Creating Probabilistic Databases from Imprecise Time-Series Data

Saket Sathe, Hoyoung Jeung, Karl Aberer

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{saket.sathe, hoyoung.jeung, karl.aberer}@epfl.ch

***Abstract*—Although efficient processing of probabilistic databases is a well-established field, a wide range of applications are still unable to benefit from these techniques due to the lack of means for creating probabilistic databases. In fact, it is a challenging problem to associate concrete probability values with given time-series data for forming a probabilistic database, since the probability distributions used for deriving such probability values vary over time.**

In this paper, we propose a novel approach to create tuple-level probabilistic databases from (imprecise) time-series data. To the best of our knowledge, this is the first work that introduces a generic solution for creating probabilistic databases from arbitrary time series, which can work in online as well as offline fashion. Our approach consists of two key components. First, the *dynamic density metrics* that infer time-dependent probability distributions for time series, based on various mathematical models. Our main metric, called the GARCH metric, can robustly capture such evolving probability distributions regardless of the presence of erroneous values in a given time series. Second, the Ω*–View builder* that creates probabilistic databases from the probability distributions inferred by the dynamic density metrics. For efficient processing, we introduce the σ–cache that reuses the information derived from probability values generated at previous times. Extensive experiments over real datasets demonstrate the effectiveness of our approach.

I. Introduction

One of the most effective ways to deal with imprecise and uncertain data is to employ probabilistic approaches. In recent years there have been a plethora of methods for managing and querying uncertain data [1]–[7]. These methods are typically based on the assumption that probabilistic data used for processing queries is available; however, this is not always true. *Creating* probabilistic data is a challenging and still unresolved problem. Prior work on this problem has only limited scope for domain-specific applications, such as handling duplicated tuples [8], [9] and deriving structured data from unstructured data [10]. Evidently, a wide range of applications still lack the benefits of existing query processing techniques that require probabilistic data. Time-series data is one important example where probabilistic data processing is currently not widely applicable due to the lack of probability values. Although, the benefits are evident given that time series, in particular generated from sensors (environmental sensors, RFID, GPS, etc.), are often imprecise and uncertain in nature.

The work presented here was supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

Before diving into the details of our approach let us consider a motivating example (see Fig. 1). Alice is tracked by indoor-positioning sensors and her locations are recorded in a database table called `raw_values` in the form of a three-tuple $\langle time, x, y\rangle$. These raw values are generally imprecise and uncertain due to several noise factors involved in position measurement, such as low-cost sensors, discharged batteries, and network failures. On the other hand, consider a probabilistic query where an application is interested in knowing, given a particular time, the probability that Alice could be found in each of the four rooms. For answering this query we need the table `prob_view` (see Fig. 1). This table gives us the probability of finding Alice in a particular room at a given time. To derive the `prob_view` table from the `raw_values` table, however, the system faces a fundamental problem—how to *meaningfully* associate a probability distribution $p(R)$ with each raw value tuple $\langle time, x, y\rangle$, where R is the random variable associated with Alice's position.

Once the system associates a probability distribution $p(R)$ with each tuple, it can be used to derive probabilistic views, which forms a probabilistic database used for evaluating various types of probabilistic queries [1], [3]. Thus, this example clearly illustrates the importance of having a means for creating probabilistic databases. Nevertheless, there is a lack of effective tools that are capable of creating such probabilistic databases. In an effort to rectify this situation, we focus on the problem of creating a probabilistic database from given (imprecise) time series, thereupon, facilitating direct

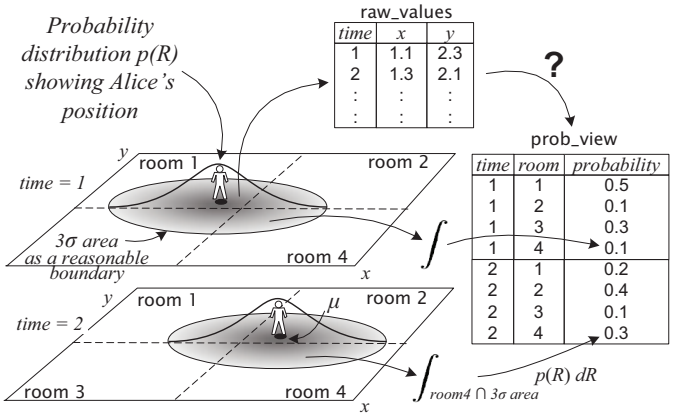


Fig. 1: An example of creating a tuple-level probabilistic database from time-dependent probability distributions.

processing of a variety of probabilistic queries.

Unfortunately, creating probabilistic databases from imprecise time-series data poses several important challenges. In the following paragraphs we elaborate these challenges and discuss the solutions that this paper proposes.

Inferring Evolving Probability Distributions.
One of the most important challenges in creating a probabilistic database from time series is to deal with evolving probability distributions, since time series often exhibit highly irregular dependencies on time [6], [11]. For example, temperature changes dramatically around sunrise and sunset, but changes only slightly during the night. This implies that the probability distributions that are used as the basis for deriving probabilistic databases also change over time, and thus must be computed dynamically.

In order to capture the evolving probability distributions of time series we introduce various *dynamic density metrics*, each of them dynamically infers time-dependent probability distributions from a given time series. The distributions derived by these dynamic density metrics are then used for creating probabilistic databases. After carefully analyzing several dynamical models for representing the dynamic density metrics (details are provided in Section III and Section VII), we identify and adopt a novel class of dynamical models from the time-series literature, which is known as the GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) model [12]. We show that the GARCH model can play an important role in efficiently and accurately creating probabilistic databases, by effectively inferring dynamic probability distributions.

An important challenge in identifying appropriate dynamic density metrics is to find a measure that precisely assess the quality of the probability distributions produced by these metrics. This assessment is important since it quantifies the quality of probabilistic databases derived using these probability distributions. A straightforward method is to compare the ground truth (i.e., true probability distributions) with the inference obtained from our dynamic density metrics, thus producing a tangible measure of quality. This is, however, infeasible since we can neither observe the ground truth nor establish it unequivocally by any other means. To circumvent this crucial limitation, we propose an indirect method for measuring quality, termed *density distance*, which is based on a solid mathematical framework. The density distance is a generic measure of quality, which is independent of the models used for producing probabilistic databases.

Unfortunately, the GARCH model works inappropriately on time series that contain erroneous values, i.e., significant outliers, which are often produced by sensors. This is because the GARCH model is generally used over precise, certain, and clean data (e.g., stock market data). In contrast, the time series that this study considers are typically imprecise and erroneous. Thus, we propose an improved version of the GARCH model, termed C-GARCH, that performs appropriately in the presence of such erroneous values.

Efficiently Creating Probabilistic Databases.
Given probability distributions inferred by a dynamic density metric, the next step of our solution is to generate views that contain probability values (e.g., `prob_view` in Fig. 1). We introduce the Ω-*View builder* that efficiently creates probabilistic views by processing a *probability value generation query*. The output of this query can be directly consumed by a wide variety of existing probabilistic queries, thus enabling higher level probabilistic reasoning.

Since the probabilistic value generation query accepts arbitrary time intervals (past or current) as inputs, this could incur heavy computational overhead on the system when the time interval spans over a large number of raw values. To address this, we present an effective caching mechanism called σ-*cache*. The σ–cache caches and reuses probability values computed at previous times for current time processing. We experimentally demonstrate that the σ–cache boosts the efficiency of query processing by an *order of magnitude*. Additionally, we provide theoretical guarantees that are used for setting the cache parameters. These guarantees enable the choice of the cache parameters under user-defined constraints of storage space and error tolerance. Moreover, such guarantees make the σ–cache an attractive solution for large-scale data processing.

Contributions.
To the best of our knowledge, this is the first work that offers a generic end-to-end solution for creating probabilistic databases from arbitrary imprecise time-series data. Specifically, we first introduce various dynamic density metrics for associating tuples of raw values with probability distributions. Since sensors often deliver error prone data values we propose effective enhancements which make the dynamic density metrics robust against unclean data. We then suggest approaches which allow applications to efficiently create probabilistic databases by using a SQL-like syntax.

To summarize, this paper makes the following contributions:

- We adopt a novel class of models for proposing various dynamic density metrics. We then enhance these metrics by improving their resilience against erroneous inputs.
- We introduce density distance that quantifies the effectiveness of the dynamic density metrics. This serves as an important measure for indicating the quality of probabilistic databases derived using a dynamic density metric.
- We present a generic framework comprising of a malleable query provisioning layer (i.e., Ω–View builder) which allows us to create probabilistic databases with minimal effort.
- We propose space- and time-efficient caching mechanisms (i.e., σ–cache) which produce manyfold improvement in performance. Furthermore, we prove useful guarantees for effectively setting the cache parameters.
- We extensively evaluate our methods by performing experiments on two real datasets.

We begin by giving details of our framework for generating probabilistic databases in Section II. Section III introduces the naive dynamic density metrics while in Section IV we propose

the GARCH metric. An enhancement of the GARCH metric, C-GARCH, is discussed in Section V. In Section VI, we suggest effective methods for generating probabilistic databases, this is followed by a discussion on σ–cache. Lastly, Section VII presents comprehensive experimental evaluations followed by the review of related studies in Section VIII.

II. Foundation

This section describes our framework, defines queries this study considers, and proposes a measure for quantifying the effectiveness of the dynamic density metrics. Table I offers the notations used in this paper.

A. Framework Overview

Fig. 2 illustrates our framework for creating probabilistic databases, consisting of two key components that are dynamic density metrics and the Ω–View builder. A *dynamic density metric* is a system of measure that dynamically infers time-dependent probability distributions of imprecise raw values. It takes as input a sliding window that contains recent previous values in the time series. In the following sections, we introduce various dynamic density metrics.

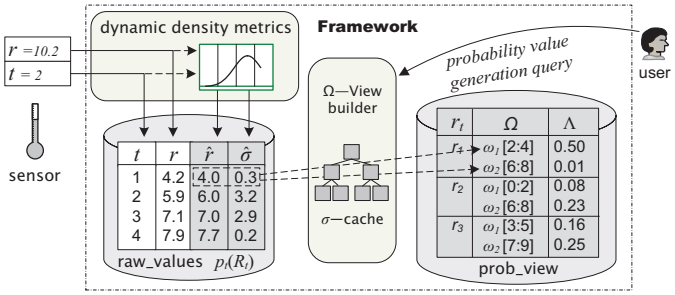


Fig. 2: Architecture of the framework.

Let $S = \langle r_1, r_2, \cdots, r_t \rangle$ be a time series, represented by a sequence of timestamped values, where $r_i \in S$ indicates a (imprecise) raw value at time i. Let $S^H_{t-1} = \langle r_{t-H}, r_{t-H+1}, \cdots, r_{t-1} \rangle$ be a (sliding) window that is a subsequence of S, where its ending value is at the previous time of t. The dynamic density metrics correspond to the following query:

Definition 1: **Inference of dynamic probability distribution.** Given a (sliding) window S^H_{t-1}, the inference of a probability distribution at time t estimates a probability density function $p_t(R_t)$, where R_t is a random variable associated with r_t.

The system stores the inferred probability density functions $p_t(R_t)$ associated with the corresponding raw values. Next, our Ω–View builder uses these inferred probability density functions to create a probabilistic database, as shown in the `prob_view` table of Fig. 2.

Suppose that the data values of a probabilistic database are decomposed into a set of ranges $\Omega = \{\omega_1, \omega_2, \cdots, \omega_n\}$, where $\omega_i = [\omega^l_i, \omega^u_i]$ is bounded by a lower bound ω^l_i and an upper bound ω^u_i. Then, the Ω–View builder corresponds to the following query in order to compute probability values for the given ranges:

Definition 2: **Probability value generation query.** Given a probability density function $p_t(R_t)$ and a set of ranges $\Omega = \{\omega_1, \omega_2, \cdots, \omega_n\}$ for the probability values in a probabilistic database, a probability value generation query returns a set of probabilities $\Lambda_t = \{\rho_{\omega_1}, \rho_{\omega_2}, \cdots, \rho_{\omega_n}\}$ at time t, where ρ_{ω_i} is the probability of occurrence of $\omega_i \in \Omega$ and is equal to $\int_{\omega^l_i}^{\omega^u_i} p_t(R_t) dR_t$.

Recall the example shown in Fig. 1. Let us assume that ω_1 corresponds to the event of Alice being present in *Room 1*. At time $t = 1$, Alice is likely to be in *Room 1* (i.e., ω_1 occurs) with probability $\rho_{\omega_1} = 0.5$.

Note that the creation of probabilistic databases can be performed in either online or offline fashion. In the online mode, the dynamic density metrics infer $p_t(R_t)$ as soon as a new value r_t is streamed to the system. In the offline mode, users may give SQL-like queries to the system (examples are provided in Section VI).

Symbol	Description
S	A time series.
S^H_{t-1}	Sliding window having H values $[t-H, t-1]$.
r_t	Raw (imprecise) value at time t.
R_t	Random variable associated with r_t.
$\hat{r}_t, \mathbb{E}(R_t)$	Expected true value at time t.
$p_t(R_t)$	Probability density function of R_t at time t.
$P_t(R_t)$	Cumulative probability distribution function of R_t at time t.
ρ_ω	Probability of occurrence of event ω.
$\mathbb{E}(X)$	Expected value of random variable X.
$\mathcal{N}(\mu, \sigma^2)$	Normal (Gaussian) probability density function with mean μ and variance σ^2.
Ω	A set of ranges for creating probability values in a probabilistic database.
$\lceil x \rceil$	A smallest integer value that is not smaller than x.

TABLE I: Summary of notations.

B. Evaluation of Dynamic Density Metrics

Quantifying the quality of a dynamic density metric is crucial, since it reflects the quality of a probabilistic database created. Here, we introduce an effective measure, termed *density distance*, that quantifies the quality of a probability density inferred by a dynamic density metric.

Let $p_t(R_t)$ be an inferred probability density at time t. A straightforward manner in which we can evaluate the quality of this inference is to compare $p_t(R_t)$ with its corresponding true density $\hat{p}_t(R_t)$. $\hat{p}_t(R_t)$, however, cannot be given nor observed, rendering this straightforward evaluation infeasible. To overcome this, we propose to use an indirect method for evaluating the quality of a dynamic density metric known as the *probability integral transform* [13]. A probability integral transform of a random variable X, with probability density function $f(X)$, transforms X to a uniformly distributed random variable Y by evaluating $Y = \int_{-\infty}^{x} f(X = u) du$ where $x \in X$. Thus, the probability integral transform of r_i with respect to $p_i(R_i)$ becomes, $z_i = \int_{-\infty}^{r_i} p_i(R_i = u) du$.

Let $p_1(R_1), \ldots, p_t(R_t)$ be a sequence of probability distributions inferred using a dynamic density metric. Also, let $z_1, \ldots, z_t$ be the probability integral transforms of raw values

$r_1, \ldots, r_t$ with respect to $p_1(R_1), \ldots, p_t(R_t)$. Then, $z_1, \ldots, z_t$ are uniformly distributed between $(0, 1)$ if and only if the inferred probability density $p_i(R_i)$ is equal to the true density $\hat{p}_i(R_i)$ for $i = 1, 2, \ldots, t$ [13].

To find out whether $z_1, \ldots, z_t$ follow a uniform distribution we estimate the cumulative distribution function of $z_1, \ldots, z_t$ using a histogram approximation method. Let us denote this cumulative distribution function as $Q_Z(z)$. We define the quality measure of a dynamic density metric as the Euclidean distance between $Q_Z(z)$ and the ideal uniform cumulative distribution function between $(0, 1)$ denoted as $U_Z(z)$. Formally, the quality measure is defined as:

$$d\{U_Z(z), Q_Z(z)\} = \sqrt{\sum_{x=0}^{1} (U_Z(x) - Q_Z(x))^2}. \quad (1)$$

We refer to $d\{U_Z(z), Q_Z(z)\}$ as *density distance*. The density distance quantifies the difference between the observed distribution of $z_1, \ldots, z_t$ and their expected distribution. Thus, it gives a measure of quality for the inferred densities $p_1(R_1), \ldots, p_t(R_t)$. The density distance will be used in Section VII to compare the effectiveness of each dynamic density metrics this paper introduces.

III. Naive Dynamic Density Metrics

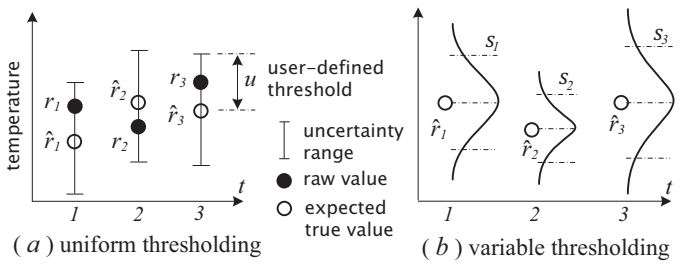
This section presents two relatively simple dynamic density metrics that capture evolving probability densities in time series.

Uniform Thresholding Metric.
Cheng *et al.* [1], [14] have proposed a generic query evaluation framework over imprecise data. The key idea in these studies is to model a raw value as a user-provided uncertainty range in which the corresponding unobservable true value resides. Queries are then evaluated over such uncertainty ranges, instead of the raw values.

Our *uniform thresholding metric* extends this idea for estimating probability distributions by inferring a true value. We define such a true value as:

Definition 3: **Expected true value.** Given a probability density function $p_t(R_t)$, the expected true value $\hat{r}_t$ is the expected value of R_t, denoted as $\mathbb{E}(R_t)$.

Next, the uniform thresholding metric takes a user-defined threshold value u to bound uniform distributions, centered on the inferred true value. Fig. 3(a) illustrates an example of this process where a user-defined threshold value u is used



(*a*) uniform thresholding (*b*) variable thresholding

Fig. 3: Examples of naive dynamic density metrics.

for specifying the uncertainty ranges. The difference between a true value $\hat{r}_t$ and its corresponding raw value r_t is then assumed to be not greater than u.

To infer expected true values, we adopt the *AutoRegressive Moving Average* (ARMA) model [12] that is commonly used for predicting expected values in time series [15]. Specifically, given a time series $S = \langle r_1, r_2, \cdots, r_t \rangle$ and a sliding window S_{t-1}^H, the ARMA model models $r_i = \hat{r}_i + a_i$, where $t - H \leq i \leq t - 1$ and a_i obeys a zero mean normal distribution with variance σ_a^2. Now, given an ARMA(p,q) model, we infer the expected true value $\hat{r}_t$ as:

$$\hat{r}_t = \phi_0 + \sum_{j=1}^{p} \phi_j r_{t-j} + \sum_{j=1}^{q} \theta_j a_{t-j}, \quad (2)$$

where (p, q) are non-negative integers denoting the model order, $\phi_1, \ldots, \phi_p$ are autoregressive coefficients, $\theta_1, \ldots, \theta_q$ are moving average coefficients, ϕ_o is a constant, and $t > max(p, q)$. More details regarding the estimation and choice of the model parameters (p, q) are described in Chapter 3 in [12].

Variable Thresholding Metric.
We propose another dynamic density metric, termed *variable thresholding metric*, that differs in two ways from the uniform thresholding metric. First, the variable thresholding metric works on Gaussian distributions, while the uniform thresholding metric is applicable only to uniform distributions. Second, unlike the uniform thresholding metric, the variable thresholding metric does not require the user-defined threshold for specifying uncertainty ranges. Instead, it computes a sample variance s_t^2 for a window S_{t-1}^H, so that s_t^2 is used to model a Gaussian distribution.

Given S_{t-1}^H, the variable thresholding metric infers a normal distribution at time t as:

$$p_t(R_t = r_t) = \frac{1}{\sqrt{2\pi s_t^2}} e^{-(r_t - \hat{r}_t)^2 / 2s_t^2}, \quad (3)$$

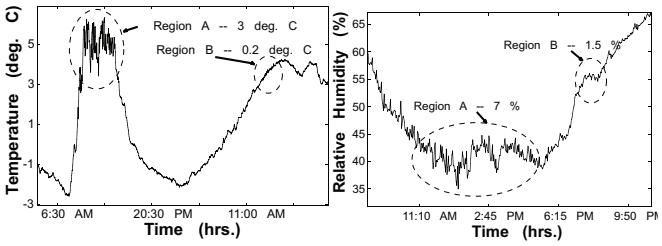
where $\hat{r}_t$ is an expected true value inferred by the ARMA model.

Fig. 3(b) demonstrates an example of estimating normal distributions based on the variable thresholding metric. First, the ARMA model infers the expected true values $\hat{r}_t$ that are used as the mean values for the normal distributions. It then computes the variances that are used to derive the standard deviations s_t.

IV. GARCH Metric

As stated in the previous section, it is common to capture the uncertainty of an imprecise time series with a fixed-size uncertainty range as shown in Fig. 3(a) [1], [14]. This approach, however, may not be effective in practice, since in a wide variety of real-world settings, the size of the uncertainty range typically varies over time. For example, Fig. 4 shows two time series obtained from a real sensor network deployment monitoring ambient temperature and relative humidity. The regions marked as *Region A* in Fig. 4(a) and Fig. 4(b) exhibit higher volatility[1] than those marked as *Region B*. This

[1]We use *variance* and *volatility* interchangeably.



(a) (b)

Fig. 4: Regions of changing volatility in (a) ambient temperature and (b) relative humidity.

observation strongly suggests that the underlying model should support time-varying variance and mean value when it infers a probability density function. We experimentally verify this claim in Section VII-D.

Motivated by this, we introduce a new dynamic density metric, the *GARCH metric*. The GARCH metric models $p_t(R_t)$ as a Gaussian probability density function $\mathcal{N}(\hat{r}_t, \hat{\sigma}_t^2)$. This metric assumes that the underlying time series exhibits not only time-varying average behavior ($\hat{r}_t$) but also time-varying variance ($\hat{\sigma}_t^2$). For inferring $\hat{\sigma}_t^2$ we propose using the GARCH model. And, for inferring $\hat{r}_t$ we can either use the ARMA model from Section III or Kalman Filters.

A. The GARCH Model

The GARCH (Generalized AutoRegressive Conditional Heteroskedasticity) model [12] efficiently captures time-varying volatility in a time series. Specifically, given a window S_{t-1}^H, the ARMA model models $r_i = \hat{r}_i + a_i$ where $t - H \leq i \leq t - 1$.

We then define the conditional variance σ_i^2 as:

$$\sigma_i^2 = \mathbb{E}((r_i - \hat{r}_i)^2 | F_{i-1}), \quad \sigma_i^2 = \mathbb{E}(a_i^2 | F_{i-1}), \tag{4}$$

where $\mathbb{E}(a_i^2|F_{i-1})$ is the variance of a_i given all the information F_{i-1} available until time $i-1$. The GARCH(m,s) model models volatility in (4) as a linear function of a_i^2 as:

$$a_i = \sigma_i \epsilon_i, \quad \sigma_i^2 = \alpha_0 + \sum_{j=1}^{m} \alpha_j a_{i-j}^2 + \sum_{j=1}^{s} \beta_j \sigma_{i-j}^2, \tag{5}$$

where ϵ_i is a sequence of independent and identically distributed (*i.i.d*) random variables, (m, s) are parameters describing the model order, $\alpha_0 > 0$, $\alpha_j \geq 0$, $\beta_j \geq 0$, $\sum_{j=1}^{max(m,s)} (\alpha_j + \beta_j) < 1$, and i takes values between $t - H + max(m, s)$ and $t - 1$.

The underlying idea of the GARCH(m,s) model is to reflect the fact that large shocks (a_i) tend to be followed by other large shocks. Unlike the s_t^2 in the variable thresholding metric, σ_i^2 is a variance that is estimated after subtracting the local trend $\hat{r}_i$. In many practical applications the GARCH model is typically used as the GARCH(1,1) model, since for a higher order GARCH model specifying the model order is a difficult task [12]. Thus, we restrict ourselves to these model order settings. More details regarding the estimation of model parameters and the choice for the sliding window size H are described in [12].

For inferring time-varying volatility, we use the GARCH(m,s) model and a_i as follows:

$$\hat{\sigma}_t^2 = \alpha_0 + \sum_{j=1}^{m} \alpha_j a_{t-j}^2 + \sum_{j=1}^{s} \beta_j \sigma_{t-j}^2. \tag{6}$$

Recall that we use the ARMA model for inferring the value of $\hat{r}_t$ given S_{t-1}^H. We also consider the Kalman Filter [12] for inferring $\hat{r}_t$. We show the difference in performance between the Kalman Filter and the ARMA model in Section VII-A. Basically, the Kalman Filter models $\hat{r}_t$ using the following two equations,

$$\text{state equation: } \hat{r}_i = c_1 \cdot \hat{r}_{i-1} + e_{i-1} \quad e_i \sim \mathcal{N}(0, \sigma_e^2), \tag{7}$$

$$\text{observation equation: } r_i = c_2 \cdot \hat{r}_i + \eta_i \quad \eta_i \sim \mathcal{N}(0, \sigma_\eta^2), \tag{8}$$

where $\hat{r}_1$ is given a priori and c_1 and c_2 are constants. Since the GARCH model in (5) takes errors a_i as input, they are computed as $a_i = r_i - \hat{r}_i$ and are used by the GARCH model.

Considering both approaches for inferring $\hat{r}_t$ (ARMA model and Kalman Filter) we propose two dynamic density metrics, namely, *ARMA-GARCH* and *Kalman-GARCH*. Both of them use the GARCH model for inferring $\hat{\sigma}_t$. But for inferring $\hat{r}_t$ they use ARMA model and Kalman Filter respectively.

Algorithm 1 Inferring $\hat{r}_t$ and $\hat{\sigma}_t^2$ using ARMA-GARCH.

Input: ARMA model parameters (p, q), sliding window S_{t-1}^H, and scaling factor κ.
Output: Inferred $\hat{r}_t$, inferred volatility $\hat{\sigma}_t^2$, and κ-scaled bounds u_b, l_b.
1: Estimate an $ARMA(p, q)$ model on S_{t-1}^H and obtain a_i where $t - H + max(p, q) \leq i \leq t - 1$
2: Estimate a $GARCH(1, 1)$ model using a_i's
3: Infer $\hat{r}_t$ using $ARMA(p, q)$ and $\hat{\sigma}_t^2$ using $GARCH(1, 1)$
4: $u_b \leftarrow \hat{r}_t + \kappa\hat{\sigma}_t$ and $l_b \leftarrow \hat{r}_t - \kappa\hat{\sigma}_t$
5: **return** $\hat{r}_t$, $\hat{\sigma}_t^2$, u_b, and l_b

Algorithm 1 gives a concise description of the ARMA-GARCH metric. This algorithm uses the ARMA model for inferring $\hat{r}_t$ and the GARCH model for inferring $\hat{\sigma}_t^2$ (Step 3). The algorithm for Kalman-GARCH metric is the same as Algorithm 1, except that it uses the Kalman filter in Step 3 for inferring $\hat{r}_t$ instead of using the ARMA model. Here, $\kappa \geq 0$ is a scaling factor that decides the upper bound u_b and the lower bound l_b. For example, when $\kappa = 3$, the probability that r_t lies between u_b and l_b is very high (approximately 0.9973).

The time complexities of the estimation step for the ARMA model and the GARCH model (Step 1 and 2) are $\mathcal{O}(H \cdot max(p, q))$ and $\mathcal{O}(H \cdot max(m, s))$ respectively [16]. Nevertheless, as the model order parameters are small as compared to H these estimation steps become significantly efficient.

V. Enhanced GARCH Metric

In practice, time series often contain values that are erroneous in nature. For example, sensor networks, like weather monitoring stations, frequently produce erroneous values due to various reasons; such as loss of communication, sensor failures, etc. Unfortunately, the GARCH model is incapable

of functioning appropriately when input streams contain such erroneous values. This is because the GARCH model has been generally used over precise, certain, and clean data (e.g., stock market data). To tackle this problem, we propose an enhancement of the GARCH metric, which renders the GARCH metric robust against erroneous time-series inputs.

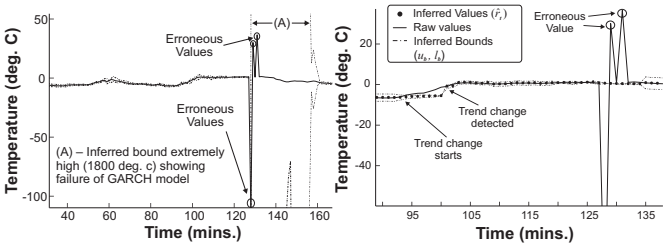
Before proceeding further, we note the difference between *erroneous values* and *imprecise values*. Imprecise values have an inherent element of uncertainty but still follow a particular trend, while erroneous values are significant outliers which exhibit large unnatural deviations from the trend.

To give an idea of the change in behavior exhibited by the GARCH model we run the ARMA-GARCH algorithm on all sliding windows S^H_{t-1} of a time series $S = \langle r_1, r_2, \ldots, r_{t_m} \rangle$ where $H + 1 \leq t \leq t_m$ and $\kappa = 3$. The result of executing this algorithm is shown in Fig. 5(a) along with the upper and lower bounds. Notice that at time 127, when the first erroneous value occurs in the training window, the GARCH model infers an extremely high volatility for the following time steps. This mainly happens since the GARCH equation (5) contains square terms, which significantly amplifies the effect of the presence of erroneous values. To avoid this we introduce novel heuristics which can be applied to input data in an online fashion and thus obtain a correct volatility estimate even in the presence of erroneous values. We term our approach C-GARCH (an acronym for Clean-GARCH).

A. C-GARCH Model

Let $S = \langle r_1, r_2, \ldots, r_{t_m} \rangle$ be a time series containing some erroneous values. We then start executing the ARMA-GARCH procedure (see Algorithm 1) at time $t > H$. For this we set $\kappa = 3$, thus making the probability of finding r_t outside the interval defined by u_b and l_b low. When we find that r_t resides outside u_b and l_b, we mark it as erroneous value and replace it with the corresponding inferred value $\hat{r}_t$. Simultaneously, we also keep the track of the number of consecutive values we have marked as erroneous values most recently. If this number exceeds a predefined constant $o_{c_{max}}$ then we assume that the observed raw values are exhibiting a changing trend. For example, during sunrise the ambient temperature exhibits a rapid change of trend. This idea inherently assumes that the probability of finding $o_{c_{max}}$ consecutive erroneous values is low. And, if we find $o_{c_{max}}$ consecutive erroneous values we should re-adjust the model to the new trend.

Although it rarely happens in practice that there are many consecutive erroneous values may be present in raw data. To rule out the possibility of using these values for inference, we introduce a novel heuristic that is applied to the values in the window $[r_{t-o_{c_{max}}}, \ldots, r_t]$ before they are used for the inference. This step ensures that we have not included any erroneous values present in the raw data into our system. Thus we avoid the problems that occur by using a simple ARMA-GARCH metric.



(a) (b)

Fig. 5: (a) Behavior of the GARCH model when window S^H_{t-1} contains erroneous values. (b) Result of using the C-GARCH model.

B. Successive Variance Reduction Filter

The heuristic that we use for filtering out significant anomalies is shown in Algorithm 2. This algorithm takes values $\mathcal{V} = [v_1, v_2, \ldots, v_K]$ containing erroneous values and a thresholding parameter SV_{max} as input. It first measures dispersion of $\mathcal{V}$ by computing its sample variance denoted as $SV(\mathcal{V})$ (Step 3). Then we delete a point, say v_k, and compute the sample variance of all the other points $[v_1, \ldots, v_{k-1}, v_{k+1}, \ldots, v_K]$ denoted as $SV(\mathcal{V} \backslash v_k)$ (Step 9). We perform this procedure for all points and then finally find a value $v_{\bar{k}}$ such that this value, if deleted, gives us the maximum variance reduction. We delete this point and reconstruct a new value at $\bar{k}$ using interpolation. We stop this procedure when the total sample variance becomes less than the variance threshold SV_{max}. In Steps 8 and 9, we use the intermediate values $\hat{v}'_K$ and $\hat{v}_K$ to compute $SV(\mathcal{V} \backslash v_K)$, thus reducing the computational complexity of the algorithm to quadratic.

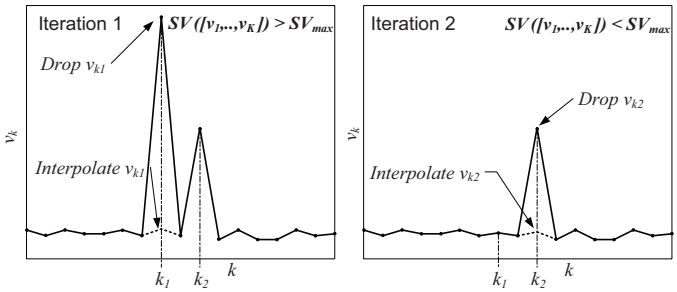


Fig. 6: Showing sample run of the Successive Variance Reduction Filter (Algorithm 2).

A graphical example of our approach is shown in Fig. 6. From this figure we can see that values at k_1 and k_2 are erroneous. In the first iteration our algorithm deletes value v_{k_1} and reconstructs it. Next, we delete v_{k_2} and obtain a new value using interpolation. At this point we stop since $SV(\mathcal{V})$ becomes less than SV_{max}. Moreover, it is very important to know a fair value for SV_{max}, since if a higher value is chosen we might include some erroneous values and if a lower value is chosen we might delete some non-erroneous values. Also, the value of SV_{max} depends on the underlying parameter monitored. For example, ambient temperature in Fig. 4 shows rapid changes in trend as compared to relative humidity. Thus, using a sample of size T of clean data, we compute SV_{max}

as the maximum sample variance (dispersion) we observe in all sliding windows of size $o_{c_{max}}$. This gives a fair estimate of the threshold between trend changes and erroneous values.

Fig. 5(b) shows the result of using C-GARCH model on the same data as shown in Fig. 5(a) with $o_{c_{max}} = 7$. We can observe that at $t = 93$ a trend change starts to occur and is smoothly corrected by the C-GARCH model at $t = 101$. Most importantly, the successive variance reduction filter effectively handles the erroneous values occurring at times $t = 127$ and $t = 132$. Thus the C-GARCH model performs as expected and overcomes the shortcomings of the plain ARMA-GARCH metric. In Section VII we will demonstrate the efficacy of the C-GARCH model on real data obtained from sensor networks.

Algorithm 2 The Successive Variance Reduction Filter.

Input: A time series $\mathcal{V}$ containing erroneous values and variance threshold SV_{max}.
Output: Cleaned values $\mathcal{V}$.

while true **do**
  $\hat{v}'_K \leftarrow \sum_{k=1}^{K} v_k^2$ and $\hat{v}_K \leftarrow \frac{1}{K}\sum_{k=1}^{K} v_k$
  $SV(\mathcal{V}) \leftarrow \frac{1}{K-1}\hat{v}'_K - \frac{K}{K-1}(\hat{v}_K)^2$
  if $SV(\mathcal{V}) > SV_{max}$ **then**
    break
  $cVar \leftarrow -\infty$, $\bar{k} \leftarrow 0$, and $k \leftarrow 1$
  repeat
    $\hat{v}'_{K-1} \leftarrow \hat{v}'_K - v_k^2$ and $\hat{v}_{K-1} \leftarrow \hat{v}_K - \frac{v_k}{K}$
    $SV(\mathcal{V}\backslash v_k) \leftarrow \frac{1}{K-2}\hat{v}_{K-1} - \frac{K-1}{K-2}(\hat{v}_{K-1})^2$
    if $SV(\mathcal{V}\backslash v_k) < cVar$ **then**
      $cVar \leftarrow SV(\mathcal{V}\backslash v_k)$
      $\bar{k} \leftarrow k$
    $k \leftarrow k + 1$
  until $k \leq K$
  Mark $v_{\bar{k}}$ as erroneous and delete
  if $\bar{k} \neq 1$ **and** $\bar{k} \neq K$ **then**
    Use $v_{\bar{k}-1}$ and $v_{\bar{k}+1}$ to interpolate the value of $v_{\bar{k}}$
  else
    Extrapolate $v_{\bar{k}}$

Guidelines for Parameter Setting: The C-GARCH model requires three parameters κ, SV_{max}, and $o_{c_{max}}$. In most cases we assign $\kappa = 3$. As seen before, SV_{max} is learned from a sample of clean data. On the contrary, setting $o_{c_{max}}$ requires domain knowledge about sensors used for data gathering. If there are unreliable sensors which frequently emit erroneous values then setting a higher value for $o_{c_{max}}$ is advisable and vice versa. Our experiments suggest that the C-GARCH model performs satisfactorily when the value for $o_{c_{max}}$ is set to twice the length of the longest sequence of erroneous values. In practice, $o_{c_{max}}$ is generally small, making the execution of Algorithm 2 efficient.

VI. Probabilistic View Generation

Recall Definition 2 that defines the query for generating probability values for a tuple-independent probabilistic database (view). To precisely specify the user-defined range Ω in the definition, we define $\Omega = \{\hat{r}_t + \lambda\Delta | \lambda = -\frac{n}{2}, \ldots, \frac{n}{2}\}$, where Δ is a positive real number and n is an even integer. We refer to Δ and n as *view parameters*. These parameters describe n ranges of size Δ around the expected true value $\hat{r}_t$. In the online mode of our system, the query is evaluated at each time when a new value is streamed to the system. In the offline mode, all necessary parameters can be specified by users using a SQL-like syntax. For example, the syntax in Fig. 7 creates the probabilistic view in Fig. 2.

```
CREATE VIEW prob_view AS DENSITY r
OVER t OMEGA delta=2, n=2
FROM raw_values WHERE t >= 1 AND t <= 3
```

Fig. 7: Example of the probabilistic view generation query.

In the example shown in Fig. 7, `AS DENSITY r OVER t` illustrates the time-varying density for time series `r`. The `OMEGA` clause specifies the ranges of the data values of the probabilistic view, and the `WHERE` clause defines a time interval. Notice that the query given in Definition 2 is evaluated at each time t to obtain Λ_t. Specifically, at each t and for each $\lambda = \{-\frac{n}{2}, \ldots, (\frac{n}{2} - 1)\}$ we compute the following integral:

$$\rho_\lambda = \int_{\hat{r}_t+\lambda\Delta}^{\hat{r}_t+(\lambda+1)\Delta} p_t(R_t)dR_t,$$
$$= P_t(R_t = \hat{r}_t + (\lambda+1)\Delta) - P_t(R_t = \hat{r}_t + \lambda\Delta), \quad (9)$$

where $P_t(R_t)$ is the cumulative distribution function of r_t.

In short, (9) involves computing $P_t(R_t)$ for each value of $\lambda = \{-\frac{n}{2}, \ldots, \frac{n}{2}\}$. Unfortunately, this computation may incur high cost when the time interval specified by the query spans over many days comprising of a large number of raw values. Moreover, this processing becomes significantly challenging when the query requests for a view with finer granularity (low Δ) and large range n, since such values for the view parameters considerably increase the computational cost.

To address this problem, we propose an approach that caches and reuses the computations of $P_t(R_t)$, which were already performed at earlier times. The intuition behind this approach is to observe that probability distributions for a time series do not generally exhibit dramatic changes in short terms. For example, temperature values often exhibit only slight changes within short time intervals. In addition, similar probability distributions may be found periodically (e.g., early morning hours every day). Thus, the query processing can take advantage of the results from previous computation. In the rest of this section, we introduce an effective caching mechanism, termed *σ–cache*, that substantially boosts the performance of query evaluation by caching the values of $P_t(R_t)$.

A. σ–cache

As introduced before, let $P_t(R_t)$ be a Gaussian cumulative distribution function of r_t at time t. If required for clarity, we denote it as $P_t(R_t; \hat{\Theta}_t)$ where $\hat{\Theta}_t = (\hat{r}_t, \hat{\sigma}_t^2)$. Observe that the shape of $P_t(R_t; \hat{\Theta}_t)$ is completely determined by $\hat{\sigma}_t^2$, since $\hat{r}_t$ only specifies the location of the curve traced by $P_t(R_t; \hat{\Theta}_t)$. This observation leads to an important property: suppose we move from time t to t', then the values of $P_t(R_t = \hat{r}_t + \lambda\Delta; \hat{\Theta}_t)$, $P_{t'}(R_{t'} = \hat{r}_{t'} + \lambda\Delta; \hat{\Theta}_{t'})$, and consequently ρ_λ are the same if $\hat{\sigma}_t$ is equal to $\hat{\sigma}_{t'}$. We illustrate this property graphically in Fig. 8. Moreover, since the shapes of $P_t(R_t; \hat{\Theta}_t)$

and $P_{t'}(R_{t'};\hat{\Theta}_{t'})$ solely depend on $\hat{\sigma}_t$ and $\hat{\sigma}_{t'}$ respectively, we can assume in the rest of the analysis that the mean values of $P_t(R_t)$ and $P_{t'}(R_{t'})$ are zero. This could be done using a simple mean shift operation on $P_t(R_t)$ and $P_{t'}(R_{t'})$.

Our aim is to approximate $P_{t'}(R_{t'})$ with $P_t(R_t)$. This is possible only if we know how the distance (similarity) between $P_t(R_t;\hat{\Theta}_t)$ and $P_{t'}(R_{t'};\hat{\Theta}_{t'})$ behaves as a function of $\hat{\sigma}_t$ and $\hat{\sigma}_{t'}$. If we know this relation then we can, with a certain error, approximate $P_{t'}(R_{t'};\hat{\Theta}_{t'})$ with $P_t(R_t;\hat{\Theta}_t)$ simply by looking up $\hat{\sigma}_t$ and $\hat{\sigma}_{t'}$. Thus, if we have already computed $P_t(R_t;\hat{\Theta}_t)$ at time t then we can reuse it at time t' to approximate $P_{t'}(R_{t'};\hat{\Theta}_{t'})$.

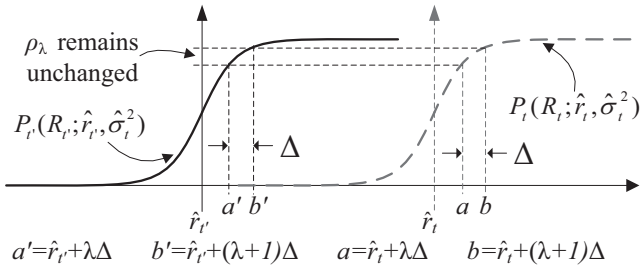


Fig. 8: An example illustrating that ρ_λ remains unchanged under mean shift operations when two Gaussian distributions have equal variance.

B. Constraint-Aware Caching

In practice, systems that use the σ–cache could have constraints of limited storage size or of error tolerance. To reflect this, we guarantee certain user-defined constraints. Specifically, we focus on the following:

- *Distance constraint* guarantees that the maximum approximation error is upper bounded by the distance constraint when the cache is used.
- *Memory constraint* guarantees that the cache does not use more memory than that specified by the memory constraint.

Before proceeding further, we first characterize the distance between two probability distributions using a measure known as the *Hellinger distance* [17]. It is a distance measure similar to the popular Kullback-Leibler divergence. However, unlike the Kullback-Leibler divergence, the Hellinger distance takes values between zero and one which makes its choice simple and intuitive. Formally, the square of Hellinger distance $\mathcal{H}$ between $P_t(R_t)$ and $P_{t'}(R_{t'})$ is given as:

$$\mathcal{H}^2[P_t(R_t), P_{t'}(R_{t'})] = 1 - \sqrt{\frac{2\hat{\sigma}_t\hat{\sigma}_{t'}}{\hat{\sigma}_t^2 + \hat{\sigma}_{t'}^2}}. \tag{10}$$

The Hellinger distance assigns minimum value of zero when $P_{t'}(R_{t'})$ and $P_t(R_t)$ are the same and vice versa.

Guaranteeing Distance Constraint.
We use the Hellinger distance to prove the following theorem that allows us to approximate $P_{t'}(R_{t'})$ with $P_t(R_t)$.

Theorem 1: Given $P_{t'}(R_{t'})$, $P_t(R_t)$, and a user-defined distance constraint $\mathcal{H}'$, we can approximate $P_{t'}(R_{t'})$ with $P_t(R_t)$, such that $\mathcal{H}[P_t(R_t), P_{t'}(R_{t'})] \leq \mathcal{H}'$, where $\hat{\sigma}_{t'} = d_s \cdot \hat{\sigma}_t$ and $\hat{\sigma}_{t'} > \hat{\sigma}_t$. The parameter d_s can be chosen as any value satisfying,

$$d_s \leq \frac{2 + \sqrt{4 - 4\left(1 - \mathcal{H}'^2\right)^4}}{2\left(1 - \mathcal{H}'^2\right)^2}. \tag{11}$$

Proof: Substituting $\hat{\sigma}_{t'} = d_s \cdot \hat{\sigma}_t$ in (10) we obtain,

$$(1 - \mathcal{H}'^2)\sqrt{1 + d_s^2} - \sqrt{2 \cdot d_s} = 0.$$

Solving for d_s we obtain,

$$d_s \leq \frac{2 + \sqrt{4 - 4\left(1 - \mathcal{H}'^2\right)^4}}{2\left(1 - \mathcal{H}'^2\right)^2}.$$

Since d_s is monotonically increasing in $\mathcal{H}'$, choosing a value of d_s as given by the above inequality guarantees the distance constraint $\mathcal{H}'$. ■

The above theorem states that if we have a user-defined *distance constraint* $\mathcal{H}'$ then we can approximate $P_{t'}(R_{t'})$ by $P_t(R_t)$ only if $\hat{\sigma}_{t'} > \hat{\sigma}_t$ and d_s is chosen using (11). Moreover, since d_s is defined as the ratio between $\hat{\sigma}_{t'}$ and $\hat{\sigma}_t$ we call it the *ratio threshold*.

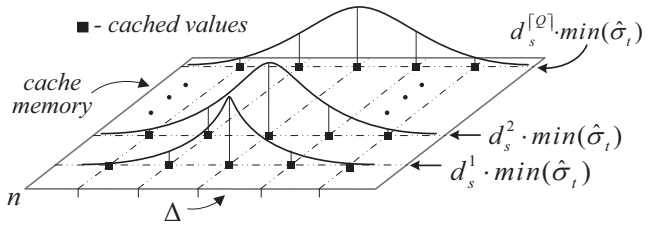


Fig. 9: Structure of the σ–cache.

Now, we describe how Theorem 1 allows us to efficiently store and reuse values of $P_t(R_t)$ while query processing. First, we compute the maximum and minimum values amongst all $\hat{\sigma}_t$ matching the `WHERE` clause of the probabilistic view generation query (see Fig. 7). Let us denote these extremes as $max(\hat{\sigma}_t)$ and $min(\hat{\sigma}_t)$. We then define the *maximum ratio threshold* $\mathcal{D}_s$ as,

$$\mathcal{D}_s = \frac{max(\hat{\sigma}_t)}{min(\hat{\sigma}_t)}. \tag{12}$$

Given the user-defined distance constraint $\mathcal{H}'$ we use (11) to obtain a suitable value for d_s. Then we compute a $\mathcal{Q}$, such that,

$$max(\hat{\sigma}_t) = d_s^{\mathcal{Q}} \cdot min(\hat{\sigma}_t). \tag{13}$$

Let $\lceil x \rceil$ denote the smallest integer value that is not smaller than x. Then, $\lceil\mathcal{Q}\rceil$ gives us the maximum number of distributions that we should cache such that the distance constraint is satisfied. We populate the cache by pre-computing values for $\lceil\mathcal{Q}\rceil$ distributions having standard deviations $d_s^q \cdot min(\hat{\sigma}_t)$, where $q = 1, 2, \ldots, \lceil\mathcal{Q}\rceil$. As shown in Fig. 9, these values are computed at points specified by the view parameters Δ and n.

We store each of these pre-computed distributions in a sorted container like a B-tree along with key $d_s^q \cdot min(\hat{\sigma}_t)$. When we need to compute $P_{t'}(R_{t'};\hat{\Theta}_{t'})$, we first look up the container to find keys $d_s^q \cdot min(\hat{\sigma}_t)$ and $d_s^{q+1} \cdot min(\hat{\sigma}_t)$,

such that $\hat{\sigma}_{t'}$ lies between them. We then use the values associated with key $d_s^q \cdot min(\hat{\sigma}_t)$ for approximating $P_{t'}(R_{t'})$. By following this procedure we always guarantee that the distance constraint is satisfied due to Theorem 1.

Guaranteeing Memory Constraint.
Let us assume that we have a user-defined memory constraint $\mathcal{M}$. We then consider an integer $\mathcal{Q}'$ which indicates the maximum number of distributions that can be stored in the memory size $\mathcal{M}$. Here we prove an important theorem that enables the guarantee for memory constraint.

Theorem 2: Given the values of $\mathcal{Q}'$, $max(\hat{\sigma}_t)$, and $min(\hat{\sigma}_t)$, the memory constraint $\mathcal{M}$ is satisfied if and only if the value of the ratio threshold d_s is chosen as,

$$d_s \geq \mathcal{D}_s^{\frac{1}{\mathcal{Q}'}}. \tag{14}$$

Proof: From (13) we obtain,

$$\log_e(max(\hat{\sigma}_t)) = \mathcal{Q}' \cdot \log_e(d_s) + \log_e(min(\hat{\sigma}_t)),$$
$$d_s = max(\hat{\sigma}_t)^{\frac{1}{\mathcal{Q}'}} \cdot min(\hat{\sigma}_t)^{-\frac{1}{\mathcal{Q}'}}.$$

From the above equation we can see that d_s is monotonically decreasing in $\mathcal{Q}'$. Since $\mathcal{D}_s = \frac{max(\hat{\sigma}_t)}{min(\hat{\sigma}_t)}$, we obtain,

$$d_s \geq \mathcal{D}_s^{\frac{1}{\mathcal{Q}'}}.$$

Choosing a value for d_s as given in the above equation guarantees that at most $\mathcal{Q}'$ distributions are stored, thus guaranteeing the memory constraint $\mathcal{M}$. ■

The above theorem states that given user-defined memory constraint $\mathcal{Q}'$ we set d_s according to (14) so as not to store more than $\mathcal{Q}'$ distributions. Also, given a distance constraint $\mathcal{H}'$ the rate at which the memory requirement grows is $\mathcal{O}(log(\mathcal{D}_s))$. Thus the cache size does not depend on the *number* of tuples that match the `WHERE` clause of the query in Fig. 7. Instead, it only grows logarithmically with the ratio between $max(\hat{\sigma}_t)$ and $min(\hat{\sigma}_t)$. Observe that the number of distributions stored by the σ–cache is independent from the view parameters Δ and n. This is a desirable property since it implies that, queries with finer granularity are answered by storing the same number of distributions.

There is an interesting trade-off between the distance constraint and the memory constraint (see (11) and (14)). When the distance constraint increases, the amount of memory required by the σ–cache decreases in order to guarantee the distance constraint and vice versa. Thus, as expected, there exists a give-and-take relationship between available memory size and prescribed error tolerance.

In the following section, we will demonstrate significant improvement with respect to query processing by using the σ–cache.

VII. Experimental Evaluation

The main goals of our experimental study are fourfold. First, we show that the performance of the proposed dynamic density metrics, namely, ARMA-GARCH and Kalman-GARCH are efficient and accurate over real-world data. Second, we compare the performance of the ARMA-GARCH metric with that of the C-GARCH enhancement, in order to show that C-GARCH is efficient as well as accurate in handling erroneous values in time series. We then demonstrate that the use of the σ–cache significantly increases query processing performance. Lastly, we perform experiments validating that real world datasets exhibit regimes of changing volatility.

In our experiments, we use two real datasets, details of these datasets are as follows:

Campus Dataset: This dataset comprises of ambient temperature values recorded over twenty five days. It consists of approximately eighteen thousand samples. These values are obtained from a real sensor network deployment on the EPFL university campus in Lausanne, Switzerland. We refer to this dataset as *campus-data*.

Moving Object Dataset: This dataset consists of GPS logs recorded from on-board navigation systems in 192 cars in Copenhagen, Denmark. Each log entry consists of time and x-y coordinate values. In our evaluation we use only x-coordinate values. This dataset contains approximately ten thousand samples recorded over five and half hours. We refer to this dataset as *car-data*.

Table II provides a summary of important properties of both datasets. We have implemented all our methods using MATLAB Ver. 7.9 and Java Ver. 6.0. We use a Intel Dual Core 2 GHz machine having 3GB of main memory for performing the experiments.

	campus-data	*car-data*
Monitored parameter	Temperature	GPS Position
Number of data values	18031	10473
Sensor accuracy	± 0.3 deg. C	± 10 meters
Sampling interval	2 minutes	1-2 seconds

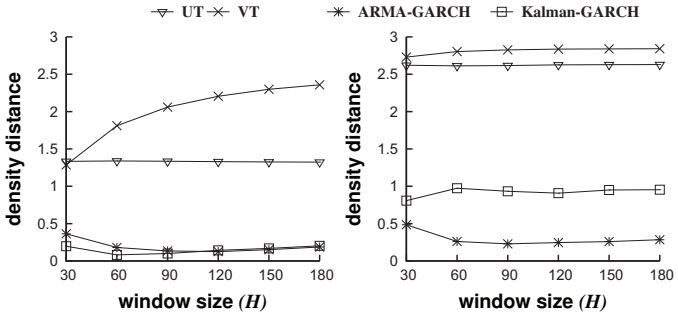
TABLE II: Summary of datasets.

A. Comparison of Dynamic Density Metrics

We compare our main proposals (ARMA-GARCH and Kalman-GARCH) with uniform thresholding (UT) and variable thresholding (VT). These evaluations are performed on both datasets. As described in Section II, we used the density distance for comparing the quality of distributions obtained using the dynamic density metrics.

Fig. 10 shows a comparison of density distance for the various dynamic density metrics for both datasets along with increasing window size (H). Clearly, both the ARMA-GARCH metric and the Kalman-GARCH metric outperform the naive density metrics. Specifically, those advanced dynamic density metrics outperform the naive density metrics by giving *upto 20 times and 12.3 times lower* density distances for *campus-data* and *car-data* respectively.

Among the advanced dynamic density metrics, the ARMA-GARCH metric performs better than all the other metrics. For *car-data* we can observe that the Kalman-GARCH metric gives low accuracy as the window size increases. This behavior is expected since when larger window sizes are used for the Kalman Filter, there is a greater chance of error in inferring $\hat{r}_t$. In our observation, the use of smaller window sizes (e.g.,



(a) campus-data *(b) car-data*

Fig. 10: Comparing quality of the dynamic density metrics.

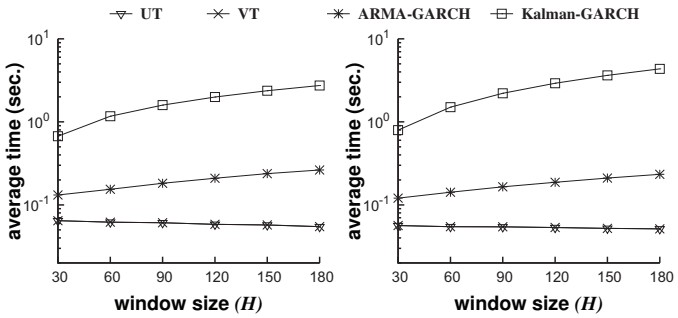
$H = 10$) for the Kalman-GARCH metric performs twice better, compared to the ARMA-GARCH metric.

Next, we compare the efficiency of the dynamic density metrics. Fig. 11 shows the average times required to perform one iteration of density inference. Because of the large performance gain of the ARMA-GARCH metric, the execution times are shown on logarithmic scale. The ARMA-GARCH metric achieves a *factor of 5.1 to 18.6 speedup* over the Kalman-GARCH metric. This is due to slow convergence of the iterative EM (Expectation-Maximization) algorithm used for estimating parameters of the Kalman Filter. Thus, unlike the ARMA model, computing parameters for the Kalman Filter takes longer for large window sizes. The naive dynamic density metrics are much more efficient than the Kalman-GARCH metric. But they are only marginally better than the ARMA-GARCH metric. Overall the ARMA-GARCH metric shows excellent characteristics in terms of both efficiency and accuracy.

In the next set of experiments, we discuss the effect of model order of an ARMA(p,0) model on density distance. Fig. 12 shows the density distance obtained by using several metrics when the model order p increases. Observe that for the ARMA-GARCH metric the density distance increases with model order. This justifies our choice of a low model order for the ARMA-GARCH metric.

B. Impact of C-GARCH

In the following, we demonstrate the improved performance of the C-GARCH model by comparing it with the plain ARMA-GARCH metric using *campus-data* (we omit the results from *car-data* because they are similar). We start by inserting erroneous values synthetically, since for comparing accuracy we should know beforehand the number of erroneous values present in the data. The insertion procedure inserts a pre-specified number of very high (or very low) values uniformly at random in the data.



(a) campus-data *(b) car-data*

Fig. 11: Comparing efficiency of the dynamic density metrics. Note the logarithmic scale on the y-axis.

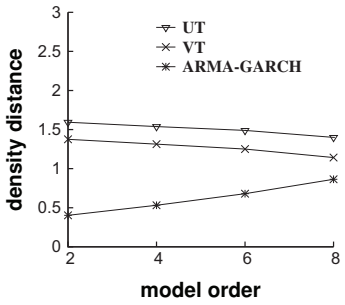
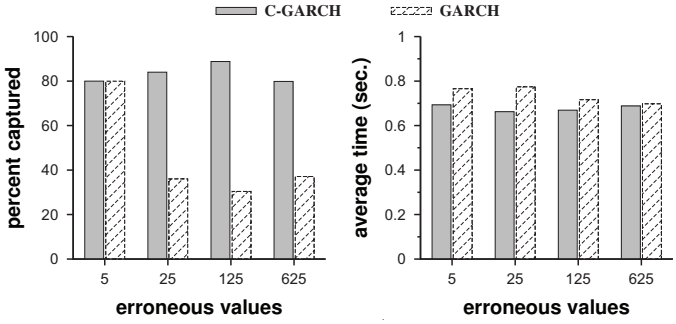


Fig. 12: Effect of model order on *campus-data*.

For evaluating the C-GARCH approach we first compute SV_{max} using a given set of clean values and then execute the C-GARCH model while setting $o_{c_{max}} = 8$. Fig. 13(a) compares the percentage of total erroneous values detected for C-GARCH and ARMA-GARCH. Admittedly, the C-GARCH approach is *more than twice effective* in detecting and cleaning erroneous values. Additionally, from Fig. 13(b) it can be observed that the C-GARCH approach does not require excessive computational cost as compared to ARMA-GARCH. The reason is that the ARMA model estimation takes more time if there are erroneous values in the window S^H_{t-1}. This additional time offsets the time spent by the C-GARCH model in cleaning erroneous values before they are given to the ARMA-GARCH metric.



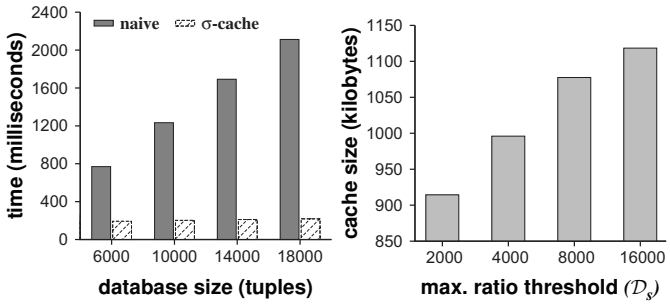
(a) *(b)*

Fig. 13: Comparing C-GARCH and GARCH. (a) Percentage of erroneous values successfully detected and (b) average time for processing a single value.

C. Impact of using σ–cache

Next, we show the impact of using the σ–cache while creating a probabilistic database. Particularly, we are interested in knowing the increase in efficiency obtained from using a σ–cache. Moreover, we are also interested in verifying the rate at which the size of the σ–cache grows as the maximum ratio threshold $\mathcal{D}_s$ increases. Here, we expect the cache size to grow logarithmically in $\mathcal{D}_s$.

We use *campus-data* for demonstrating the space and time efficiency of the σ–cache. We choose $\Delta = 0.05$, $n = 300$, Hellinger distance $\mathcal{H} = 0.01$, and compute d_s using (11). Fig. 14(a) shows the improvement in efficiency obtained for the probabilistic view generation query with increasing number of tuples. Here, the naive approach signifies that the σ–cache is not used for storing and reusing previous computation. In



(a) (b)

Fig. 14: (a) Impact of using the σ–cache on efficiency. (b) Scaling behavior of the σ–cache. Note the exponential scale on the x-axis.

Fig. 14 all values are computed by taking an ensemble average over ten independent executions. Clearly, using the σ–cache exhibits manyfold improvements in efficiency. For example, when there are 18K raw value tuples we observe a *factor of 9.6 speedup* over the naive approach. Fig. 14(b) shows the memory consumed by the σ–cache as $\mathcal{D}_s$ is increased. As expected, the cache size grows only logarithmically as the maximum ratio threshold $\mathcal{D}_s$ increases. This proves that the σ-cache is a space- and time-efficient method for seamlessly caching and reusing computation.

D. Verifying Time-varying Volatility

Before we infer time-varying volatility using the ARMA-GARCH metric or the Kalman-GARCH metric it is important to verify whether a given time series exhibits changes in volatility over time. For testing this we use a null hypothesis test proposed in [12]. The null hypothesis tests whether the errors obtained from using a ARMA model (a_i^2) are independent and identically distributed (*i.i.d*). This is equivalent to testing whether $\xi_1 = \cdots = \xi_m = 0$ in the linear regression,

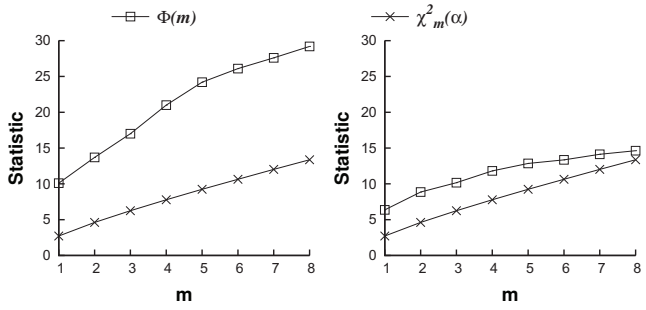
$$a_i^2 = \xi_0 + \xi_1 a_{i-1}^2 + \cdots + \xi_m a_{i-m}^2 + e_i, \tag{15}$$

where $i \in \{m+1, \ldots, H\}$, e_i denotes the error term, $m \geq 1$, and H is the window size. If we reject the null hypothesis (i.e., $\xi_j \neq 0$) then we can say that the errors are not *i.i.d*, thus establishing that the given time series exhibits time-varying volatility. First, we start by computing the sample variance of a_i^2 and e_i denoted as γ_0 and γ_1 respectively. Then,

$$\Phi(m) = \frac{(\gamma_0 - \gamma_1)/m}{\gamma_1/(K - 2m - 1)}, \tag{16}$$

is asymptotically distributed as a chi-square distribution χ_m^2 with m degrees of freedom. Thus we reject the null hypothesis if $\Phi(m) > \chi_m^2(\alpha)$, where $\chi_m^2(\alpha)$ is in the upper $100(1-\alpha)^{th}$ percentile of χ_m^2 or the p-value of $\Phi(m) < \alpha$ [12]. In our experiments we choose $\alpha = 0.05$.

To show that our datasets exhibit regimes of changing volatility we compute the value of $\Phi(m)$ where $m = \{1, 2, \ldots, 8\}$ on 1800 windows containing 180 samples each (i.e., $H = 180$) for *campus-data* and *car-data*. Then we reject the null hypothesis if the average value of $\Phi(m)$ over all windows is greater than $\chi_m^2(\alpha)$.



(a) campus-data *(b) car-data*

Fig. 15: Verifying time-varying volatility.

Fig. 15 shows the results from this evaluation. Clearly, we can reject the null hypothesis for both datasets because for all values of m, $\chi_m^2(\alpha)$ is much lower than $\Phi(m)$. This means that a_i^2 are not *i.i.d* and thus we can find regimes of changing volatility. Interestingly, for *car-data* (see Fig. 15(b)) we can see that $\chi_m^2(\alpha)$ and $\Phi(m)$ are close to each other. Thus the *car-data* contains less time-varying volatility as compared to the *campus-data*.

The above results support the claim that real datasets show change of volatility with time, thus justifying the use of the GARCH model.

VIII. Related Work

In order to effectively deal with uncertain data, a vast body of research on probabilistic databases has been conducted in the literature, including concepts and foundations [18]–[20], query processing [3], [4], [21], [22], and indexing schemes [5], [23], [24]. All these studies, however, share the common condition that probability values associated with data must be given a priori. As a result, a large variety of applications are still incapable of receiving benefits from such well-established tools for processing probabilistic databases, due to the lack of methods for establishing the required probability values.

Some previous work highlights the fact that creating probabilistic databases is a non-trivial problem. They then propose effective solutions for the problem; however, the studies have only limited scope for domain-specific applications, such as handling duplicated data records [8], [9] and building structured data from unstructured data [10].

More recently, the concept of probabilistic databases has been extended into stream data processing, so-called *probabilistic streams* [6], [7], [24]. Ré *et al.* [7] propose a framework for query processing over probabilistic (Markovian) streams. Later, an access method for such Markovian streams is introduced in [24] for efficient query processing. Cormode and

Garofalakis [6] also propose efficient algorithms based on hash-based sketch synopsis structure for processing aggregate queries over probabilistic streams. While all these studies assume probabilistic streams are given beforehand, Tran *et al.* [11] introduce a complete solution to create probabilistic streams. Unfortunately, this proposal is focused on RFID data, whereas our solution accepts arbitrary time-series data including such RFID data.

Processing probabilistic queries is another related area to our work. Cheng *et al.* [1] introduce several important types of probabilistic queries, as well as a generic query evaluation framework over inherently imprecise data. Although they assume that an uncertainty bound for data can be easily given by users, the assumption may not hold in many real-world applications. Deshpande and Madden [25] introduce the abstraction of *model-based views* that are database views created from the underlying data by applying numerical models. These views are then used for query processing instead of using the actual data. This idea is then extended by Kanagal and Deshpande [26], in which various particle filters are used for generating model-based views. This proposal requires a sufficient number of generated particles to obtain reliable probabilistic inferences, however, this substantially decreases the efficiency of the system.

Some prior research focuses on system perspectives associated with uncertain data. Wang *et al.* [27] introduce BayesStore which stores joint probability distribution functions encoded in a Bayesian network. Jampani *et al.* [28] propose a novel concept, by which the system does not store probabilities but parameters for generating the probabilities. Our work inherits this idea. Antova *et al.* [29] introduce the abstractions of world-sets and world-tables for capturing attribute-level uncertainty and possible world semantics of a probabilistic database. Cheng *et al.* [30] propose U-DBMS for managing uncertain data where the probability density function for the uncertain attributes is pre-specified.

IX. Conclusions

Due to the lack of methods for generating probabilistic databases, a large variety of applications that are built on (imprecise) time series are still incapable of having benefits from well-established tools for processing probabilistic databases. To address this, we proposed a novel and generic solution for creating probabilistic databases from imprecise time-series data. Our proposal includes two novel components: the dynamic density metrics that effectively infer time-dependent probability distributions for time series and the Ω–View builder that uses the inferred distributions for creating probabilistic databases. We also introduced the σ–cache that enables efficient creation of probabilistic databases while obeying user-defined constraints. Comprehensive experiments highlight the effectiveness of our approach.

References

[1] R. Cheng, D. V. Kalashnikov, and S. Prabhakar, "Evaluating probabilistic queries over imprecise data," in *SIGMOD*, 2003, pp. 551–562.

[2] M. Hua, J. Pei, W. Zhang, and X. Lin, "Ranking queries on uncertain data: a probabilistic threshold approach," in *SIGMOD*, 2008, pp. 673–686.

[3] N. Dalvi and D. Suciu, "Efficient query evaluation on probabilistic databases," *The VLDB Journal*, vol. 16, no. 4, pp. 523–544, 2007.

[4] D. Olteanu, J. Huang, and C. Koch, "SPROUT: Lazy vs. eager query plans for tuple-independent probabilistic databases," in *ICDE*, 2009, pp. 640–651.

[5] Y. Tao, R. Cheng, X. Xiao, W. K. Ngai, B. Kao, and S. Prabhakar, "Indexing multi-dimensional uncertain data with arbitrary probability density functions," in *VLDB*, 2005, pp. 922–933.

[6] G. Cormode and M. Garofalakis, "Sketching probabilistic data streams," in *SIGMOD*, 2007, pp. 281–292.

[7] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu, "Event queries on correlated probabilistic streams," in *SIGMOD*, 2008, pp. 715–728.

[8] P. Andritsos, A. Fuxman, and R. J. Miller, "Clean answers over dirty databases: A probabilistic approach," in *ICDE*, 2006, p. 30.

[9] O. Hassanzadeh and R. J. Miller, "Creating probabilistic databases from duplicated data," *The VLDB Journal*, vol. 18, no. 5, pp. 1141–1166, 2009.

[10] R. Gupta and S. Sarawagi, "Creating probabilistic databases from information extraction models," in *VLDB*, 2006, pp. 965–976.

[11] T. Tran, C. Sutton, R. Cocci, Y. Nie, Y. Diao, and P. Shenoy, "Probabilistic inference over RFID streams in mobile environments."

[12] R. Shumway and D. Stoffer, *Time series analysis and its applications*. Springer-Verlag, New York, 2005.

[13] F. Diebold, T. Gunther, and A. Tay, "Evaluating density forecasts with applications to financial risk management," *International Economic Review*, vol. 39, no. 4, pp. 863–883, 1998.

[14] R. Cheng, Y. Xia, S. Prabhakar, R. Shah, and J. S. Vitter, "Efficient indexing methods for probabilistic threshold queries over uncertain data," in *VLDB*, 2004, pp. 876–887.

[15] D. Tulone and S. Madden, "PAQ: Time series forecasting for approximate query answering in sensor networks," in *EWSN*, 2006, pp. 21–37.

[16] T. Minka, "A comparison of numerical optimizers for logistic regression," 2007.

[17] D. Pollard, *A user's guide to measure theoretic probability*. Cambridge University Press, 2002.

[18] R. Cavallo and M. Pittarelli, "The theory of probabilistic databases," in *VLDB*, 1987, pp. 71–81.

[19] L. V. S. Lakshmanan, N. Leone, R. Ross, and V. S. Subrahmanian, "ProbView: A flexible probabilistic database system," *ACM TODS*, vol. 22, no. 3, pp. 419–469, 1997.

[20] N. Dalvi and D. Suciu, "Management of probabilistic data: foundations and challenges," in *PODS*, 2007, pp. 1–12.

[21] N. Khoussainova, M. Balazinska, and D. Suciu, "Towards correcting input data errors probabilistically using integrity constraints," in *MobiDE*, 2006, pp. 43–50.

[22] C. Mayfield, J. Neville, and S. Prabhakar, "ERACER: A database approach for statistical inference and data cleaning."

[23] B. Kanagal and A. Deshpande, "Indexing correlated probabilistic databases," in *SIGMOD*, 2009, pp. 455–468.

[24] J. Letchner, C. Re, M. Balazinska, and M. Philipose, "Access methods for Markovian streams," in *ICDE*, 2009, pp. 246–257.

[25] A. Deshpande and S. Madden, "MauveDB: Supporting model-based user views in database systems," in *SIGMOD*, 2006, pp. 73–84.

[26] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *ICDE*, 2008, pp. 1160–1169.

[27] D. Wang, E. Michelakis, M. Garofalakis, and J. Hellerstein, "BayesStore: Managing large, uncertain data repositories with probabilistic graphical models," *PVLDB*, vol. 1, no. 1, pp. 340–351, 2008.

[28] R. Jampani, F. Xu, M. Wu, L. L. Perez, C. Jermaine, and P. J. Haas, "MCDB: A monte carlo approach to managing uncertain data," in *SIGMOD*, 2008, pp. 687–700.

[29] L. Antova, T. Jansen, C. Koch, and D. Olteanu, "Fast and simple relational processing of uncertain data," in *ICDE*, 2008.

[30] R. Cheng, S. Singh, and S. Prabhakar, "U-DBMS: A database system for managing constantly-evolving data," in *VLDB*, 2005, pp. 1271–1274.