Efficiently Answering Durability Prediction Queries

Junyang Gao* jygao@google.com Google
Yifan Xu xuyifa@amazon.com Amazon.com
Pankaj K. Agarwal pankaj@cs.duke.edu Duke University
Jun Yang junyang@cs.duke.edu Duke University

ABSTRACT

We consider a class of queries called *durability prediction queries* that arise commonly in predictive analytics, where we use a given predictive model to answer questions about possible futures to inform our decisions. Examples of durability prediction queries include "what is the probability that this financial product will keep losing money over the next 12 quarters before turning in any profit?" and "what is the chance for our proposed server cluster to fail the required service-level agreement before its term ends?" We devise a general method called *Multi-Level Splitting Sampling (MLSS)* that can efficiently handle complex queries and complex models—including those involving black-box functions—as long as the models allow us to simulate possible futures step by step. Our method addresses the inefficiency of standard Monte Carlo (MC) methods by applying the idea of *importance splitting* to let one "promising" sample path prefix generate multiple "offspring" paths, thereby directing simulation efforts toward more promising paths. We propose practical techniques for designing splitting strategies, freeing users from manual tuning. Experiments show that our approach is able to achieve unbiased estimates and the same error guarantees as standard MC while offering an order-of-magnitude cost reduction.

ACM Reference Format:
Junyang Gao, Yifan Xu, Pankaj K. Agarwal, and Jun Yang. 2021. Efficiently Answering Durability Prediction Queries. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3457305

1 INTRODUCTION

Increasingly, we rely on *predictive analytics* to inform decision making. Typically, we build a model to predict the future, using historical data and expert domain knowledge. Then, using this model, we can ask questions about possible futures to inform our decisions. A common type of such questions are what we call *durability prediction queries*, which predict how likely is it that a condition will remain over a given duration into the future. For example, business analysts ask durability prediction queries for financial risk assessments: "what is the probability that this financial product will keep losing money over the next 12 quarters before turning in any profit?" or "how likely is it that our client will not have a default on the mortgage loan in the next five years?" As another example, engineers ask durability prediction queries about reliability: "what is the probability that a self-driving car makes a serious misjudgement within its warranty period?" or "what is the chance for our proposed server cluster to fail the required service-level agreement before its term ends?" In this paper, we consider the problem of answering durability prediction queries given a predictive model of the future.

Durability prediction queries are challenging for several reasons. In contrast to queries over historical data, these queries must deal with uncertainties about the possible futures. Moreover, temporal dependence broadly exists in temporal data: across examples such as financial markets, customer behaviors, or system workloads, the state of the world at the present time often depends strongly on the recent past. This observation renders inapplicable much of the existing work on query processing over probabilistic databases [17, 26, 32, 33, 49, 55, 62], where uncertainty in data is assumed to be independent across objects (e.g., attribute or tuple values).

A second challenge stems from the growing popularity of complex predictive models. Oftentimes, due to cost and data privacy considerations, we do not have the luxury of building a custom model directly just to answer one specific durability prediction query; instead, we would be given a general model with which we can simulate possible future states of data and use them to answer various queries. This paper does not prescribe how to come up with such a model; we assume it is given to us and focus on how to use it to answer durability prediction queries efficiently. However, we do want to support a wide gamut of models—be it a traditional stochastic processes with good analytical properties or a black-box model that powers its predictions by deep neural networks. While it is possible to derive analytical answers to durability prediction queries for simple queries and simple models on a case-by-case basis (e.g., when the value of an auto-regressive process [54] hits a particular threshold), our goal is to derive a general-purpose procedure that works for any query and for any model, provided that the model allows us to simulate possible futures step by step. For example, the simulation model can use a recurrent neural network [31, 37, 52] to predict prices for a collection of stocks for the next hour using their prices during the past 36 hours, and the query can ask for the probability that a given stock's P/E ratio will rank among the top 10 by the end of the week. In general, we will need to resort to Monte Carlo (MC) techniques [6] to generate multiple "sample paths" (each corresponding to one possible sequence of future states) and evaluate the query condition on them, in order to derive an estimate to the query answer.

A third challenge, however, stems from the serious inefficiency of the standard MC technique of simple random sampling. It suffers greatly when the answer probability is small. In fact, as previous

*Most of the work was conducted when author was at Duke University.

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457305

studies have pointed out [42, 46], the *relative error* of standard MC increases to infinity as the underlying probability approaches 0, therefore requiring a prohibitively expensive number of simulations in order to achieve acceptable error. Unfortunately, in many practical use cases of durability prediction queries such as the examples above, people are interested in looking for robust and consistent behaviors over time, which naturally leads to small answer probabilities. Hence, the drawbacks of simple random sampling are further amplified by durability prediction queries.

To address these challenges, we propose an alternative approach to answer durability prediction queries that preserves the generality and simplicity of standard MC, but provides significant efficiency improvement. The key insight is that not all sample paths are equally promising. Instead of generating each sample path independently from the very start (as in simple random sampling), we can gauge, from where a path has been so far, how close it is to hitting the condition of interest. We would then "split" a promising partial path into multiple "offsprings," by continuing multiple simulations from this same partial path. With this approach, we effectively direct more simulation efforts towards those more promising simulation paths. Note that we achieve this goal simply by choosing when and where (during sampling) to invoke the given per-step simulation procedure (which can be an arbitrarily complex black-box) without changing its internals, which makes this approach very general and practical.

Our main technical contributions are as follows:

- We formalize the notion of *durability prediction queries* given a predictive model with a step-by-step simulation procedure. The generality of our novel problem formulation and solutions means that they are widely applicable, even for complex models and complex queries that are increasingly common in practice.
- We propose *Multi-Level Splitting Sampling (MLSS)* as a general method for answering durability prediction queries. This method applies the idea of *importance splitting* [25], which has been well-studied in statistic community [10, 42, 58]. However, the original idea has limited applicability because of several strong assumptions on the underlying stochastic process, and it will produce incorrect estimates when applied blindly to our problem. Our approach drops many unrealistic assumptions and is generally applicable on a larger class of processes; it still provides significant efficiency improvement with provably unbiased estimates.
- Practical application of MLSS requires designing "levels" that correspond to "progress milestones" where we split a sample path upon reaching them. We further propose an adaptive greedy strategy that automatically and incrementally searches for a good level design, thereby freeing users from manual tuning. The strategy incurs low overhead and obtains good level design in practice.

2 PRELIMINARIES

2.1 Problem Formulation

Stochastic Process and Simulation Model. Consider a discrete time domain of interest $\mathbb{T} = \{0, 1, 2, 3, \dots\}$ and a discrete-time stochastic process $\{X_t\}_{t\in\mathbb{T}}$ with state space $\mathcal{X}$, where $X_t \in \mathcal{X}$ is a random variable of state at time t. We are given an *initial state* x_0 and a *step-wise simulation procedure* $\mathfrak{g}$ that simulates the process forward step by step: given previous states $x_{<t} = \{x_0, \dots, x_{t-1}\}$ up to time $t-1$, $\mathfrak{g}(x_{<t}, t)$ returns (randomly) the state x_t at time t (x_t is the observed value of X_t). Starting from x_0, we can generate a *sample path* $\{x_0, x_1, x_2, \dots\}$ of arbitrary length for the process by repeatedly invoking $\mathfrak{g}$. Multiple sample paths can be generated simply by restarting the sequence of invocations.

This definition covers a wide range of generative models used in practice for temporal data. It is worth noting that we place no restriction on how complex the state space $\mathcal{X}$ and the step-wise simulation procedure $\mathfrak{g}$ are. We highlight some examples:

(1) *Auto-regressive or AR(m) model*: Here the simulation procedure $\mathfrak{g}$ draws the value v_t at time t according to values of the last m time steps, $\{v_{t-1}, v_{t-2}, \dots, v_{t-m}\}$, by $\sum_{i=1}^{m} \phi_i v_{t-i} + \epsilon_t$, where ϕ_i's are model parameters and ϵ_t's are random errors.

(2) *Time-homogeneous (discrete-time) Markov chains*: Here $\mathfrak{g}$ generates the state at time t according to a given probability distribution $\mathbf{Pr}[X_t \mid X_{t-1}]$ independent of t and conditionally independent of all states prior to the last.

(3) *Black-box models*: The popularity of deep learning in recent years has given rise of highly complex models that are difficult to reason with analytically. Consider, for example, a model that captures the relationship between successive states using a recurrent neural network (RNN). Here, $\mathfrak{g}$ generates the value v_t for time t according to $v_t \sim o(g(h_{t-1}, v_{t-1}; \theta); \theta)$, where h_{t-1} denotes the state of the hidden layer(s) at time $t-1$, $o(\cdot)$ and $g(\cdot)$ are activation functions, and θ denotes (time-invariant) model parameters; $\mathfrak{g}$ further updates h_{t-1} to h_t. The state at time t hence includes both v_t and h_t.

Durability Prediction Queries. Given a stochastic process $\{X_t\}_{t\in\mathbb{T}}$ governed by $\mathfrak{g}$ with initial state X_0, let $q : \mathcal{X} \to \{0, 1\}$ be a user-specified Boolean *query function* that returns 1 if a given state satisfies a condition of interest (and 0 otherwise). A *durability prediction query* (or *durability query* for short) $Q(q, s)$ returns the probability that the process ever reaches any state for which q returns 1 by the end of the prescribed time horizon $s \in \mathbb{T}$. Formally, $Q(q, s) = \mathbf{Pr}[\bigvee_{1\leq t\leq s} q(X_t) = 1]$. Alternatively, consider the time T before the process first "hits" (meets) the condition of interest; T is an random variable, and the durability query $Q(q, s)$ returns $\mathbf{Pr}[T \leq s]$, i.e., the probability that the hitting time is within the prescribed threshold s. As mentioned above, $Q(q, s)$ tends to be small probabilities in real-life applications.

To illustrate, suppose we use a RNN-based model described earlier to predict the price and earning for n stocks. The state at time t consists of h_t (of the hidden layers) as well as a vector $v_t = \langle p_t^{(1)}, e_t^{(1)}, \dots p_t^{(n)}, e_t^{(n)} \rangle$, where $p_t^{(i)}$ and $e_t^{(i)}$ are the price and earning for stock i at time t. For a durability query $Q(q, s)$ concerning the probability that stock i can break into top 10 by time s in terms of P/E ratio, the query function q would access the vector of prices and earnings in the state, compute all P/E ratios, and check if i's rank is within 10. $Q(q, s)$ is the probability that a random sample path reaches a state for which q evaluates to 1 within time s.

For simple models and simple query functions (e.g., the condition of interest is whether an AR(m) process exceeds a given value), we can in fact compute durability prediction queries analytically and exactly. In general, with complex models or complex query functions, computing durability prediction queries becomes exceedingly difficult. Especially when the model itself is complex, we have to resort to Monte Carlo simulations using $\mathfrak{g}$. Let $\tau = Q(q, s)$ denote the exact answer to the query. Instead of returning τ, our goal is to devise an algorithm that can produce an unbiased estimate $\hat{\tau}$ of τ together with some statistical quality guarantee (e.g., confidence interval or estimator variance). We measure the cost of the algorithm by the total number of invocations of $\mathfrak{g}$. In practice, the user can specify a cost budget, and our algorithm will produce a final estimate with quality guarantee when the budget runs out. Alternatively, the user can specify a target level of quality guarantee, and our algorithm will run until the target guarantee is reached. In this paper, we are interested in achieving better guarantees given a fixed budget, or achieving the target guarantee with lower costs.

2.2 Background and Other Approaches

Durability prediction queries are deeply connected to a classic problem in statistics called *first-hitting time* or *first-passage time* in stochastic system [16, 50, 60]. Similar problems related to first-hitting time are also independently studied in very diverse fields, from economics [53] to ecology [22]. We briefly introduce several existing approaches for durability prediction queries (or the first-hitting time problem) here, and lay the foundation for later sections.

Analytical Solution. As mentioned earlier, there exist analytical solutions for some simple stochastic processes [29], e.g., Random Walks, AR(m) model, to name but a few. However, real applications often require more complex structures. For instance, Compound-Poisson process is a well-known stochastic model for risk theory in financial worlds. In [61], authors derived an analytical solution for such stochastic processes. However, the exact solution itself is very complicated, involving multiple integrals that still require numerical approximations. In general, the analytical solution to first-hitting problem is model-specific, may not exist for most applications, and hence cannot be directly used for durability query processing.

Simple Random Sampling (SRS). Monte Carlo simulations is the most general approach for answering durability prediction queries. SRS is the standard Monte Carlo technique. To answer durability query $Q(q, s)$ with query function q and prescribed time threshold s, we randomly simulate n independent sample paths according to the procedure $\mathfrak{g}$. For each sample path $SP_i = \{x_0, x_1, \dots\}$, we define a label function indicating whether the simulated path satisfies the query condition:

$$l(SP_i) = \begin{cases} 1, & \bigvee_{1 \le t \le s} q(x_t) = 1, \\ 0, & \text{otherwise.} \end{cases}$$

Then, an unbiased estimator of SRS is $\hat{\tau}_{srs} = \frac{\sum_{i=1}^{n} l(SP_i)}{n}$, with estimated variance $\widehat{\mathrm{Var}}(\hat{\tau}_{srs}) = \frac{\hat{\tau}_{srs}(1-\hat{\tau}_{srs})}{n}$.

We use SRS as the main baseline solution throughout the paper. The major drawback of SRS is its "blind search" nature—it randomly simulates sample paths and just hopes that they could reach the target. For durability prediction queries with small ground truth answer τ, SRS would waste significantly much simulation effort on those sample paths that do not ever satisfy the query condition.

Importance Sampling (IS). Importance sampling is one of the most popular variance reduction techniques for Monte Carlo simulations. It is a special case of biased sampling, where sampling distribution systematically differs from the underlying distribution in order to obtain more precise estimate using fewer samples. Let us use the following concrete example for illustration. Consider an AR(1) model, where simulation procedure $\mathfrak{g}$ draws the value v_t according to $\phi_1 v_{t-1} + \epsilon_t$. Here, ϕ_1 is a constant parameter and ϵ_t is independent Gaussian noise; i.e., $\epsilon_t \sim N(0, \sigma)$ for $t \in \mathbb{T}$. Given time threshold s, the random variable of interest $l(SP)$ has probability density $g(l) \sim \prod_{i=1}^{s} N(0, \sigma)$. IS draws samples from an instrumental distribution ω, and an unbiased estimator is $\hat{\tau}_{is} = \frac{1}{n} \sum_{i=1}^{n} \frac{g(l(SP_i))}{\omega(l(SP_i))} l(SP_i)$. Choosing a good instrumental distribution ω is critical for the success of IS. An iterative approach called *Cross-Entropy (CE)* [18, 51] is widely used for importance sampling optimization. However, IS typically requires a priori knowledge about the model, e.g., model parameters or state transition probabilities. This requirement can be impractical for some complex temporal processes, not to mention black-box models that we consider in this paper.

Learning Durability Directly. Instead of answering durability prediction queries from using a simulation model, one could also learn predictive models that *directly* answer durability prediction queries. However, a general simulation model as we considered in this paper is often preferred based on the following considerations: (1) In some cases, we do not always have the luxury of training custom models from real data points just to answer queries. It is costly or even unethical to collect training data for such purposes, e.g., autonomous driving car testing. (2) Building an one-off model to answer durability prediction queries (with different query conditions and different parameters) would quickly become infeasible because each type of durability prediction implies a different modeling exercise. In contrast, a general simulation model can be conveniently reused for answering a variety of queries using our technique without requiring extra data collections or modeling expertise. Moreover, a nice byproduct of utilizing simulation models is that we also produce a set of concrete sample paths alongside the point estimate and confidence interval. Users can look into these "possible worlds" to get a better understanding of query answers. Compared to the relatively opaque direct models that output only the final durability prediction, this approach provides more interpretability and credibility. We acknowledge the difficulty of having good simulation functions, but we note that for many domains, e.g., finances, autonomous vehicles, etc., the use of such general simulation models are well-established and common.

Remarks. All solutions reviewed above have limitations: analytical solutions and IS are not generally applicable to durability prediction queries; SRS can be very inefficient; and the alternative of learning custom models to predict durability directly may be infeasible for practical (ethical or cost) reasons. In the ensuing sections, we introduce a novel approach for answering durability prediction queries from simulation models that achieves the generality of SRS as well as the efficiency of IS.

Table 1: Notations

s	Prescribed time horizon of the durability query.
r	Splitting ratio (or branching factor).
m	Number of levels.
β_i/β	The partition boundary for the i-th level / target value.
L_i	The i-th level. L_m is the target level.
N_i	Number of first-time entrance state into L_i. N_0 is the number of root paths; N_m is the number of hits to the target.
p_i	(conditional) Level advancement probability from level L_{i-1} to L_i.

3 SIMPLE MLSS

Since generating too many paths that do not meet the query condition can be a waste of simulation cost, it is natural to design a sampling procedure that more frequently produces paths that reach the target. To this end we apply the idea of *splitting* [25]. The intuition is to encourage further explorations of paths that are more likely to hit the condition of interest by splitting them into multiple offsprings when they reach particular "milestones" (see Figure 1 for illustration). Such treatment is analogous to the notion of sample weight/importance in importance sampling, but without explicitly tweaking the underlying step-wise simulation procedure $\mathfrak{g}$.

Traditional applications of the splitting idea are mostly concerned with much simpler settings (e.g., process with strong Markov properties). To apply splitting to our setting of durability prediction queries, we need to introduce the concept of *value functions*.

Value Functions. We first capture how promising a path prefix is using a heuristic *value function* $f(x_t) : \mathcal{X} \times \mathbb{T} \to (0, 1]$. The closer $f(x_t)$ is to 1, the more likely that the process will reach the query condition given the current state. We further require that $f(x_t) = 1$ if and only if $q(x_t) = 1$.

As outlined in the definition of *levels* below, f guides when to split a path, thus a properly defined f leads to more efficient simulation efforts. It is worth noting that the unbiasedness of our estimator in this section does *not* depend on f; only its efficiency does.

The best choice of f is problem-specific as it depends on both the simulation model and the query. In practical applications, the query condition often takes the form $z(x_t) \geq \beta$, where $z : \mathcal{X} \to \mathbb{R}$ is a real-valued evaluation of a state and β is a user-specified value threshold, and $z(x_t)$ has a higher chance to hit the boundary when x_t is closer to it. Thus a reasonable value function would be $f(x_t) = \min\{z(x_t)/\beta, 1\}$. More sophisticated designs of value functions are certainly possible, but are beyond the scope of this paper.

Levels. With the help of the value function f, we can now introduce the notion of *levels* to capture multiple intermediate "milestones" a sample path can reach before meeting the query condition. We partition $[0, 1]$, the range of value function f, into $m+1$ disjoint *levels* (intervals) with boundaries $0 = \beta_0 < \beta_1 < \cdots < \beta_m = 1$, where $L_i = [\beta_i, \beta_{i+1})$ for $0 \leq i \leq m-1$ are the first m levels, and the degenerated interval $L_m = [1, 1]$ is the last level. Let $T_i(SP) = \inf\{t \geq 0 \mid f(x_t) \in L_i\}$ be the first time that a sample path $SP : \{x_t\}_{t\geq 1}$ enters level L_i. Let $\Xi_i = \{SP \mid T_i(SP) \leq s\}$ denote the event that the process enters L_i before time threshold s; i.e., the set of all possible simulated paths that enter L_i before s.

3.1 s-MLSS Sampler and Estimator

We are now ready to describe a *simple* version of *Multi-Level Splitting Sampling*, *s-MLSS*. Frequently used notations are summarized in Table 1. As noted earlier, the idea of *splitting* can be traced back to 1951 [38] and has been used by several authors in statistic community [25]. The interested readers can refer to [25] for a more comprehensive introduction. However, prior studies mostly focused on stochastic processes with strong Markov property. Here we introduce s-MLSS for more general simulation models in the context of durability prediction queries. Nonetheless, s-MLSS still inherits a critical assumption from existing literature:

(No Level-Skipping) Any sample path generated from the stochastic procedure $\mathfrak{g}$ has to enter L_i before it enters L_{i+1}, for every $i \leq m$.

This assumption is rather restrictive and does not hold in general. It is possible (especially in practice with discrete time) for a sample path's value to jump, between two consecutive time instants, from a level to a higher one, crossing multiple levels in between. We show with experiments in Section 6 that ignoring this assumption and blindly applying s-MLSS will in practice lead to incorrect answers. In Section 4, we instead see how *g-MLSS*, our general version of MLSS, lifts this assumption and correctly handles the general case. Nonetheless, s-MLSS serves as a good starting point for our exposition.

As a result of the no-level skipping assumption, we have the following containment relation:

$$\Xi_m \subset \Xi_{m-1} \subset \cdots \subset \Xi_1 \subset \Xi_0. \quad (1)$$

With (1) and the chain rule for probability we decompose the target probability τ as

$$\tau = \mathbf{Pr}[\Xi_m] = \mathbf{Pr}[\Xi_m \mid \Xi_{m-1}] \cdots \mathbf{Pr}[\Xi_1 \mid \Xi_0]\, \mathbf{Pr}[\Xi_0] = \textstyle\prod_{i=1}^{m} p_i\ , \quad (2)$$

where $p_i = \mathbf{Pr}[\Xi_i \mid \Xi_{i-1}]$ is referred to as the *level advancement probability*. Next we show the s-MLSS sampling approach that estimates p_i's and in turn τ.

s-MLSS Sampler. In a nutshell, s-MLSS works in rounds of stages, estimating the decomposed probability p_i separately between consecutive levels. For each level L_i, we maintain a counter N_i denoting the number of sample paths that enter L_i for the first time. In the first stage, we start the simulation of a path from the initial level L_0, which we refer to as the *root path*, and increment the counter N_0 by 1. We continue the simulation up to time s:

(1) If the sample path does not enter the next level L_1, we stop and start a new round of simulation for the next root path.
(2) Otherwise, we increment the counter N_1 by 1 and split the root path into r independent copies at the first time it enters L_1, where r is a constant called *splitting ratio*. Assume the hitting time is t, we define the state $\mathcal{X}_t$ of sample path as the *entrance state* to L_1. All splitting copies from the original path will use the same entrance state $\mathcal{X}_t$ as starting point for future simulations.

Then in the next stage, for each of the splitting offspring of the root path, we recursively follow the similar procedure as described above: simulate the path up to time s; if it reaches the next level, increment the counter of that level, split and repeat; If not, finish the simulation at time s. The simulation of a root path stops when

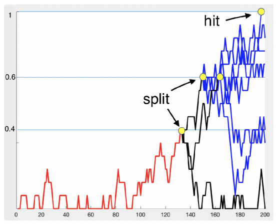


Figure 1: Simulations of a root path using MLSS with splitting ratio $r = 3$. Horizontal axis is time, with horizon $s = 200$; vertical axis shows the result of the value function.

we finish the simulations of all its splitting offspring—either enters the target level L_m or runs until the time s.

Figure 1 illustrates a concrete example of the simulations of one root path. Here we have $s = 200$, levels $L_0 = [0, 0.4), L_1 = [0.4, 0.67), L_2 = [0.67, 1), L_3 = [1, 1]$, and splitting ratio $r = 3$. The root path (red line) starts from L_0 and enters L_1 at timestamp 133. Then it splits into 3 copies (black lines) and continues the simulations forward. Two out of the three splitting paths (from L_1) enter L_2 and each of them further splits into three more copies (blue lines), respectively. Finally, one out of the six splits (from L_2) enters the target level L_3. All other copies (that do not have the chance to split) run till time s and stop. Following the above procedure, assume we sample and simulate N_0 root paths until the stopping criteria is met (i.e., the simulation budget runs out or the estimate achieves the target quality guarantee).

s-MLSS Estimator. Using the counters we have maintained through MLSS for each level, we have $\hat{p}_1 = \frac{N_1}{N_0}, \hat{p}_2 = \frac{N_2}{rN_1}, \ldots, \hat{p}_m = \frac{N_m}{rN_{m-1}}$. The estimator for MLSS is

$$\hat{\tau}_{mlss} = \prod_{i=1}^{m} \hat{p}_i = \frac{N_1}{N_0}\frac{N_2}{rN_1}\cdots\frac{N_m}{rN_{m-1}} = \frac{N_m}{N_0 r^{m-1}}. \tag{3}$$

It can be shown that $\hat{\tau}_{mlss}$ is an unbiased estimator of τ. Intuitively, s-MLSS generates a forest of N_0 r-ary trees of sample paths with depth m: the root is the initial state, nodes are the states at which we split, and edges are simulated sample paths. The total number of leaf nodes is at most $N_0 r^{m-1}$, and we count the total number of them, N_m, reaching the target. In turn, it gives us the estimator in form $N_m/N_0 r^{m-1}$.

Proposition 1. *Under the "no level-skipping" assumption (1), using the MLSS with m levels and a splitting ratio r, $\hat{\tau}_{mlss}$ is an unbiased estimator of τ; that is, $\mathbb{E}[\hat{\tau}_{mlss}] = \tau$.*

Variance Analysis. Assume that we have sampled and simulated N_0 independent root paths. The variance of our estimator (using standard variance estimator) is

$$\widehat{\mathrm{Var}}(\hat{\tau}_{mlss}) = \frac{\sum_{i=1}^{N_0}(N_m^{\langle i\rangle} - \overline{N_m})^2}{N_0(N_0-1)r^{2(m-1)}}, \tag{4}$$

where $N_m^{\langle i\rangle}$ denotes the number of hits to the final target contributed by the root path i, and $\overline{N_m}$ is the sample mean of target hits from simulations of N_0 root paths. Please refer to the extended version of this paper [24] for full derivations.

Relationship between SRS and MLSS. It is not hard to prove that SRS is a special case of MLSS with splitting ratio $r = 1$. As $r = 1$,

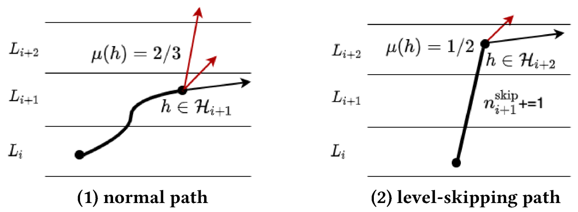


Figure 2: Different types of simulated partial paths.

$\hat{\tau}_{mlss} = N_m/N_0 = \hat{\tau}_{srs}$. Similarly, $\mathrm{Var}(\hat{\tau}_{mlss}) = \mathrm{Var}(N_m^{\langle 1\rangle})/N_0 = \tau_{srs}(1-\tau_{srs})/N_0$, that degenerates to $\mathrm{Var}(\hat{\tau}_{srs})$.

However, we still need careful considerations to apply MLSS in practice for the best performance; e.g., how to select splitting ratio r, how many partitions of levels do we need and how to decide the boundaries of partitions. There are many trade-offs among those choices. We discuss how to solve for the optimal setting of MLSS that minimizes the simulation cost in Section 5.

4 GENERAL MLSS

The last section introduced the s-MLSS estimator under the "no level-skipping" assumption. In general, this assumption can be easily violated, e.g., with volatile stochastic processes such as stock prices. To remove this assumption and make MLSS more widely applicable, we propose a novel and general MLSS procedure.

Without the "no level-skipping" assumption, (1) and (2) no longer hold and need to be modified. With the same sequence of boundaries $0 = \beta_0 < \beta_1 < \cdots < \beta_m = 1$, we denote by $U_i(SP) = \inf\{t \geq 0 \mid f(x_t) \geq \beta_i\}$ the first time that a sample path $SP : \{x_t\}_{t\geq 1}$ *crosses* boundary β_i. Notice the difference here between U_i and T_i (in Section 3.1) that T_i specifically requires the process to land inside L_i while U_i only requires the process to pass the lower boundary of L_i. Denote by $\Theta_i = \{SP \mid U_i(SP) \leq s\}$ the event that the process crossed boundary β_i before s. Similarly, we have

$$\Theta_m \subset \Theta_{m-1} \subset \cdots \subset \Theta_1 \subset \Theta_0, \tag{5}$$

and subsequently

$$\tau = \mathbf{Pr}[\Theta_m] = \mathbf{Pr}[\Theta_m \mid \Theta_{m-1}]\cdots\mathbf{Pr}[\Theta_1 \mid \Theta_0]\,\mathbf{Pr}[\Theta_0] = \textstyle\prod_{i=1}^{m}\pi_i \tag{6}$$

where $\pi_i = \mathbf{Pr}[\Theta_i \mid \Theta_{i-1}]$. The above probability decomposition is general and carries no assumption. Next we outline the g-MLSS sampling procedure that unbiasedly estimates π_i.

4.1 g-MLSS Sampler and Estimator

The g-MLSS sampler starts simulations from root paths, and recursively split the sample path whenever it *lands* in *any* level for the first time until time runs out or the path satisfies the query condition. Whenever a splitting happens, we record the proportion of offspring processes that cross the higher boundary of the level in which the splitting happens.

Formally, in a realized g-MLSS simulation, denote by $\mathcal{H}_i \subset \mathcal{X}\times\mathbb{T}$ the set of splitting states in L_i, each of which belongs to a separate path that lands in L_i before s. On the other hand, denote by n_i^{skip} the number of paths that pass β_{i+1} *without* landing in L_i, where level-skipping happens (see Figure 2). For any $h \in \mathcal{H}_i$, let $\mu(h)$ denote the ratio of h's offsprings that cross β_{i+1}. The estimator of

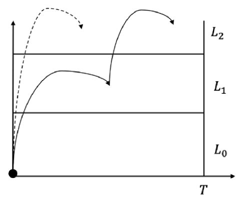


Figure 3: A simple two-level case with level-skipping. Dashed path represents a (discrete-time) series that directly goes from L_0 to L_2 skipping L_1.

$\pi_{i+1}, i > 0$ is given by

$$\hat{\pi}_{i+1} = \frac{1}{|\mathcal{H}_i| + n_i^{\text{skip}}} \Big(\sum_{h \in \mathcal{H}_i} \mu(h) + n_i^{\text{skip}} \Big). \tag{7}$$

It is worth noting that, with g-MLSS, there is no need to use a unified splitting ratio r as in s-MLSS. After all, when a path splits at h, we only need the ratio $\mu(h)$ instead of the size of its offsprings.

The special case is the starting level L_0, since we directly start with N_0 independent root paths. The estimation of π_1 is given by $\hat{\pi}_1 = \frac{|\mathcal{H}_1| + n_1^{\text{skip}}}{N_0}$. Following the probability decomposition by (6), the general MLSS estimator for τ is

$$\hat{\tau}_{mlss} = \prod_{i=1}^{m} \hat{\pi}_i. \tag{8}$$

Proposition 2. *In general, using the Multi-Level Splitting Sampling with m levels, $\hat{\tau}_{mlss}$ in (8) is an unbiased estimator of τ; that is, $\mathbb{E}[\hat{\tau}_{mlss}] = \tau$.*

Given the general form of MLSS as above, it is more clear how s-MLSS is a special case of the general one. With the "no level-skipping" assumption, n_i^{skip}'s are always zero. Given a unified splitting ratio r, for any splitting state $h \in \mathcal{H}_i$ in L_i, $\mu(h) = N_{i+1}(h)/r$, where $N_{i+1}(h)$ denotes the number of hits (from h's split offsprings) to hit the next level L_{i+1}. Additionally, $|\mathcal{H}_i| = N_i$ (recall that N_i is the number of entrances to L_i). Hence, (7) degenerates to $\hat{\pi}_{i+1} = \frac{1}{N_i} \frac{\sum_{h \in \mathcal{H}_i} N_{i+1}(h)}{r} = \frac{N_{i+1}}{rN_i}$, and the g-MLSS estimator by (8) is equivalent to the s-MLSS estimator by (3).

In summary, g-MLSS greatly extends the applicability of s-MLSS by allowing level-skipping and a dynamic splitting ratio. With the g-MLSS algorithm and estimator, we are able to efficiently obtain quality estimations of durability prediction queries on mostly any temporal process that exhibits continuity and temporal dependence. Moreover, the flexible splitting procedure opens up many interesting opportunities for optimization, e.g., how to optimally allocate splitting ratios across sample paths, or how to learn and converge to the optimal assignment on-the-fly while conducting MLSS.

4.2 Variance Analysis

In order to practically apply g-MLSS, we need its variance term to determine the stopping condition (i.e., confidence interval or relative error) for durability query processing. The variance of general MLSS estimator as in (8) would be very complicated and challenging, because the underlying stochastic process takes general forms. Unfortunately, we do not have a closed-form expression of the variance for the general case in this paper. With that being said, it will not limit the utility of g-MLSS in practice. In this section, we first showcase the variance analysis of general MLSS estimator for a simple but non-trivial case—two levels with level-skipping (Figure 3). Then, we show how to use *bootstrapping* [20] to provide a variance estimate of g-MLSS for the general case in practice.

Simple Two-level Level-skipping. There are two types of paths hitting the target: solid line path (with no level-skipping) and dashed line path (directly jump from L_0 to L_2). With abuse of notation, let $p_{0,1} = p_1$ and $p_{1,2} = p_2$ (recall (1) and (2)). The numbers in subscript simply represent the transition between levels. These probabilities represent the normal case as we discussed in Section 3. However, with the existence of level-skipping, we need to introduce an additional probability $p_{0,2}$ denoting the chance of level-skipping. Hence, the ground truth hitting probability consists of two parts: $\tau = p_{0,1}p_{1,2} + p_{0,2}$. Accordingly, we decompose counter N_2 (number of hits to the target) as $N_2 = N_2^{(ns)} + N_2^{(s)}$, where $N_2^{(ns)}$ is the number of hits from non-skipping paths while $N_2^{(s)}$ is the number of hits from level-skipping paths. Then, our estimator also consists of the estimations of these two parts, $\hat{\tau}_{mlss} = N_2^{(ns)}/N_0 r + N_2^{(s)}/N_0$. For variance, we have $Var(\hat{\tau}_{mlss}) = \text{Var}(N_2^{(ns)})/N_0^2 r^2 + \text{Var}(N_2^{(s)})/N_0^2$. First, $N_2^{(s)}$ can be viewed as a binomial variable with N_0 trials and probability $p_{0,2}$; i.e., $N_2^{(s)} \sim B(N_0, p_{0,2})$. Thus $\text{Var}(N_2^{(s)}) = N_0 p_{0,2}(1 - p_{0,2})$. Second, the quantity $N_2^{(ns)}$ conditions on the number of paths without skipping (N_1), which it is also an random variable throughout the sampling procedure. We cannot just break it up as in standard variance analysis. Instead, we should do a conditioning on number of non-skipping paths and use the law of total variance:

$$\begin{aligned}
\text{Var}(N_2^{(ns)}) &= \text{Var}\Big(\mathbb{E}[N_2^{(ns)} \mid N_1]\Big) + \mathbb{E}\Big[\text{Var}(N_2^{(ns)} \mid N_1)\Big] \\
&= \text{Var}(N_1 r p_{1,2}) + \mathbb{E}[N_1 \text{Var}(N_2^{\langle 1 \rangle})] \\
&\quad (\text{Var}(N_2^{\langle i \rangle}) = \text{Var}(N_2^{\langle j \rangle}) = \text{Var}(N_2^{\langle 1 \rangle}), \quad \forall i, j < N_0) \\
&= r^2 p_{1,2}^2 \text{Var}(N_1) + \mathbb{E}[N_1]\text{Var}(N_2^{\langle 1 \rangle}) \\
&= r^2 p_{1,2}^2 N_0 p_{0,1}(1 - p_{0,1}) + \mathbb{E}[N_1]\text{Var}(N_2^{\langle 1 \rangle}). \\
&\quad\quad (N_1 \sim B(N_0, p_{0,1}))
\end{aligned}$$

Hence,

$$\frac{\text{Var}\big(N_2^{(ns)}\big)}{N_0^2 r^2} = p_{1,2}^2 \frac{p_{0,1}(1-p_{0,1})}{N_0} + p_{0,1} \frac{\text{Var}(N_2^{\langle 1 \rangle})}{N_0 r^2}.$$

Putting it all together, we have

$$\text{Var}(\hat{\tau}_{mlss}) = p_{1,2}^2 \frac{p_{0,1}(1-p_{0,1})}{N_0} + p_{0,1} \frac{\text{Var}(N_2^{\langle 1 \rangle})}{N_0 r^2} + \frac{p_{0,2}(1-p_{0,2})}{N_0}. \tag{9}$$

In practice, we can use $\hat{p}_{0,1} = N_1/N_0$ as an unbiased estimation for $p_{0,1}$. Similarly, $\hat{p}_{0,2} = N_2^{(s)}/N_0$ and $\hat{p}_{1,2} = N_2^{(ns)}/N_1 r$ as unbiased estimations for $p_{0,2}$ and $p_{1,2}$, respectively. $\text{Var}(N_2^{\langle 1 \rangle})$ can be estimated unbiasedly similar to (4) by reusing the simulated root paths. Again, it is not hard to find that the variance term we derived in (4) is a special case of the above equation when there is no level-skipping; i.e., $p_{0,2} = 0$ and $p_{0,1} = 1$.

We believe the variance analysis of g-MLSS estimator, though very complex, would follow the similar procedure as in the simple two-level case. We leave this part as one of the future work.

General Level-skipping and Bootstrapping Evaluation. In the general case, the standard technique of bootstrap sampling can be used in practice to provide a good estimation for variance of g-MLSS estimator. This approach is widely used to empirically estimate the variance or the distribution of sample mean when the population variance is complex and or not accessible.

More specifically, in our setting, assume we have already simulated N_0 root paths and obtained an estimate $\hat{\tau}_0$ of the hitting probability. In one bootstrap run, we randomly draw n root paths, *with replacement*, from the existing root paths, from which we calculate a g-MLSS estimate, called a bootstrap estimate. We perform N such independent bootstrap runs to obtain N bootstrap estimates $\hat{\tau}_i, i = 1, ..., N$. From the empirical distribution of $\hat{\tau}_i, i = 1, ..., N$, we calculate the (bootstrapped) variance for g-MLSS, i.e., $\widehat{Var}(\hat{\tau}_0) = \sum_{i=1}^{N}(\hat{\tau}_i - \bar{\tau})^2/N$, where $\bar{\tau}$ is the mean of bootstrap estimates from N bootstrap runs.

Despite the simplicity and effectiveness of bootstrap sampling, it may incur considerable evaluation cost, as bootstrapping essentially replays the history multiple times, and may become increasingly expensive as we expand the sample pool. Compared to the case of s-MLSS or g-MLSS with the two-level setting where variance can be directly calculated, applying g-MLSS with bootstrap evaluation requires more care to achieve good overall performance. There are several techniques for speeding up bootstrap sampling, ranging from more advanced subsampling procedures [4, 40] to parallel computation. A practical rule of thumb that we found in practice is to run bootstrap evaluation conservatively—compared with frequent bootstrapping to ensure that we never overshoot the given quality target, sometimes overrunning the simulation a little would be overall more efficient. As we will see in Section 6, applying this rule, even with an unoptimized bootstraping implementation, g-MLSS can still provide up to 5x overall speedup over SRS.

5 OPTIMIZING MLSS DESIGN

There is still one missing piece in applying MLSS: how do we choose its parameters? Specifically, how many levels do we need, and given the number of levels, how do we properly partition the value function range into levels? In Section 4, we saw that g-MLSS also allows variable splitting ratios—how do we additionally choose these? Manually tuning all these parameters is clearly impractical. On the other hand, automatic optimization is also challenging because of the vast space of possibilities as well as the difficulty of not knowing the effectiveness of our choices a priori.

We make some simplifying assumptions to make the optimization problem tractable. 1) We focus only on choosing level partition plans; we forgo the freedom of setting variable splitting ratios and instead choose a small, fixed splitting ratio r. As validated in experiments in Section 6, large ratios tend to be suboptimal because they dramatically increase the number of paths at higher levels. Furthermore, variable splitting ratios can be effectively approximated by replacing a level having a large ratio with multiple levels, each having the same small fixed ratio. 2) We derive an empirical measure for evaluating different MLSS parameter settings (Section 5.1). For ease of derivation and measurement, we make the same no level-skipping assumption as in s-MLSS. While this assumption does not hold in general, it allows to obtain a surrogate measure that is much cheaper to estimate. Importantly, it does not affect the correctness of our sampling and estimation procedures in any way because it is only used to choose a partition plan. Moreover, as we will see in experiments in Section 6.3, this measure work well in practice for both s-MLSS and g-MLSS on various models and query types. 3) We then present an adaptive greedy strategy (Section 5.2) that searches for the parameters settings aimed at optimizing the above empirical metric. This strategy is generic, applicable to both s-MLSS and g-MLSS, and can work with other, better empirical measures if available.

5.1 Partition Plan Evaluation

Previous work in statistics [42] showed an analogy between MLSS (with fixed splitting ratio) and branching process theory [30], and concluded that the optimal setting for MLSS is to make advancement probabilities between consecutive levels roughly the same, called a "balanced growth." That is, consider MLSS with m levels,

$$p_1 = p_2 = \cdots = p_m = p = \tau^{1/m}. \tag{10}$$

From standard branching process theory, we have

$$\mathrm{Var}(\hat{\tau}_{mlss}) = \frac{m(1-p)p^{2m-1}}{N_0}. \tag{11}$$

The above expression indicates that given a fixed number of root paths, more levels lead to smaller variance. However, more levels also lead to more expensive simulation cost of a root path because of the exponential splitting growth of a root path through the levels. Our optimization goal, using MLSS as approximate query processing technique, is not just to minimize variance. Instead, we hope to minimize the variance in a fixed amount of the time. Ultimately, the query time using MLSS is determined by the variance of the estimator. A smaller variance in unit time directly leads to less simulation cost for answering durability query. Though the "balanced growth" strategy is a reasonable guideline to partition the space, it is still not clear, in practice, how to partition levels that create balanced growth and how to choose the right number of levels.

To meet our needs, we propose the following evaluation metric. Consider a level partition plan B, which consists of a set of values what we call "partition boundaries"; that is, $B = \{v \mid v \in (0, 1)\}$. Given a fixed amount of simulation budget, say t_0 time, we have simulated $N(t_0)$ root paths (including all its splitting copies). Note that $N(t_0)$ is a random variable depending on the total time t_0 and the average simulation time c_B of a root path using partition plan B. We define an evaluation function for B in terms of variance of estimator $\hat{\tau}_{mlss}$ by $N(t_0)$:

$$eval(B) = \mathrm{Var}\left(\frac{N_m(t_0)}{N(t_0)r^{m-1}}\right), \tag{12}$$

where $N_m(t_0)$ is a random variable denoting the total number of target hits within t_0 time. In this case, $m = |B| + 1$, denoting the total number levels induced by plan B. Since $N(t_0)$ is a random variable, we should express the variance term conditioning on $N(t_0)$, and use the law of total variance and the decomposition trick in (4):

$$eval(B) = \mathbb{E}\left[\frac{\mathrm{Var}(N_m^{(1)})}{N(t_0)r^{2(m-1)}} \mid N(t_0)\right] + \mathrm{Var}\left(\mathbb{E}\left[\frac{N_m(t_0)}{N(t_0)r^{m-1}} \mid N(t_0)\right]\right).$$

Algorithm 1: Adaptive Greedy Partition.

```
Input  : Interval $I = [0, 1]$.
Output : A partition plan $B = \{v \mid v \in (0, 1)\}$.
1  $B \leftarrow \emptyset$;
2  $opt\_eval \leftarrow$ INT_MAX;            // remember the minimum so far
3  $v_{lo} \leftarrow 0, v_{hi} \leftarrow 1$;
4  for round $i = \{1, 2, \cdots\}$ do
5      Uniformly generate a value set $C = \{v \mid v \in (v_{lo}, v_{hi})\}$ as
         candidates for the $i$-th partition boundary;
6      $e^* = \min_{v \in C} eval(B \cup v)$;
7      $v^* = \arg\min_{v \in C} eval(B \cup v)$;
8      if $e^* < opt\_eval$ then
9          $B \leftarrow B \cup v^*$;
10         $opt\_eval \leftarrow e^*$;
11         Find the level $[\beta_i, \beta_j] (\beta_i, \beta_j \in B, \beta_i < \beta_j)$, induced by $B$
             and $I$, that has the smallest level advancement probability
             $p_{i,j}$;
12         $v_{lo} \leftarrow \beta_i, v_{hi} \leftarrow \beta_j$;
13     else
14         break;
15 return $B$;
```

Given $N(t_0)$, $\mathbb{E}\left[\frac{N_m(t_0)}{N(t_0)r^{m-1}}\right] = \tau$, thus the second term in the above equation is 0. Recall that $N_m^{\langle 1 \rangle}$ is a random variable denoting the number of target hits from a root path. Given $N(t_0)$ and partition plan B, $\mathrm{Var}(N_m^{\langle 1 \rangle})$ becomes a constant. Hence, the first term in the equation becomes $\frac{\mathrm{Var}(N_m^{\langle 1 \rangle})}{r^{2(m-1)}}\mathbb{E}[1/N(t_0)]$. The term $\mathbb{E}[1/N(t_0)]$ can be roughly estimated by $1/\frac{t_0}{c_B} = c_B/t_0$. Finally, the evaluation function of a partition plan B is

$$eval(B) = \frac{\mathrm{Var}(N_m^{\langle 1 \rangle})}{r^{2(m-1)}} \frac{c_B}{t_0}. \tag{13}$$

Ideally, given a fixed amount of time t_0, we hope to solve for partition plan B that minimizes the objective $eval(B)$. However, this optimization problem is hard to solve analytically, since $\mathrm{Var}(N_m^{\langle 1 \rangle})$ and c_B are themselves variables when the plan B changes. But fortunately, we can optimize the objective empirically, as $\mathrm{Var}(N_m^{\langle 1 \rangle})$ and c_B can be estimated through the MLSS simulations. $\mathrm{Var}(N_m^{\langle 1 \rangle})$ can be estimated using variances of target hits from all simulated root paths, and c_B can also be estimated simply dividing t_0 by the number of simulations of root path within time t_0. In this way, we can start with a candidate pool of partition plans. For each candidate, we run MLSS simulations for the same amount of time t_0 as trial runs to estimate the objective $eval(B)$, and finally pick the best candidate that produces the minimum value.

5.2 An Adaptive Greedy Partition Strategy

To empirically optimize MLSS, it is prohibitively expensive to run trial simulations for all feasible partition plans and splitting ratios. In this section, we present a heuristic greedy strategy that works well in practice to automatically search for (near-) optimal MLSS parameters.

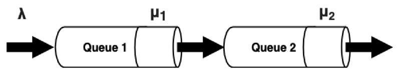


Figure 4: Tandem Queue with Poisson arrivals and Exponential service time.

The main idea of our strategy is to adaptively and recursively partition the space—place the partition boundaries one by one and always partition the level with smaller level advancement probability. The intuition behind our greedy behavior is two-fold: (1) A level with smaller level advancement probability means that this level is an "obstacle" blocking sample paths reaching the target. Partition such levels would focus the simulation resources more on success paths; (2) as we recursively bisect levels with smaller advancement probability, it automatically moves towards a "balanced growth" situation where advancement probabilities from all levels are roughly the same. Recall our discussion in Section 5.1; this behavior has already been confirmed by [42] to have a better sampling efficiency.

Full description is shown in Algorithm 1. Throughout the procedure, we adaptively make two decisions: the optimal number of levels, and the placement of these levels. At the beginning, we start with the original interval $[0, 1]$ (Line 1). Then we place the partition boundary one by one, recursively bisecting the value intervals, until a stopping condition is met (Line 4-14). In the loop, we first generate a set of candidate boundaries (Line 5) and then use the empirical evaluation approach, as elaborated in Section 5.1, to find the optimal partition boundary (Line 6 and Line 7). Finally, we need to update our partition plan and decide when to stop the procedure. If the current best evaluation is better the previous, we continue to add a new partition boundary (Line 8-12). Note that here we need to greedily pick the next interval with smallest advancement probability to partition (Line 11-12). Otherwise, if the current best evaluation is already worse than the previous, there is no need to further add more partition boundaries to the plan, since more levels lead to exponential growth of splitting paths and would incur more expensive simulation cost overall.

The aforementioned empirical optimization framework saves users from the time-consuming manual parameters tuning process when applying MLSS in practice. We set a reasonable fixed splitting ratio (which we further justify in Section 6.3) in advance, and our optimization framework will take care of the rest. An additional benefit of our empirical optimization solution is that all trial runs of MLSS are not "wasted." Since each trial simulation, no matter which plan it follows, returns an unbiased estimator. So in process of picking the optimal parameters, we also are building up towards a reliable estimation for the query.

6 EXPERIMENTS

We select three stochastic temporal processes with simulation models that are commonly used in practical applications.

(1) Tandem Queues: As shown in Figure 4, we have a queueing system with tandem queues, which is the simplest non-trivial network of queues in *queueing theory* [15]. The process is the following. Customers come into Queue 1 following a Poisson distribution with $Pois(\lambda)$. Queue 1 services each customer following an Exponential distribution with $Exp(\mu_1)$, and then sends customers into Queue 2. Queue 2 services each customer by another Exponential distribution with $Exp(\mu_2)$ before customers leave the system.

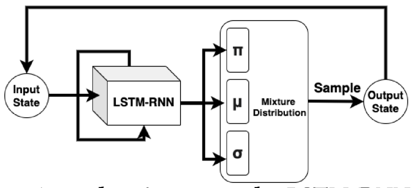


Figure 5: A stochastic process by LSTM-RNN-MDN.

We consider the number of customers in Queue 2 as a stochastic process, and always start with an empty system (i.e., two empty queues). In our experiments, we set $\lambda = 0.5, \mu_1 = \mu_2 = 2$. Queuing system is the foundation for many real-world problems [45], e.g., birth-death process, supply chains, transportation scheduling, and computer networks analysis [41]. Durability queries on such models are widely used to evaluate the robustness of systems.

(2) Compound-Poisson Process: A Compound-Poisson Process (CPP) can be described by the following stochastic process $U = \{U(t)\}_{t\geq 0}$. $U(t) = u + ct - S(t)$, where $S(t)$ is a compound Poisson process with jump density λ and jump distribution F, and $u, c > 0$ are constants. This type of process is commonly used in the financial world for risk management and financial product design [2]. Intuitively, imagine a insurance policy with u as initial surplus and c as users' monthly payment. The compound Poisson process $S(t)$ represents the aggregate claim payments up to time t. Then the overall stochastic process U shows the net profit of this insurance policy. In our experiments, we set Poisson jump density $\lambda = 0.8$ and use uniform distribution $Uni(5, 10)$ as jump distribution. We choose $u = 15$ and $c = 4.5$. Recall durability query examples in Section 1, durability queries in financial domains can be used for revenue projection, risk management, and financial product design.

(3) Recurrent Neural Networks: As shown in Figure 5, we train a Recurrent Neural Networks (RNN) with Long-Short Term Memory (LSTM) and Mixture Density Network (MDN) [7] using Google's 5-year daily stock prices from 2015 to 2020. The LSTM-RNN-MDN structure has proven its success at many real-life tasks of probabilistically modeling and generating sequence data: e.g., language models [5], speech recognition [27], hand-writing analysis [28], and music composition [19]. In our network, we use two stacked RNN layers, 256 LSTM units per RNN layer, and a 2-dimensional mixture layer with 5 mixtures. During training phase, we trained the model for sequence length of 50, and for 100 epochs with a batch size of 32. Such learning-based black-box model demonstrates the general applicability of MLSS and of durability queries.

Evaluation Metric. We evaluate the performance of different methods using the following two metrics: total number of simulation steps (invocations of simulation procedure $\mathfrak{g}$) and total simulation time. In our experiments, we run sampling procedures until the estimation satisfies a given quality target. Specifically, we use two quality measurements throughout our experiments:

(1) Confidence Interval: Confidence interval (CI) is a statistical measurement for point estimates. It shows how likely (or how confident) that the true parameter is in the proposed range. There is no universal formula to construct CI for an arbitrary estimator. However, if a point estimator $\hat{\mu}$ takes the form of the mean of n independent and identically distributed (i.i.d.) random variables with equal expectation μ, then by the Central Limit Theorem and Normal Approximation, an approximate 1-α CI of μ can be constructed by: $[\hat{\mu} - z_{\alpha/2}\sqrt{\sigma^2/n}, \hat{\mu} + z_{\alpha/2}\sqrt{\sigma^2/n}]$, where $z_{\alpha/2}$ is the Normal critical value with right-tail probability $\alpha/2$, and σ^2 is the variance of estimate. By default, to obtain reliable query answers, we require that all estimations should have a 1% CI with 95% confidence level (i.e., $z_{\alpha/2} = 1.96$). Unfortunately, the standard CI, as in the above equation, has a limitation: when the true probability μ is very close to 0 or 1, where the Normal Approximation assumption does not hold, the CI guarantee would break. Hence, we also consider another quality measurement below for extreme cases.

Table 2: Query settings on different models.

Query Type	Medium (s, β)	Small (s, β)	Tiny (s, β)	Rare (s, β)
Queue Model	500, 20	500, 26	500, 40	500, 45
CPP Model	500, 300	500, 350	500, 450	500, 500
RNN Model	-	200, 1550	200, 1600	-

Table 3: Query answer comparisons on Queue Model. Results are averaged over 100 runs with standard deviation.

Query Type	Medium	Small	Tiny	Rare
SRS	17.2%±0.5%	5.1%±0.5%	0.15%±0.03%	0.04%±$2e^{-5}$
MLSS	17.9%±0.4%	5.5%±0.5%	0.17%±0.02%	0.04%±$3e^{-5}$

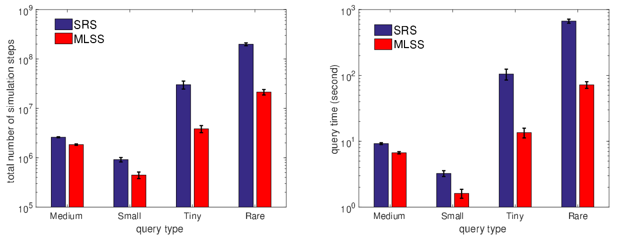
Table 4: Query answer comparisons on CPP Model. Results are averaged over 100 runs with standard deviation.

Query Type	Medium	Small	Tiny	Rare
SRS	15.5%±0.5%	5.3%±0.5%	0.24% ±0.02%	0.03%±$3e^{-5}$
MLSS	15.6%±0.4%	5.3%±0.5%	0.26%±0.01%	0.03%±$4e^{-5}$

(2) Relative Error: Relative Error (RE) measures the variance (of estimate) as a relative ratio to the true probability, defined as follows: $RE = \sqrt{\sigma^2}/\mu$, where μ is the true probability and σ^2 is the variance of estimate. This is not feasible to calculate directly in practice, since we do not know the true probability μ before the query. But in practice, we can roughly estimate the ground truth probability, and use that to fairly compare the RE ratio among different methods. By default, we require that all estimations should have a low relative error at 10%. Unlike CI, RE is widely applicable to any scenario.

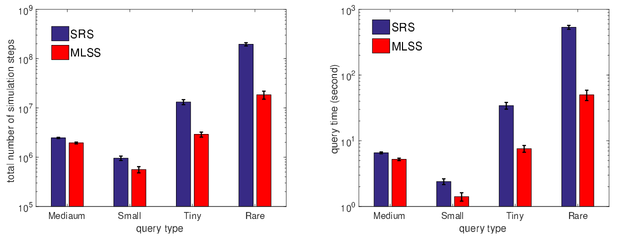
In sum, throughout the experiment section, we evaluate durability queries with different ground truth probabilities. For queries that have small-to-moderate probability (i.e., > 0.05), we use CI as the quality measure. For queries that have tiny probability (i.e., 10^{-4} to 10^{-2}), we use RE as the alternative measure.

Implementation Details. All stochastic temporal models and proposed solutions were implemented in Python3. More specifically, for neural network's construction and training, we use Keras [14] (back end by TensorFlow [1]). Unless otherwise stated, for MLSS (s-MLSS and g-MLSS), to limit the number factors influencing performance, by default we fix the splitting ratio $r = 3$ and use "balanced-growth" level-partition plans (recall discussions in Section 5.1), which are obtained by manual tuning given the number of partitions. Effectiveness of the adaptive greedy partition strategy will be examined separately in Section 6.3. For durability queries, we use the query condition in the form $z(x_t) \geq \beta$ and the simple value function $f(x_t) = \min\{z(x_t)/\beta, 1\}$ as we introduced in Section 3. Recall that $z(\cdot)$ is a real-valued evaluation of a state and β is a user-specified value threshold. For Queue model, $z(\cdot)$ evaluates the state by returning the number of customers in Queue 2; for CPP model, it is the value of $U(t)$, and for RNN model, $z(\cdot)$ returns the (simulated) stock price at a given state. All experiments were performed on a Linux machine with two Intel Xeon E5-2640 v4 2.4GHz processor with 256GB of memory.



(1) Simulation steps **(2) Query time**

Figure 6: Query efficiency on Queue Model



(1) Simulation steps **(2) Query time**

Figure 7: Query efficiency on CPP Model

6.1 MLSS vs. SRS

In this section, we comprehensively compare the performance of MLSS and SRS. Note that under the settings of experiments in this particular section, "level skipping" will not occur, so g-MLSS is equivalent to s-MLSS. Therefore, we do not distinguish between s-MLSS and g-MLSS in this section, and just use the term MLSS for simplicity to compare with the baseline. In Section 6.2, we modify the processes to make them more volatile (so "level skipping" will happen) and further evaluate the performance of g-MLSS. For each stochastic temporal model, we design four types of durability queries: Medium, Small, Tiny and Rare, denoting the quantity of the (estimated) true answer probability of the queries. Detailed query parameters are summarized in Table 2.

Estimations and Overall Efficiency. We first demonstrate the answer quality, i.e., unbiasedness, of MLSS. For each model and for each type of query, we repeatedly run SRS and MLSS 100 times, respectively, and average the returned answers along with empirical standard deviations. Results are summarized in Tables 3 (Queue Model), 4 (CPP Model) and, 5 (RNN Model). As shown in these tables, the answers (hitting probability) returned by SRS and MLSS, on all types of queries and on all models, are essentially the same. Even though they are not identical, the differences are well within the standard deviation. This finding confirmed our analysis and proof in Section 3 about MLSS's unbiased estimation.

Next, let us compare the query efficiency between SRS and MLSS. We time the query until its answer (estimation) achieves certain quality target. As shown in Figure 6 (Queue Model) and Figure 7 (CPP Model), MLSS generally runs significant faster than SRS (note the log scale on y-axis). For Medium and Small queries, we can see a 40% to 60% query time reduction brought by MLSS. For Tiny and Rare queries, MLSS runs 10x faster than SRS, without loss of answer quality. As discussed earlier in Section 3.1, the main advantage of MLSS to SRS is the ability to focus and encourage simulations that move towards the target. This property is especially helpful for those durability queries with lower probability, since MLSS can better distribute simulation efforts to promising paths hitting the target, instead of blindly wasting time on those failure paths (which would be a large portion of the total) as SRS did. We observe similar query efficiency improvement on the more complex RNN model. In Table 5, for Small and Tiny queries (which are more commonly asked in practice) on RNN model, there is a roughly 80% to an order-of-magnitude query time reduction provided by MLSS.

Table 6: Performance comparison between s-MLSS and g-MLSS on temporal process with volatile values changes.

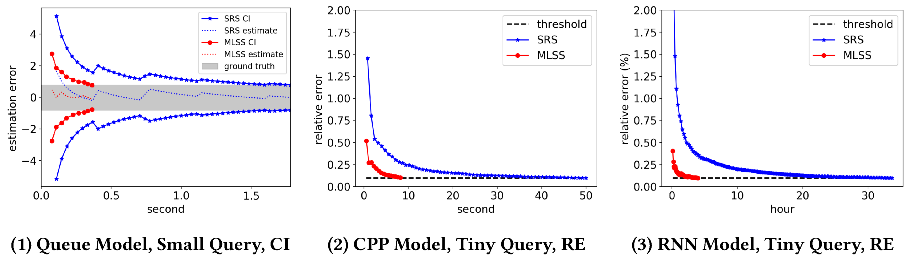
	Volatile CPP		Volatile Queue	
	Tiny Query ($s:500, \beta:700$)	Rare Query ($s:500, \beta:1000$)	Tiny Query ($s:500, \beta:65$)	Rare Query ($s:500, \beta:75$)
SRS	2.2%±1.5%	0.1%±0.2%	1.7%±0.9%	0.3%±0.26%
s-MLSS	1.1%±0.8%	0.05%±0.07%	1.2%±0.5%	0.2%±0.11%
g-MLSS	2.1%±1.2%	0.09%±0.1 %	1.7%±0.5%	0.3%±0.17%

Overall, MLSS clearly surpasses SRS across different models and on different types of commonly asked durability queries in practice, providing query speedup from 40% up to an order of magnitude, without sacrificing answer quality. It is also worth mentioning that MLSS is best suited for Tiny and Rare queries, and may not provide much benefit for larger queries. If the target is relatively easy to reach (which corresponds to large probability), the splitting behavior of MLSS would bring little benefit and may result in unnecessary overhead. Hence, in later sections, we would focus our discussions/evaluations more on Tiny and Rare queries, which are also the commonly asked durability queries in practice.

Query Performance over Time. To take a closer look at query performance comparison of MLSS and SRS, we monitor query answers and its quality (CI or RE) over time, and plot the convergence of estimations on single run of MLSS and SRS, respectively. See Figure 8 for details. In Figure 8(1), we run a Small query on Queue model and use CI as estimation quality measure. For better illustration, CI intervals are interpreted as percentage to the true probability such that it will be centered at 0. The grey ribbon in the plot shows the desired region for a reliable estimate (true probability with 1% CI). Symmetric red lines and blue lines demonstrate how CIs of MLSS and SRS converge over time. Red dotted line and blue dotted line are the estimate of MLSS and SRS over time. It is clear that MLSS converges faster than SRS on estimation quality. On the other hand, we can also see that the estimates (red dotted line and blue dotted line) from MLSS and SRS are always nicely contained by its corresponding CI, showing the statistical guarantees brought by CI. We observe similar behaviors on CPP model (Figure 8(2)) and RNN model (Figure 8(3)). Here we use run Tiny queries on these two models and use RE as quality measure. Similarly, the time that MLSS needs for a reliable estimate (10% RE, dashed line in the plot) is significantly shorter than that of SRS.

6.2 Simple MLSS vs. General MLSS

s-MLSS works well in practice if the underlying process satisfies the no level-skipping assumption, as demonstrated in previous sections. To show the limitation of s-MLSS and the generality of g-MLSS, we consider level skipping by experimenting with new temporal processes based on CPP model and Queue model with impulse value jumps between consecutive time instants. More specifically, when $t > 0.8s$ we introduce large value increase of (200 for CPP, and 5 for Queue) with small probabilities (0.005 for CPP and 0.2 for Queue). We refer to these two processes as Volatile CPP and Volatile Queue.

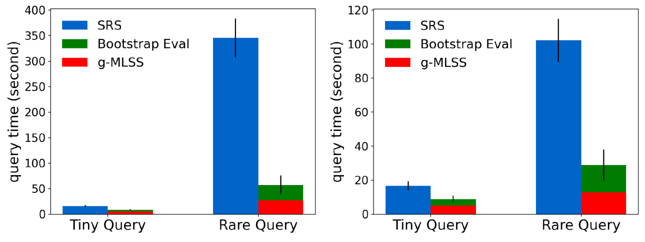


(1) Queue Model, Small Query, CI (2) CPP Model, Tiny Query, RE (3) RNN Model, Tiny Query, RE

Figure 8: Query answer quality over time.

Table 5: Query performance (single run) on RNN Model.

Query Type	Small	Tiny
SRS	2.6% 3.8 hours 1,009,431 steps	0.51% 33.7 hours 7,262,735 steps
MLSS	1.9% 0.75 hour 196,913 steps	0.45% 3.9 hours 804,035 steps



(1) Query time, Volatile CPP (2) Query time, Volatile Queue

Figure 9: g-MLSS query efficiency on models with volatile value changes.

Estimation and Overall Efficiency. First, we test the unbiasedness of our approach. we fix the simulation budget (i.e., 50000 invocations to the simulation procedure) and compare average estimations with empirical standard deviation based on estimates obtained from 100 independent runs. Results are summarized in Table 6. It is clear that, with the existence of level skipping, s-MLSS gives wrong estimates. In contrast, g-MLSS still provides unbiased estimation by gracefully handling level-skipping paths, and it has higher precision (smaller standard deviation) compared to SRS under the same simulation budget.

Second, we evaluate the query efficiency of g-MLSS. Figure 9 shows the performance of g-MLSS on Volatile CPP and Volatile Queue. Recall from our discussions in Section 4 that we do not have an analytical expression for g-MLSS variance; instead, we implement bootstrap sampling to empirically estimate it for evaluating the stopping condition. Hence, the bootstrap evaluation time is also counted towards the total query time (shown in green in the plot). Overall, as presented in Figure 9, g-MLSS beats SRS by a large margin. Especially for Rare, we can see nearly 80% improvement on both models. Focusing on the breakdown of total query time, we can see that the bootstrap evaluation takes up a large portion (more than 50%) of the query time. Our currently implementation of bootstrapping is rather unoptimized; with more sophisticated implementations, we expect g-MLSS to still have plenty of room for further efficiency improvement.

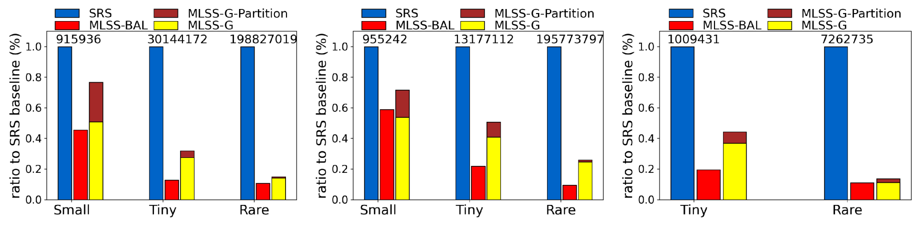
6.3 MLSS Optimization

In previous sections, we have shown the dominance of MLSS (both s-MLSS and g-MLSS) over SRS across a variety of models and query types. We now focus more on MLSS itself, and investigate how sampling parameters of MLSS affect its overall efficiency and how to efficiently fine-tune MLSS in practice. Due to space limit, we present some experiments (on the trade-off between splitting ratio/number of level partitions and the overall efficiency) into the extended version of the paper [24]. Here, we focus on experimentally validating our greedy level partition strategy (Algorithm 1). The baseline partition plans for comparison are the corresponding "balanced-growth" partition plans, which are obtained via manual tuning under the balanced-growth guideline and have the optimal number of levels. In the following, we will refer to MLSS using such pre-tuned balanced-growth partition plans as MLSS-BAL; we do not charge the cost of manual tuning to running MLSS-BAL.

Greedy Level Partitions for s-MLSS. Figure 10 shows the effectiveness of the proposed greedy strategy on s-MLSS (in terms of the resulting overall running time to meet a given quality). For better visualization, we normalize all running times as ratios relative to the SRS baseline; hence in all plots, SRS is shown as the blue bars with ratio 1. We also show the total number of simulation steps on top of each bar. Red bars represent the running times of MLSS-BAL (recall that they do not include the cost of fine-tuning to find their balance-growth partition plans). Yellow bars (MLSS-G) and brown bars (MLSS-G-Partition) together reflect the total running times of MLSS using the greedy algorithm, with MLSS-G-Partition representing the search overhead of greedy algorithm. Overall, across all three models and different types of queries, the greedy algorithm is able to find a partition plan comparable to the manually tuned "balanced growth" plan—the cost of MLSS-G is not so far away from MLSS-BAL,[1] and is still significantly lower than SRS with a 60% to an order-of-magnitude improvement. The search overhead (MLSS-G-Partition) is 10% to 30% of the total cost; importantly, overhead seems lower for harder cases of Tiny and Rare, making greedy adaption an attractive approach in practice.

Greedy Level Partitions for g-MLSS. We further test the greedy strategy on g-MLSS under volatile stochastic processes in Figure 11. Again, we use (a rather unoptimized implementation of) bootstrapping to estimate the variance of g-MLSS in order to test the stopping condition; that cost is shown as a green bar. Overall, the total cost of g-MLSS with greedy adaptation (including greedy search overhead and bootstrapping overhead on top of simulation time) is lower than SRS in most cases acceptably close to MLSS-BAL, which has the benefit of pre-tuning. Compared to the SRS baseline, our fully automated approach has a ~20% speedup on Tiny query and up to 80% improvement on Rare query on both models.

[1]As can be seen in Figures 10(2) and (3), sometimes our greedy strategy can discover a even better partition plan than MLSS-BAL. This should not be surprising because the optimality of plans with balanced growth is based on certain assumptions (Section 5) that may not hold in practice. Nonetheless, MLSS-BAL is a reasonable yardstick for comparison because the balanced growth guideline is, to the best of our knowledge, the only one that offers some theoretical guarantee of optimality.

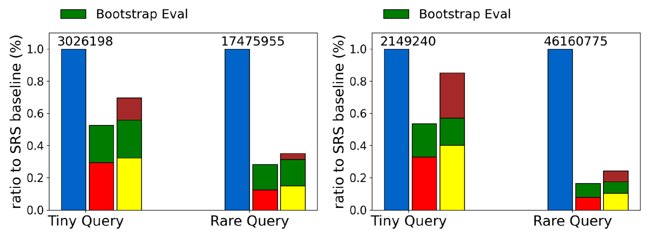


(1) Queue Model (2) CPP Model (3) RNN Model (single run)

Figure 10: Efficiency of Greedy Level Partitions with s-MLSS.

Table 7: Running times of MLSS and SRS on Queue/CPP model inside PostgreSQL. Time usage in second.

Queue Model	Medium	Small	Tiny	Rare
SRS	6.6	2.4	146	1111
MLSS	6.1	1.3	23	79
CPP Model	Medium	Small	Tiny	Rare
SRS	10.2	3.7	112	3012
MLSS	8.8	2.5	27	173



(1) Volatile Queue (2) Volatile CPP

Figure 11: Efficiency of Greedy Level Partitions on volatile temporal processes with g-MLSS.

In sum, considering that the greedy strategy does not need any information in advance and can automatically search for partition plans, it is a reasonable approach to try in practice if users do not have related knowledge of the model or the query.

6.4 Implementations inside DBMS

The database management system (DBMS) provides a single platform for not only data management, transformation, and querying, but also increasing machine learning support [8]. Predictive models, ranging from classic statistical models (i.e., Queue and CPP) to complex learning-based models, can be seamlessly encoded inside DBMS for data analytics; e.g., MCDB [34]. MLSS can also be straightforwardly integrated into a DBMS by implementing its sampler and estimator as stored procedures. In this section, we move the query answering pipeline inside a DBMS (PostgreSQL), including both the predictive models and query processing algorithm. More specifically, we use a database table for storing parameters of the procedure $\mathfrak{g}$ to allow step-by-step forward simulations and implement MLSS as stored procedure using Python Procedural Language. We repeat our experiments as in Section 6.1 and report results in Table 7. We see the advantage of MLSS over SRS as in earlier experiments; for example, we brought the running times of Rare queries from 0.3-0.8 hour required by SRS to under a few minutes. This demonstrates sufficient promises towards an end-to-end ML lifecycle inside DBMS: data ETL (Extract, Transform and Load), building predictive models, and efficiently answering durability queries based on predictions for various data analytics. Moreover, we can materialize sample paths generated from MLSS simulations as separate database tables, which can be further used for visualizations or other analysis.

7 RELATED WORK

The closest line of work to ours is query processing over probabilistic databases [17]: range search queries [11, 13, 56, 57], top-k queries [26, 32, 33, 49, 55, 62], join queries [12, 39] and skyline queries [47]. But there is a fundamental difference between these previous studies and our problem. In this paper, we consider query processing based on predictive models that predict future temporal data, where temporal dependence is not neglectable when modelling data uncertainty. As a comparison, previous work on probabilistic databases mainly focuses on the static (snapshot) data, where data uncertainty is considered independently for individuals.

Another similar line of work is MCDB and its variants [3, 9, 34, 48]. Unlike probabilistic databases, MCDB does not make strong assumptions about uncertainty independence, but generally embodies data uncertainty with user-defined variable generation (VG) functions. The use of VG functions is analogous to the way that we handle uncertain temporal data with predictive models. Moreover, MCDB's solutions are simulation-based too. The only difference is that our work devise novel sampling procedure to improve sampling efficiency while MCDB focuses on making standard Monte Carlo simulations run faster inside database management system. In [21], authors used Markov Chains to present uncertain spatio-temporal data and studied how to answer probabilistic range queries. However, their solutions are specific to Markov Chains and requires the transition probability matrix as a priori information. In contrast, our techniques are generally applicable to a variety of predictive models, and are largely independent of the underlying model itself.

Regarding durability queries, there are several papers exploring the notions of durability on temporal data. In [23, 43, 44, 59], authors consider durability as a fraction of times (that satisfies certain conditions) over a (temporal) sequence of snapshot data, and answer queries to return the top k objects with highest durability. In [35, 36, 63], authors view durability as the length of time interval. They proposed that, on the two-dimensional space coordinating by durability and data values, skyline queries can discover interesting insights or facts from temporal data that are robust and consistent. Though in different forms, these papers studied durability on existing historical data, which is certain. To the best of our knowledge, our work is among the first to extend the notion of durability into the future, where data can only be probabilistic.

ACKNOWLEDGEMENTS

This work was supported by NSF grants IIS-1718398, IIS-1814493, CCF-2007556, and a grant from the Knight Foundation. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. http://tensorflow.org/ Software available from tensorflow.org.

[2] Hansjörg Albrecher, Jean-François Renaud, and Xiaowen Zhou. 2008. A Lévy insurance risk process with tax. *Journal of Applied Probability* 45, 2 (2008), 363–375.

[3] Subi Arumugam, Fei Xu, Ravi Jampani, Christopher Jermaine, Luis L Perez, and Peter J Haas. 2010. MCDB-R: Risk analysis in the database. *Proceedings of the VLDB Endowment* 3, 1-2 (2010), 782–793.

[4] Shahab Basiri, Esa Ollila, and Visa Koivunen. 2015. Robust, scalable, and fast bootstrap method for analyzing large scale data. *IEEE Transactions on Signal Processing* 64, 4 (2015), 1007–1017.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. A neural probabilistic language model. *Journal of machine learning research* 3, Feb (2003), 1137–1155.

[6] Kurt Binder, Dieter Heermann, Lyle Roelofs, A John Mallinckrodt, and Susan McKay. 1993. Monte Carlo simulation in statistical physics. *Computers in Physics* 7, 2 (1993), 156–157.

[7] Christopher M Bishop. 1994. Mixture density networks. (1994).

[8] Matthias Boehm, Arun Kumar, and Jun Yang. 2019. Data Management in Machine Learning Systems. *Synthesis Lectures on Data Management* 11, 1 (2019), 1–173.

[9] Zhuhua Cai, Zografoula Vagena, Luis Perez, Subramanian Arumugam, Peter J. Haas, and Christopher Jermaine. 2013. Simulation of Database-Valued Markov Chains Using SimSQL. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data* (New York, New York, USA) *(SIGMOD '13)*. Association for Computing Machinery, New York, NY, USA, 637–648. https://doi.org/10.1145/2463676.2465283

[10] Frédéric Cérou and Arnaud Guyader. 2007. Adaptive multilevel splitting for rare event analysis. *Stochastic Analysis and Applications* 25, 2 (2007), 417–443.

[11] Reynold Cheng, Dmitri V Kalashnikov, and Sunil Prabhakar. 2003. Evaluating probabilistic queries over imprecise data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 551–562.

[12] Reynold Cheng, Sarvjeet Singh, Sunil Prabhakar, Rahul Shah, Jeffrey Scott Vitter, and Yuni Xia. 2006. Efficient join processing over uncertain data. In *Proceedings of the 15th ACM international conference on Information and knowledge management*. 738–747.

[13] Reynold Cheng, Yuni Xia, Sunil Prabhakar, Rahul Shah, and Jeffrey Scott Vitter. 2004. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*. 876–887.

[14] François Chollet et al. 2015. keras.

[15] Robert B Cooper. 1981. Queueing theory. In *Proceedings of the ACM'81 conference*. 119–122.

[16] David Roxbee Cox and Hilton David Miller. 1977. *The theory of stochastic processes*. Vol. 134. CRC press.

[17] Nilesh Dalvi, Christopher Ré, and Dan Suciu. 2009. Probabilistic databases: diamonds in the dirt. *Commun. ACM* 52, 7 (2009), 86–94.

[18] Pieter-Tjerk De Boer, Dirk P Kroese, Shie Mannor, and Reuven Y Rubinstein. 2005. A tutorial on the cross-entropy method. *Annals of operations research* 134, 1 (2005), 19–67.

[19] Douglas Eck and Juergen Schmidhuber. 2002. *A First Look at Music Composition Using LSTM Recurrent Neural Networks*. Technical Report.

[20] Bradley Efron and Robert J Tibshirani. 1994. *An introduction to the bootstrap*. CRC press.

[21] Tobias Emrich, Hans-Peter Kriegel, Nikos Mamoulis, Matthias Renz, and Andreas Zufle. 2012. Querying uncertain spatio-temporal data. In *2012 IEEE 28th international conference on data engineering*. IEEE, 354–365.

[22] Per Fauchald and Torkild Tveraa. 2003. Using first-passage time in the analysis of area-restricted search and habitat selection. *Ecology* 84, 2 (2003), 282–288.

[23] Junyang Gao, Pankaj K Agarwal, and Jun Yang. 2018. Durable top-k queries on temporal data. *Proceedings of the VLDB Endowment* 11, 13 (2018), 2223–2235.

[24] Junyang Gao, Yifan Xu, Pankaj K. Agarwal, and Jun Yang. 2021. Efficiently Answering Durability Prediction Queries (Technical Report Version). https://arxiv.org/abs/2103.12887

[25] Marnix Joseph Johann Garvels. 2000. The splitting method in rare event simulation. (2000).

[26] Tingjian Ge, Stan Zdonik, and Samuel Madden. 2009. Top-k queries on uncertain data: on score distribution and typical answers. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 375–388.

[27] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. 2013. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6645–6649.

[28] Alex Graves and Jürgen Schmidhuber. 2009. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*. 545–552.

[29] Geoffrey Grimmett, Geoffrey R Grimmett, David Stirzaker, et al. 2001. *Probability and random processes*. Oxford university press.

[30] Theodore Edward Harris. 1964. The theory of branching process. (1964).

[31] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[32] Ming Hua, Jian Pei, and Xuemin Lin. 2011. Ranking queries on uncertain data. *The VLDB Journal* 20, 1 (2011), 129–153.

[33] Ming Hua, Jian Pei, Wenjie Zhang, and Xuemin Lin. 2008. Efficiently answering probabilistic threshold top-k queries on uncertain data. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 1403–1405.

[34] Ravi Jampani, Fei Xu, Mingxi Wu, Luis Leopoldo Perez, Christopher Jermaine, and Peter J Haas. 2008. MCDB: a monte carlo approach to managing uncertain data. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 687–700.

[35] Bin Jiang and Jian Pei. 2009. Online interval skyline queries on time series. In *2009 IEEE 25th International Conference on Data Engineering*. IEEE, 1036–1047.

[36] Xiao Jiang, Chengkai Li, Ping Luo, Min Wang, and Yong Yu. 2011. Prominent streak discovery in sequence data. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1280–1288.

[37] Michael I Jordan. 1997. Serial order: A parallel distributed processing approach. In *Advances in psychology*. Vol. 121. Elsevier, 471–495.

[38] Herman Kahn and Theodore E Harris. 1951. Estimation of particle transmission by random sampling. *National Bureau of Standards applied mathematics series* 12 (1951), 27–30.

[39] Benny Kimelfeld and Yehoshua Sagiv. 2007. Maximally joining probabilistic data. In *Proceedings of the twenty-sixth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 303–312.

[40] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I Jordan. 2014. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B: Statistical Methodology* (2014), 795–816.

[41] Edward D Lazowska, John Zahorjan, G Scott Graham, and Kenneth C Sevcik. 1984. *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc.

[42] Pierre L'Ecuyer, Valérie Demers, and Bruno Tuffin. 2006. Splitting for rare-event simulation. In *Proceedings of the 2006 winter simulation conference*. IEEE, 137–148.

[43] Mong Li Lee, Wynne Hsu, Ling Li, and Wee Hyong Tok. 2009. Consistent top-k queries over time. In *International Conference on Database Systems for Advanced Applications*. Springer, 51–65.

[44] U Leong Hou, Nikos Mamoulis, Klaus Berberich, and Srikanta Bedathur. 2010. Durable top-k search in document archives. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*.

[45] C Newell. 2013. *Applications of queueing theory*. Vol. 4. Springer Science & Business Media.

[46] Matthew O'Kelly, Aman Sinha, Hongseok Namkoong, Russ Tedrake, and John C Duchi. 2018. Scalable end-to-end autonomous vehicle testing via rare-event simulation. In *Advances in Neural Information Processing Systems*. 9827–9838.

[47] Jian Pei, Bin Jiang, Xuemin Lin, and Yidong Yuan. 2007. Probabilistic skylines on uncertain data. In *Proceedings of the 33rd international conference on Very large data bases*. Citeseer, 15–26.

[48] Luis L Perez, Subi Arumugam, and Christopher M Jermaine. 2010. Evaluation of probabilistic threshold queries in MCDB. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. 687–698.

[49] Christopher Re, Nilesh Dalvi, and Dan Suciu. 2007. Efficient top-k query evaluation on probabilistic data. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 886–895.

[50] Sidney Redner. 2001. *A guide to first-passage processes*. Cambridge University Press.

[51] Reuven Y Rubinstein. 1997. Optimization of computer simulation models with rare events. *European Journal of Operational Research* 99, 1 (1997), 89–112.

[52] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1985. *Learning internal representations by error propagation*. Technical Report. California Univ San Diego La Jolla Inst for Cognitive Science.

[53] Albert N Shiryaev. 1999. *Essentials of stochastic finance: facts, models, theory*. Vol. 3. World scientific.

[54] Robert H Shumway and David S Stoffer. 2017. *Time series analysis and its applications: with R examples*. Springer.

[55] Mohamed A Soliman, Ihab F Ilyas, and Kevin Chen-Chuan Chang. 2007. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*. IEEE, 896–905.

[56] Yufei Tao, Reynold Cheng, Xiaokui Xiao, Wang Kay Ngai, Ben Kao, and Sunil Prabhakar. 2005. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*, Vol. 5. Citeseer, 922–933.

[57] Yufei Tao, Xiaokui Xiao, and Reynold Cheng. 2007. Range search on multidimensional uncertain data. *ACM Transactions on Database Systems (TODS)* 32, 3 (2007), 15–es.

[58] Manuel Villén-Altamirano and José Villén-Altamirano. 1994. RESTART: a straightforward method for fast simulation of rare events. In *Proceedings of Winter Simulation Conference*. IEEE, 282–289.

[59] Hao Wang, Yilun Cai, Yin Yang, Shiming Zhang, and Nikos Mamoulis. 2013. Durable queries over historical time series. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (2013), 595–607.

[60] GA Whitmore. 1986. First-passage-time models for duration data: regression structures and competing risks. *Journal of the Royal Statistical Society: Series D (The Statistician)* 35, 2 (1986), 207–219.

[61] Yifan Xu. 2012. First exit times of compound Poisson processes with parallel boundaries. *Sequential Analysis* 31, 2 (2012), 135–144.

[62] Ke Yi, Feifei Li, George Kollios, and Divesh Srivastava. 2008. Efficient processing of top-k queries in uncertain databases. In *2008 IEEE 24th International Conference on Data Engineering*. IEEE, 1406–1408.

[63] Gensheng Zhang, Xiao Jiang, Ping Luo, Min Wang, and Chengkai Li. 2014. Discovering general prominent streaks in sequence data. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 2 (2014), 1–37.