1.Temp
FROM sensor_models AS sm1, sm1.Model AS m1
WHERE T = 2014/01/01#00:00:00
```
□

Allowing data analysts to write SQL queries over infinite tables enables Plato to perform query optimizations, such as join reordering. However, the fact that models are conceptually infinite tables also creates two major challenges:

(a) Defining a class of safe queries. Since infinite tables cannot be represented in practice, Plato should only accept *safe queries*; i.e., queries that are guaranteed to return a finite result. Examples of such queries are queries asking about a particular point in the spatiotemporal domain, or a finite set of points. The goal of this work is to find the most general class of queries that are guaranteed to be safe.

(b) Defining the query answering semantics for aggregate queries. The semantics of safe Select-Project-Join queries is well-defined, as the query is operating on the finite domain, specified by the condition that makes the query safe. For instance, in Example 6.6 the query is not operating on the infinite spatiotemporal domain, but only on a finite domain, consisting of the single time point 2014/01/01#00:00:00 specified in the WHERE clause. However, this does not hold for aggregate queries. Even though an aggregate query may be safe (i.e., its final result will be finite), the aggregate function may still operate on the infinite spatiotemporal domain and thus cannot be defined in the same way as in SQL. For instance, consider a model providing precipitation values and the aggregate query computing the sum of precipitation values over a week. While the SUM function is defined in SQL as the sum of a finite number of values, in the case of the model it has to be defined as the integral of the precipitation model. The goal of this work is to formally define the semantics of safe aggregate queries through the use of calculus.

Probabilistic queries Since models return probability distributions, any query can also specify probability guarantees by providing the desired probability p of the result being correct. In that case, Plato returns not only a single value v but a confidence interval $[v-\epsilon, v+\epsilon]$ that is guaranteed to contain the correct answer with probability p. Such a query can be specified through the WITH PROBABILITY clause, as illustrated through the following example.

EXAMPLE 6.7 Query of Example 6.6 can be augmented with a guideline for returning results with 0.95 probability as follows:

```
SELECT sm1.Sensor, m1.Temp
FROM sensor_models AS sm1, sm1.Model AS m1
WHERE T = 2014/01/01#00:00:00
WITH PROBABILITY 0.95
```
□

7 Inferring Models and Representing them Compactly

The practice of data compression [12] shows that most real-world sensor generated sequences can be captured reasonably well using one of a small set of statistical models such as ARMA models, Fourier and wavelet-based models and Singular Value Decomposition models (SVD or PCA). Data compression and statistical models are closely related concepts. Good statistical models allow Plato to achieve high compression rates. Vice versa, high compression rates indicate that Plato has correctly identified significant statistical patterns in the data and has got rid of the noise. The following discussion explains how the model administrator chooses the appropriate models and makes the case for providing additive, reduced-noise models.

The key responsibility of the model administrator (see Figure **??**) is to choose a set of learning algorithms to create corresponding models over the measurements tables. Each learning algorithm creates a model that fits the data and (in case the administrator has selected to materialize it) stores it using a learning algorithm-specific storage representation. For instance, as shown in Figure **??**, a possible storage representation of a model created by the ARMA learning algorithm is the set of the ARMA coefficients. Once the chosen models have been created, the administrator can compare them in terms of compression and distortion and choose the one that performs best.

There are multiple models and respective compressions that are applicable, as explained next. In the interest of comparisons to prior work, we initiate the discussion with a presentation of lossless compression and then continue with models involving lossy compression. In practice, Plato will use additive and reduced-noise models only. The underlying logic is that we aim for minimum query processing time and high quality models, while storage is a secondary only consideration.

7.1 Lossless compression

A lossless compression method is one which can reconstruct the original measurements without error. Popular lossless compression methodss are `gzip` and `compress`. A lossless compression method corresponds to a statistical model over the data and the expected compression ratio is determined by the distance between the statistical model and the empirical distribution of the data (as measured by the the Kullback-Leibler divergence).

Unfortunately, lossless compression methods usually achieve a compression ratio of around 2 to 4. This corresponds to the fact that lossless compression accurately captures both the true state of the world (the signal) and the many random influences on the measurements (the noise). Lossless compression does not distinguish between the two.

7.2 Lossy compression

In order to reach higher compression ratios than lossless compression, it is common to use *lossy compression*.

Lossy compression is based on the assumption that the data to be compressed is a point in Euclidean space. From this point onward we assume that the measurements arriving from a (single) sensor are represented as a sequence of real values $\mathbf{x} = (x_1, x_2, \ldots, x_t)$. We denote the reconstruction of the sequence from the compressed version by $\hat{\mathbf{x}} = (\hat{x}_1, \hat{x}_2, \ldots, \hat{x}_t)$. As we are using lossy compression $\hat{\mathbf{x}}$ is not equal to $\mathbf{x}$. The difference between the two sequences is called the *residual* which we denote by $\mathbf{r} = \mathbf{x} - \hat{\mathbf{x}}$. We require that the residual is small. The measure of the size of the residual is called the *distortion*. We use two of the most popular distortion measures.

- By far, the most common measure of distortion is the L_2 error also called *root-mean-square* error or RMS:

$$\text{RMS}\left(\hat{\mathbf{x}}, \mathbf{x}\right) \doteq \sqrt{\frac{\sum_{i=1}^{t} r_i^2}{t}}$$

- If the errors are usually small and only rarely very large, the RMS is misleadingly large. A better error measure in such situations is the fraction of times the error is larger than some threshold:

$$\text{err}_\epsilon\left(\hat{\mathbf{x}}, \mathbf{x}\right) \doteq \frac{\sum_{i=1}^{t} \mathbf{1}\left(|r_i| > \epsilon\right)}{t}$$

Lossy compression methods such as `jpeg2000` for images and `mp3` for audio can achieve compression ratios of 100 or more with no perceptible degradation in quality.

7.3 Additive models

Some of the most commonly used models for lossy compression are additive with respect to the RMS distortion measure. This set of models includes FFT, Wavelets, and SVD. What we mean here by additivity is that any model is a sum of components: $\hat{\mathbf{x}} = \sum_{i=1}^{k} \mathbf{c}_i$; that the best model for $k_2 > k_1$ shares the first k_1 components with the best model using k_1 components; and that the reduction in the distortion is largest for the first component and decreases monotonically as k increases.

Additive models lend themselves to an incremental compression scheme. Starting by setting the residual to the original measurements $\mathbf{r}_0 \leftarrow \mathbf{x}$, repeat the following steps for $i = 1, 2, \ldots$

• Find the vector $\mathbf{c}_i$ that minimizes the RMS error: RMS $(\mathbf{c}, \mathbf{r})$
• Subtract the identified component from the residual: $\mathbf{r}_i = \mathbf{r}_{i-1} - \mathbf{c}_i$
This iterative process is repeated until $\mathbf{r} = 0$. At that point we have a lossless compression of the original sequence. However, this sequence is organized in such a way that one can build the reconstruction of the original sequence incrementally, quickly getting a reasonable approximation and then improving that approximation to the desired level.

7.4 Reduced-noise models

Lossy models invariably loose some of the original data. However, even with a lossy model it is often the case that the result is a reconstruction that is *closer* to the true underlying real-world values than the original measurements. Noise reduction gives rise to a different way of measuring the quality of a compression scheme. Instead of considering the *amplitude* of the residual using RMS or err$_\epsilon$ we can consider whether or not the residual is white noise. If the models that we consider are linear we can measure how similar the residual is to white noise by considering the auto-correlation function for each residual and the cross-correlation between each pair of residuals. We expect the auto-correlation to consist of a single delta-function at zero and we expect the cross-correlation functions to be close to zero everywhere.
The following section opens a second path towards noise reduction

7.5 Better models and higher compression by dependencies

When the data to be compressed arrives from a large number of proximate sensors, there are potentially large benefits from compressing them together using a predictive statistical model such as ARMA. Unfortunately, the space of models that capture all possible dependencies between the sensor readings is extremely large and hard to learn fully automatically. In this case, Plato will engage the model administrator in a semiautomatic procedure.
The role of the model administrator in this context is to restrict the set of possible interactions to make the compression most effective, while keeping the associated the learning problem reasonable easy. In the example of modeling the temperatures of rooms in a building the model might be restricted to considering the dependence between the outside temperature and that of the rooms and in addition, the dependence between neighboring rooms. A model that identifies the stronger dependencies within these restrictions is useful both for data compression, as a model of the physical reality (which rooms are better insulated than others) and as a basis for noise reduction.
Notice that noise reduction is possible when a good model of the dependencies between different sensors is available. In such situations the compression and decompression of the data shifts the data from the measured vector towards the manifold that is consistent with the physical and statistical properties of the data. The result is similar to that of *smoothing* the data, with the important difference that the smoothing is based solely on the statistical model that was inferred from the data.

7.6 Model Maintenance

When considering a database system that processes data streams over long periods of time it is necessary to consider how to maintain the models over time. We divide this problem into two parts, depending on whether or not the data source is stationary.
When the data source is stationary, efficient model maintenance is closely related to the subjects of sufficient statistics and or data synopses or sketches [3]. In any of these formulation the goal is to define a small data structure that stores the information necessary to define the model. We plan to test the methods proposed in the literature and, where necessary, develop new ones.
When the data source is not stationary, we need our model to track the distribution of the data. This causes an unavoidable tradeoff between accuracy and adaptivity. In order to adapt quickly the model has to ignore data from the distant past. At the same time, ignoring data from the distant past limit the accuracy at which the model parameters can be estimated. Calibrating the algorithm to consider the optimal window into the past has been extensively studied in online learning theory [2], in particular in [10, 1, 7] there is a detailed study of algorithms for combining models that change over time. Plato will use these online learning methods for efficient mode maintenance when the data distribution varies over time.

8 Query Processing

In Section 6.1 we described the query language that allows data analysts to interact with model tables. In this Section we describe how these queries are processed by Plato's query processor.

EXAMPLE 8.1 Consider a variation of the running example in which the temperature models are created using an FFT (Fast Fourier Transform) learning algorithm and stored as sets of frequency-amplitude pairs. Given these models, consider the following query asking for pairs of highly correlated temperature sensors.

```
SELECT sm1.Sensor, sm2.Sensor
FROM sensor_models AS sm1, sensor_models AS sm2
WHERE corelation(sm1.Model, sm2.Model) > 0.9
```

□

Processing queries by materializing model values on a grid MauveDB suggested that the analyst specifies a spatiotemporal grid and each model participating in a query returns the quantities at the grid's points. Consequently, the statistical functions of the query can be computed in a straightforward way.

EXAMPLE 8.2 The following Plato query utilizes MauveDB's technique. It dictates a grid-based execution of the query of Example 8.1.

```
SELECT sm1.Sensor, sm2.Sensor
FROM sensor_models AS sm1, sensor_models AS sm2
LET grid_start = min_coord(sm1.Model, sm2.Model)
LET grid_end = max_coord(sm1.Model, sm2.Model)
WHERE corelation(grid(sm1.Model, grid_start, grid_end, 60)
                 grid(sm2.Model, grid_start, grid_end, 60)) > 0.9
```

The function $\texttt{grid}(f, l, u, s)$ creates a discrete model f_d that is defined only on the grid defined by the start l, the end u and the step s. For every point $t_i = l + is, t_i \leq u$ it is $f_d(t_i) = f(t_i)$. □

While the grid materialization provides a baseline solution, it has two shortcomings. First, the analyst must guess an appropriate grid granularity. A too fine grid will produce accurate results but it will also lead to unnecessarily large discrete models and slow query processing. On the other extreme, a too coarse grid will produce inaccurate results. A better approach is to allow Plato to automatically devise the appropriate grid granularity, based on smoothness of the involved functions. Still, even with automatic inference of the appropriate grid, the grid-based query processing misses the large opportunity discussed next.

Processing queries directly on model representations Instead of computing the values of a function on a grid and subsequently applying a function, many statistical functions can be performed directly on the storage representation of a model, therefore reaping the benefits of the compression. In the running Example 8.1, the correlation function query can be executed faster directly on the frequency representation. To enable query processing on the storage representation, the domain specialist designing a learning algorithm can provide Plato with the corresponding implementations of the statistical functions. Consequently, query processing will be automatically using the appropriate implementation (e.g., correlation on the frequency domain). The grid technique will be the last resort, in the case where the domain specialist has not provided an implementation that works directly on the compressed model representation.

8.1 Anytime query processing

The domain specialists may elect to structure the storage representation of their models such that it enables the answer of queries with a strict deadline (also known as anytime queries). To enable anytime query processing a model's storage representation should store its components ordered by importance, so that the query result can be still computed (albeit with lower accuracy) by ignoring some of the components.
For example, consider a model produced by the FFT learning algorithm. One possible storage representa-

tion of this model is a sequence of frequency-amplitude pairs, sorted by the amplitude of each frequency. Given this storage representation, the query of Example 8.1 can be executed by applying the frequency-based correlation algorithm first at high amplitude frequencies and progressively as time allows to lower amplitude frequencies. Generally, a model representation can be stored in sequences of data where prefixes of increasing size correspond to models of increasing accuracy. Plato will utilize the above representation property to allow both anytime queries that have to be executed within a certain timeframe, as well as queries that return a continuous result with ever increasing accuracy (similar to the interface adopted by works on online aggregation).

9 Model Database Design Principles and Tools

A data warehouse designer uses well understood techniques to choose which added-value tables to materialize in the warehouse in order to support a .[9] In a sense, Plato is also a data warehouse, where models provide added value on the measurements data, by revealing the real world reality behind the measurements. Similarly, a Plato warehouse designer will need to make a good choice of tables and models, based on appropriate criteria, whose application can be assisted by tools.

The first one, called *precomputation* design criterion, is similar to conventional warehouse design. It calls for computing in advance the models that are used in recurrent queries, instead of computing the models from the measurements online. The latter approach would be too slow.

The second one, which we call *consolidation*, is novel and enables higher compression of the representations. In a sense, it is a probabilistic extension of database normalization, dictating that if some models are heavily correlated, the administrator should consider consolidating them into a single model, even if this single model is not being used directly by a query.

EXAMPLE 9.1 For instance, Example 6.3 models the sensor data by a table `sensor_models(sensor, temp_model)`, i.e., each sensor has a corresponding temporal model. The next example models the sensor data by a single model `full_model` that predicts the temperature based on `x`, `y` and `t`. Next, let us assume that the application often issues queries that ask for the predicted temperatures at locations other than the locations of the sensors. While it is possible for the query to compute the required predictions online from the models of `sensor_models(sensor, temp_model)`, such approach would probably be very slow. The precomputation design criterion says that, given such query workload, the model administrator must choose the single model.

Then let us consider an alternate scenario where the application does *not* issue queries that ask for predicted temperatures at arbitrary locations. Rather, it cares exclusively about the temperatures at the sensor sites. In such case the precomputation criterion points towards the “many models” design of Example 6.3. Yet the consolidation criterion may still dictate that it is best to choose the single model of Example **??**. The intuition is that the temperature sensors may be correlated since physics says that temperatures are correlated in sufficiently short distances. The consolidation criterion can be formally judged by comparing the entropy of the models of the “many models” design against the entropy of the “single model’ design. □

10 Relevant NSF Prior Work of the PIs

PI Papakonstantinou’s expertise is in data-driven middleware and data integration software. A theme of his research works has been the delivery of declarative, query-based platforms for solving problems in data integration and/or application development. The small NSF IIS awards **1018961** and **1219263** (PI’d by Papakonstantinou, publications [1], [3] of Papakonstantinou bio) develop a declarative, SQL-based programming system, named FORWARD, which delivers database-driven Ajax pages as rendered SQL views, obliterating the need for low level, low productivity Ajax visualizations coding. FORWARD has already been deployed in industrial use cases[10] and makes the case for the productivity and simplicity of declarative programming, which is a theme pursued in the current project also, albeit combining different technologies with SQL. PI Papakonstantinou is a co-PI of NSF SHB award **1237174** (project DELPHI), which has provided

[9] Such tables are often called *materialized views*.

[10] A limited software release of the FORWARD framework has been used in local pharmaceutical companies (Ferring, Pfizer) since 2012 and the current version is now in pilot testing at Teradata, for the purpose of providing a high productivity visualization framework.

a key inspiration towards Plato, as explained in Section 1.5 and as also articulated in the publication [4] of Papakonstantinou bio.

Co-PI Freund is an expert in the field of machine learning, statistics and information theory, his best known work is the Adaboost learning algorithm, which he co-invented with Dr Robert Schapire. In recent years his main focus has been on the analysis of large collections of sensory data, image collections and data collected using microphone arrays.

Co-PI Freund is an expert on online learning algorithms, these are algorithms for learning from a stream of data which make no assumption about the way the data is generated.

Co-PI Freund is is collaborating with researchers from the San Diego Super Computer Center and with Research firm OSI-Soft Inc. on a large scale system for the analysis of sensor measurements collected from buildings in UCSD over the last two years. This data consists of 80,000 sensor readings and amounts to about 500 TB of information.

Co-PI's is the PI on NSF grant number 1162581 **RI: Medium: Quantifying and utilizing confidence in machine learning**. The focus of this work is on identifying measures of confidence that hold with the little or no prior assumptions. Such confidence prediction measures are important for active learning and for the work planned to be done on this proposal.